SESSION

GRAPH BASED METHODS AND APPLICATIONS + RELATED ISSUES

Chair(s)

TBA

Asynchronous P systems for uniform graph partitioning and related graph problems

Kota Hasuo and Akihiro Fujiwara
Graduate School of Computer Science and Systems Engineering, Kyushu Institute of Technology
Iizuka, Fukuoka, 820-8502, Japan

Abstract— *In the present paper, we consider fully asynchronous parallelism in membrane computing, and propose asynchronous P systems for three graph problems, which are the graph partitioning, the maximum cut, and the dominating set. We first propose an asynchronous P system that solves uniform graph partitioning for a graph with n nodes, and show that the proposed P system works in $O(n^2 \cdot 2^n)$ sequential steps or $O(n^2)$ parallel steps using $O(n^3)$ kinds of objects. We also propose asynchronous P systems, which works in a polynomial number of parallel steps, for the other graph problems.*

Keywords: asynchronous membrane computing, graph partitioning, maximum cut, dominating set

1. Introduction

Membrane computing, which is a representative example of natural computing, is a computational model inspired by the structures and behaviors of living cells. In the initial study on membrane computing, a basic feature of the membrane computing was introduced by Păun[1] as a P system.

The P system and most variants are proved to be universal [2]. In addition, the models deliver superior performance on the problems that need exponential computation time such as NP-complete or NP-hard problems, and various P systems [3], [4], [5], [6], [7] have been proposed for solving NP problems. However, synchronous application of evolution rules is assumed on the above P systems with the maximal parallelism, which is a main feature of the P systems. The maximal parallelism means that all applicable rules in all membranes are applied synchronously.

On the other hand, there is obvious asynchronous parallelism in the cell biochemistry. The asynchronous parallelism means that all objects may react on rules with different speed, and evolution rules are applied to objects independently. Since all objects in a living cell basically work in asynchronous manner, the asynchronous parallelism must be considered to make P system more realistic model.

For considering the asynchronous parallelism, a number of P systems have been proposed [8], [9], [10], [11]. For example, two asynchronous P systems [9] have been proposed for solving SAT and Hamiltonian cycle problem. For another example, the asynchronous P systems [11] have been proposed for solving four graph problems, which are minimum coloring, maximum independent set, minimum vertex cover, and maximum clique. The above P systems solve the NP problems in polynomial number of parallel steps.

In the present paper, we propose asynchronous P systems for five graph problems, which are the uniform graph partitioning, the (k, v)-balanced partitioning problem, the maximum cut, the dominating set problem and connected dominating set problem. The five problems are well-known NP hard graph problems.

We first propose an asynchronous P system for the uniform graph partitioning for a graph with n nodes. The proposed P system solves the problem in $O(n^2 \cdot 2^n)$ sequential steps or $O(n^2)$ parallel steps using $O(n^3)$ kinds of objects and evolution rules of size $O(n^5)$.

We next propose an asynchronous P system that solves (k, v)-balanced partitioning problem for a graph with n nodes, and show that the proposed P system works in $O(k^n \cdot n^2)$ sequential steps or $O(n^2)$ parallel steps using $O(n^3)$ kinds of objects.

We also propose asynchronous P systems that solves the maximum cut, the dominating set, and the connected domination set for a graph with n nodes, and show that all of the three proposed P systems work in $O(n^2 \cdot 2^n)$ sequential steps or $O(n^2)$ parallel steps using $O(n^3)$ types of objects.

2. Preliminaries

2.1 Computational model

The P system [3] mainly consists of membranes and objects. A membrane is a computing cell, in which independent computations are executed in parallel, and may contain objects and other membranes. In other words, the membranes form nested structures. In the present paper, each membrane is denoted using a pair of square brackets, and the number on the right-hand side of each right-hand bracket denotes a label of the corresponding membrane. An object in the P system is a memory cell, in which each data is stored, and can divide, dissolve, and pass through membranes. In the present paper, each object is denoted by finite strings over a given alphabet, and is contained in one of the membranes.

Now, we formally define a P system Π and the sets used in the system as follows.

$\Pi = (O, \mu, \omega_1, \omega_2, \cdots, \omega_m, R_1, R_2, \cdots, R_m, i_{in}, i_{out})$

O: O is the set of all objects used in the system.
μ: μ is membrane structure that consists of m membranes. Each membrane in the structure is labeled with an integer i $(1 \leq i \leq m)$. In addition, a membrane labeled 1, which is called the skin membrane, is the outermost membrane, and the skin membrane contains all of the other membranes.
ω_j: ω_j is a set of objects initially contained in the membrane labeled j.
R_j: R_j is a set of evolution rules that are applicable to objects in the membrane labeled j.
i_{in}: i_{in} is a label of the input membrane.
i_{out}: i_{out} is a label of the output membrane.

In the present paper, we assume that input objects are given from the outside region into the skin membrane, and computation is started by applying evolution rules. We also assume that output objects are sent out from the skin membrane to the outside region.

In membrane computing, several types of rules are proposed. In the present paper, we consider five basic rules of the following forms in [3].

(1) Object evolution rule: $[\alpha]_h \rightarrow [\beta]_h$
where h is a label of the membrane, and $\alpha, \beta \in O$. Using the rule, an object α evolves into another object β. (We omit the brackets in each evolution rule for cases that a corresponding membrane is obvious.)

(2) Send-in communication rule: $\alpha[\ ]_h \rightarrow [\beta]_h$
where h is a label of the membrane, and $\alpha, \beta \in O$. Using the rule, an object α is sent into the membrane, and can evolve into another object β.

(3) Send-out communication rule: $[\alpha]_h \rightarrow [\ ]_h\beta$
where h is a label of the membrane, and $\alpha, \beta \in O$. Using the rule, an object α is sent out of the membrane, and can evolve into another object β.

(4) Dissolution rule: $[\alpha]_h \rightarrow \beta$
where h is a label of the membrane, and $\alpha, \beta \in O$. Using the rule, the membrane, which contains object α, is dissolved, and the object can evolve into another object β. (Note that the skin membrane cannot be dissolved.)

(5) Division rule: $[\alpha]_h \rightarrow [\beta]_h[\gamma]_h$
where h is a label of the membrane, and $\alpha, \beta, \gamma \in O$. Using the rule, the membrane, which contains object α, is divided into two membranes that contain objects β and γ.

In addition, membrane computing has two features, which are maximal parallelism and non-determinism. Maximal parallelism means that all applicable rules are applied in parallel. (Membrane computing is considered as a kind of parallel computing from the feature.) On the other hand, non-determinism means that applicable rules are non-deterministically chosen in case that there are several possibilities of the applicable rules.

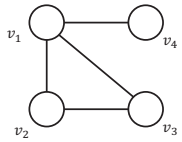


Fig. 1: An input graph

2.2 Asynchronous P systems

We describe differences between an asynchronous P system, which is considered in the paper, and conventional P system in this subsection. In the conventional P system, evolution rules are applied with maximal parallel manner, which we described in the above subsection. On the other hand, evolution rules are applied with asynchronous parallel manner, i.e., at least one of applicable evolution rules is applied, in each step of the computation in the asynchronous P system. The reason why we assume asynchronous parallelism in this paper is based on the fact that each of living cells acts independently and asynchronously. Since conventional P system ignores the asynchronous feature of living cell, and asynchronous P system is more realistic computation models for cell activities.

In the asynchronous P system, all evolution rules can be applied completely in parallel, which is the same as conventional P system, or all evolution rules can be applied sequentially. We define the numbers of steps executed in the asynchronous P system in maximal parallel manner as the number of parallel steps. On the other hand, we also define the numbers of steps in case that applicable evolution rules are applied sequentially as the number of sequential steps. The numbers of parallel and sequential steps means the best and worst case complexities for the asynchronous P system, respectively. In addition, the proposed asynchronous P system must be guaranteed to output a correct solution in any asynchronous executions.

3. Uniform graph partitioning

3.1 Input and output

Given a graph $G = (V, E)$ such that $V = \{v_1, v_2, \cdots v_n\}$($n$ is even), the uniform graph partitioning is a partitioning of vertices into two subsets V' and V'' such that V' and V'' are equal size, and the sum of costs of connecting edges between two subsets is the minimum.

For example, given a graph in Figure 1, partitioning of V into two sets of vertices $V' = \{v_1, v_4\}$ and $V'' = \{v_2, v_3\}$ is an uniform graph partition of the graph.

In the present paper, the input of the uniform graph partitioning is a set of objects O_E, which is given below.

$$O_E = \{\langle e_{i,j}, W\rangle \mid 1 \le i \le n, 1 \le j \le n, W \in \{T, F\}\}$$

Each object $\langle e_{i,j}, W\rangle$ denotes an edge (v_i, v_j). If there exists an edge (v_i, v_j) in the input graph, W pis set to T, otherwise, W is set to F.

The output of the P system is denoted using a set of n objects given below.

$$O_S = \{\langle V_i, A\rangle \mid 1 \le i \le n, A \in \{1, 2\}\}$$

Each object $\langle V_i, A\rangle$ in O_S denotes an output for vertex v_i, and $A = 1$ if v_i is in set V', otherwise $A = 2$.

We assume that a set of input objects is given from the outside region into the skin membrane, and the output object is sent out from the skin membrane to the outside region.

3.2 An overview of asynchronous P system

We now describe an overview of an asynchronous P system for the uniform graph partitioning. The P system consists of inner and outer membranes, i.e. membrane structure of the P system is $[\ [\ ]_2\ ]_1$. The computation in the asynchronous P system mainly consists of the following 7 steps.

- Step 1: Move all input objects in the outer membrane into the inner membrane.
- Step 2: Create all possible partitionings of the graph by dividing the inner membranes repeatedly.
- Step 3: Check if each divided two sets are equal size, and compute a sum of the cost of edges whose endpoints are in the same set.
- Step 4: Send out the computed costs from the inner membranes to the outer membrane, and compute the maximum cost in the outer membrane.
- Step 5: Create all possible partitionings of the graph again by dividing the inner membranes repeatedly.
- Step 6: In each divided membrane, check if each partitioning of the graph is the uniform graph partitioning or not using the maximum cost, and dissolve all inner membrane if the membrane contains a partitioning that is not the uniform graph partitioning.
- Step 7: Dissolve one of the inner membranes, which includes the uniform graph partitioning, and send out the result from the outer membrane.

3.3 Details of asynchronous P system

We now show details of each step of the asynchronous P system for the uniform graph partitioning.

In Step 1, all input objects in the outer membrane are moved into the inner membrane. Step 1 is executed applying the following set of evolution rules.

(Evolution rules for the outer membrane)

$$\begin{aligned} R_{1,1} \;=\; & \{\langle e_{1,1}, F\rangle[\ ]_2 \to [\langle M_{2,1}\rangle\langle e_{1,1}, F\rangle]_2\} \\ & \cup\{\langle M_{i,j}\rangle\langle e_{i,j}, W\rangle[\ ]_2 \to [\langle M_{i+1,j}\rangle\langle e_{i,j}, W\rangle]_2 \\ & \quad \mid 1 \le i \le n, 1 \le j \le n, W \in \{T, F\}\} \\ & \cup\{\langle M_{n+1,j}\rangle \to \langle M_{1,j+1}\rangle \mid 1 \le j \le n\} \\ & \cup\{\langle M_{1,n+1}\rangle[\ ]_2 \to [\langle M_{1,n+2}\rangle]_2\} \end{aligned}$$

(Evolution rules for the inner membrane)

$$\begin{aligned} R_{2,1} \;=\; & \{[\langle M_{i,j}\rangle]_2 \to [\ ]_2\langle M_{i,j}\rangle \\ & \quad \mid 1 \le i \le n+1, 1 \le j \le n+1\} \\ & \cup\{[\langle M_{1,n+2}\rangle]_2 \to [\langle S_1\rangle]_2[\langle D\rangle]_2\} \end{aligned}$$

In the above evolution rules, object $\langle e_{1,1}, F\rangle$ starts the computation, and input objects, $\langle e_{i,j}, W\rangle$, are moved into the inner membrane using object $\langle M_{i,j}\rangle$. After all input objects, $\langle e_{i,j}, W\rangle$, are moved into the inner membrane, object $\langle M_{i,j}\rangle$ is changed into object $\langle M_{1,n+1}\rangle$. At the end of Step 1, object $\langle D\rangle$ and object $\langle S_1\rangle$ are created by division rules. The object $\langle D\rangle$ is used for the computation of Step 5, and object $\langle S_1\rangle$ triggers computation of Step 2.

In Step 2, all possible partitionings of vertices are created by dividing the inner membranes repeatedly. Step 2 is executed applying the following set of evolution rules. In the evolution rules, $\langle V_i, 1\rangle$ denotes that v_i is contained in V', and $\langle V_i, 2\rangle$ denotes that v_i is contained in V''.
(Evolution rules for inner membrane)

$$R_{2,2} = \{[S_i]_2 \to [\langle S_{i+1}\rangle\langle V_i, 1\rangle]_2[\langle S_{i+1}\rangle\langle V_i, 2\rangle]_2 \mid 1 \le i \le n\}$$

At the end of Step 2, 2^n membranes are created. In addition, object $\langle S_{n+1}\rangle$ is also created in each divided membrane, and the object triggers computation of Step 3.

In Step 3, in each divided membrane, each partitioning of graph is checked whether the divided two sets are equal size or not, and compute a sum of the cost of edges whose endpoints are in same set.

Step 3 is executed applying the following set of evolution rules.
(Evolution rules for the inner membrane)

$$\begin{aligned} R_{2,3} \;=\; & \{\langle S_{n+1}\rangle \to \langle B_1\rangle\langle 0\rangle\} \\ & \cup\{\langle B_i\rangle\langle V_i, 1\rangle\langle a\rangle \to \langle B_{i+1}\rangle\langle V_i, 1\rangle\langle a+1\rangle \\ & \quad \mid -n \le a \le n, 1 \le i \le n\} \\ & \cup\{\langle B_i\rangle\langle V_i, 2\rangle\langle a\rangle \to \langle B_{i+1}\rangle\langle V_i, 2\rangle\langle a-1\rangle \\ & \quad \mid -n \le a \le n, 1 \le i \le n\} \\ & \cup\{\langle B_{n+1}\rangle\langle a\rangle \to \langle E_{1,1}\rangle\langle 0, 1\rangle \mid a \le -1, 1 \le a\} \\ & \cup\{\langle B_{n+1}\rangle\langle 0\rangle \to \langle C_{1,1}\rangle\langle 0\rangle\} \\ & \cup\{\langle C_{i,j}\rangle\langle e_{i,j}, F\rangle \to \langle C_{i,j+1}\rangle \\ & \quad \mid 1 \le i \le n, 1 \le j \le n\} \end{aligned}$$

$$\cup\{\langle C_{i,j}\rangle\langle e_{i,j},T\rangle\langle V_i,l\rangle\langle V_j,m\rangle \rightarrow \langle C_{i,j+1}\rangle\langle V_i,l\rangle\langle V_j,m\rangle \mid 1\le i\le n, 1\le j\le n, 1\le l\le 2, 1\le m\le 2, l\ne m\}$$
$$\cup\{\langle C_{i,j}\rangle\langle e_{i,j},T\rangle\langle V_i,l\rangle\langle V_j,l\rangle\langle a\rangle \rightarrow \langle C_{i,j+1}\rangle\langle V_i,l\rangle\langle V_j,l\rangle\langle a+1\rangle \mid 1\le i\le n, 1\le j\le n, 0\le a\le n^2, 1\le l\le 2\}$$
$$\cup\{\langle C_{i,n+1}\rangle \rightarrow \langle C_{i+1,1}\rangle \mid 1\le i\le n\}$$
$$\cup\{\langle C_{n+1,1}\rangle\langle a\rangle \rightarrow \langle E_{n+1,1}\rangle\langle a,1\rangle \mid 0\le a\le n^2\}$$

In the above evolution rules for the inner membrane, each partitioning of graph is checked whether the divided two sets are equal size or not. At first of Step 3, an object $\langle 0\rangle$ and $\langle B_1\rangle$ are created. Then, the sizes of divided two sets are counted using object $\langle k\rangle$ in the membrane. If v_i is contained in V', k is increased by 1, otherwise k is decreased by 1. After all vertices are checked, object $\langle C_{1,1}\rangle$ is created if k is 0, otherwise, object $\langle R_{1,1}\rangle$ and object $\langle 0,1\rangle$ are created. (The object $\langle 0,1\rangle$ denotes that the sum of the cost is set to 0 because the divided two sets are not equal size.) Next, cost of edges whose endpoints are in the same set is computed using object $\langle C_{i,j}\rangle$ and object $\langle k\rangle$. If a pair f two vertecs v_i and v_j satisfies the following two conditions, k is increased by 1.

- There exists $\langle e_{i,j},T\rangle$ in the membrane.
- There exists $\langle V_i,1\rangle\langle V_j,1\rangle$ or $\langle V_i,2\rangle\langle V_j,2\rangle$ in the membrane.

The above operation is executed for all pairs of two vertices v_i and v_j.

Afer the above operation, object $\langle R_{1,1}\rangle$ and object $\langle k,1\rangle$ are created for each pair of two vertices v_i and v_j. The $\langle R_{1,1}\rangle$ denotes that check for the membrane is finished, and $\langle k,1\rangle$ denotes the size of the subset in the membrane. In addition, object $\langle R_{1,1}\rangle$ triggers computation of Step 4.

In Step 4, the computed cost is sent out from the inner membrane to the outer membrane, and the maximum cost is computed in the outer membrane. Step 4 is executed applying the following set of evolution rules.
(Evolution rules for the outer membrane)

$$R_{1,4} = \{\langle l,2^k\rangle\langle m,2^k\rangle \rightarrow \langle m,2^{k+1}\rangle \mid 0\le l\le m\le n, 0\le k\le n-1\} \cup\{\langle k,2^n\rangle \rightarrow \langle k\rangle\langle T\rangle \mid 0\le k\le n\}$$

(Evolution rules for the inner membrane)

$$R_{2,4} = \{\langle E_{i,j}\rangle\langle e_{i,j},B\rangle \rightarrow \langle E_{i,j+1}\rangle \mid 1\le i\le n, 1\le j\le n, B\in\{T,F\}\} \cup\{\langle E_{i,n+1}\rangle \rightarrow \langle E_{i+1,1}\rangle \mid 1\le i\le n\} \cup\{\langle E_{n+1,j}\rangle\langle V_i,A\rangle \rightarrow \langle E_{n+1,j+1}\rangle \mid 1\le j\le n, A\in\{1,2\}\}$$
$$\cup\{[\langle E_{n+1,n+1}\rangle\langle a,1\rangle]_2 \rightarrow \langle a,1\rangle \mid 1\le a\le n^2\}$$

The $\langle E_{1,1}\rangle$ denotes that check for the membrane is finished, and $\langle k,1\rangle$ denotes the cost of partitioning in the membrane. Then, object $\langle R_{1,1}\rangle$ starts deletion of all objects in the membrane except for object $\langle k,1\rangle$, and object $\langle E_{n+2,1}\rangle$ dissolves the inner membrane.

On the other hand, in the above evolution rules for the outer membrane, the maximum cost is decided using 2^n objects $\langle k,1\rangle$, which denotes the costs sent out from the inner membranes. In the evolution rules, pairs of objects, $\langle l,2^k\rangle$ and $\langle m,2^k\rangle$, are compared, and an object $\langle m,2^{k+1}\rangle$, which denotes $l\ge m$ and the number of objects is 2^{k+1}, is created. At the end of Step 4, objects $\langle k,1\rangle$ and $\langle T\rangle$ are created. The object $\langle k,1\rangle$ denotes that the maximum cost is k, and object $\langle T\rangle$ triggers computation of Step 5.

In Step 5, all possible partitionings of the graph are created again by dividing the inner membranes repeatedly. Step 5 is executed applying the following set of evolution rules.
(Evolution rules for the outer membrane)

$$R_{1,5} = \{\langle T\rangle\langle k\rangle[\ ]_2 \rightarrow [\langle k\rangle\langle T\rangle]_2 \mid 0\le k\le n\}$$

(Evolution rules for the inner membrane)

$$R_{2,5} = \{\langle T\rangle\langle D\rangle \rightarrow \langle S'_1\rangle\} \cup\{[S'_i]_2 \rightarrow [\langle S'_{i+1}\rangle\langle V_i,0\rangle]_2[\langle S'_{i+1}\rangle\langle V_i,1\rangle]_2 \mid 1\le i\le n\}$$

In the above evolution rules for the outer membrane, objects $\langle T\rangle$ and $\langle D\rangle$, which are created in Step 4, are moved into the inner membrane.

In the above evolution rules for the inner membrane, an object $\langle S'_{n+1}\rangle$ is created, and then, the object triggers division rules. Then, 2^n membranes are created as well as Step 2, and object $\langle S'_{n+1}\rangle$, which triggers computation of Step 6, is created in each divided membrane.

In Step 6, each divided membrane is checked if each partitioning of the graph is the uniform graph partition or not using the maximum cost, and the membrane is dissolved if the membrane contains a subset that is not the uniform graph partition. Step 6 is executed applying the following set of evolution rules.
(Evolution rules for the outer membrane)

$$R_{1,6} = \{\langle D,2^k\rangle\langle D,2^k\rangle \rightarrow \langle D,2^{k+1}\rangle \mid 0\le k\le n-1\} \cup\{\langle D,2^n\rangle \rightarrow \langle B\rangle\}$$

(Evolution rules for the inner membrane)

$$R_{2,6} = \{\langle S'_{n+1}\rangle \rightarrow \langle B'_1\rangle\langle 0,1\rangle\} \cup\{\langle B'_i\rangle\langle V_i,1\rangle\langle a,1\rangle \rightarrow \langle B'_{i+1}\rangle\langle V_i,1\rangle\langle a+1,1\rangle \mid -n\le a\le n, 1\le i\le n\} \cup\{\langle B'_i\rangle\langle V_i,2\rangle\langle a,1\rangle \rightarrow \langle B'_{i+1}\rangle\langle V_i,2\rangle\langle a-1,1\rangle \mid -n\le a\le n, 1\le i\le n\} \cup\{\langle B'_{n+1}\rangle\langle a,1\rangle \rightarrow \langle E'_{1,1}\rangle \mid -n\le a\le -1, 1\le a\ \le n\}$$

$\cup\{\langle B'_{n+1}\rangle\langle 0\rangle \to \langle C'_{1,1}\rangle\}$
$\cup\{\langle C'_{i,j}\rangle\langle e_{i,j}, F\rangle \to \langle C'_{i,j+1}\rangle \mid 1 \le i \le n, 1 \le j \le n\}$
$\cup\{\langle C'_{i,j}\rangle\langle e_{i,j}, T\rangle\langle V_i, l\rangle\langle V_j, m\rangle \to \langle C'_{i,j+1}\rangle\langle V_i, l\rangle\langle V_j, m\rangle$
$\mid 1 \le i \le n, 1 \le j \le n, 1 \le l \le 2, 1 \le m \le 2, l \ne m\}$
$\cup\{\langle C'_{i,j}\rangle\langle e_{i,j}, T\rangle\langle V_i, l\rangle\langle V_j, l\rangle\langle a\rangle$
$\to \langle C'_{i,j+1}\rangle\langle V_i, l\rangle\langle V_j, l\rangle\langle a-1\rangle$
$\mid 1 \le i \le n, 1 \le j \le n, 0 \le a \le n^2, 1 \le l \le 2\}$
$\cup\{\langle C'_{i,n+1}\rangle \to \langle C'_{i+1,1}\rangle \mid 1 \le i \le n\}$
$\cup\{\langle C'_{n+1,1}\rangle\langle a\rangle \to \langle E'_{n+1,1}\rangle \mid 1 \le a \le n\}$
$\cup\{[\langle C'_{n+1,1}\rangle\langle 0\rangle]_2 \to [\ ]_2\langle D, 1\rangle\}$
$\cup\{\langle E'_{i,j}\rangle\langle e_{i,j}, B\rangle \to \langle E'_{i,j+1}\rangle$
$\mid 1 \le i \le n, 1 \le j \le n, B \in \{T, F\}\}$
$\cup\{\langle E'_{i,n+1}\rangle \to \langle E_{i+1,1}\rangle \mid 1 \le i \le n\}$
$\cup\{\langle E'_{n+1,j}\rangle\langle V_i, A\rangle \to \langle E'_{n+1,j+1}\rangle$
$\mid 1 \le j \le n, A \in \{1, 2\}\}$
$\cup\{[\langle E'_{n+1,n+1}\rangle]_2 \to \langle D, 1\rangle \mid 1 \le a \le n^2\}$

In the above evolution rules for the inner membrane, each partitioning of the graph is checked whether the divided two sets are equal size or not as well as Step 3 using object $\langle B'_i\rangle$ and object $\langle k, 1\rangle$. If the divided two sets are equal size, object $\langle C'_{1,1}\rangle$ is created, otherwise object $\langle R'_{1,1}\rangle$ is created. Next, cost of edges whose endpoints are in the same set is computed using object $\langle C'_{i,j}\rangle$ and object $\langle k\rangle$. If a pair of two vertecs v_i and v_j satisfies the following two conditions, k is decreased by 1. The operation is executed for each pair of two vertices v_i and v_j.

- There exists $\langle e_{i,j}, T\rangle$ in the membrane.
- There exists $\langle V_i, 1\rangle\langle V_j, 1\rangle$ or $\langle V_i, 2\rangle\langle V_j, 2\rangle$ in the membrane.

After each pair of two vertices v_i and v_j are checked, the object $\langle D, 1\rangle$ is sent out to the outer membrane if k is 0, otherwise, object $\langle E'_{1,1}\rangle$ is created.

The object $\langle E'_{1,1}\rangle$ deletes all objects in the membrane as well as the object $\langle E_{1,1}\rangle$ in Step 3. When the deletion is finished, object $\langle D, 1\rangle$ is created for each deleted membrane, and the inner membrane is dissolved because the partitioning of the graph is not the uniform graph partition.

Therefore, there are 2^n objects, $\langle D, 1\rangle$, are created in the outer membrane after checking for all inner membranes. Since an object $\langle D, k^{i+1}\rangle$ is created with two $\langle D, k^i\rangle$s according to the above evolution rules, $\langle D, k^n\rangle$ is evolved into the object $\langle B\rangle$, which triggers computation of Step 7.

In Step 7, one of the inner membranes, which includes the uniform graph partition, is dissolved, and the result is sent out from the outer membrane. Step 7 is executed applying the following set of evolution rules.

(Evolution rules for the outer membrane)

$$\begin{aligned} R_{1,7} &= \{\langle B\rangle[\ ]_2 \to [\langle B\rangle]_2\} \\ &\quad \cup\{\langle O_i\rangle\langle V_i, W\rangle \to \langle O_{i+1}\rangle\langle A_i\rangle\langle V_i, W\rangle \\ &\quad\ \mid 1 \le i \le n, W \in \{1, 2\}\} \\ &\quad \cup\{[\langle V_i, W\rangle\langle A_i\rangle]_1 \to [\ ]_1\langle V_i, W\rangle \\ &\quad\ \mid 1 \le i \le n, W \in \{1, 2\}\} \end{aligned}$$

(Evolution rules for the inner membrane)

$$R_{2,7} = \{[\langle B\rangle\langle e_{1,1}, F\rangle]_2 \to \langle O_1\rangle\}$$

At the beginning of Step 7, object $\langle B\rangle$ is moved into one of the inner membranes, which is selected nondeterministically. Then, the membrane is dissolved using the object, and object $\langle O_1\rangle$ is created with other objects in the inner membrane. Next, a set of output objects $\{\langle V_i, W\rangle \mid W \in \{1, 2\}\}$ is sent out from the outer membrane to the outside region using auxiliary objects $\langle O_{i+1}\rangle$ and $\langle A_i\rangle$.

We now summarize the asynchronous P system Π_{UGP} for the uniform graph partitioning as follows.

$$\Pi_{UGP} = (O, \mu, \omega_1, \omega_2, , R_1, R_2, i_{in}, i_{out})$$

$$\begin{aligned} \bullet O &= O_E \cup O_S \\ &\quad \cup\{\langle M_{i,j}\rangle\langle O_i\rangle \mid 1 \le i \le n+1, 1 \le j \le n+1\} \\ &\quad \cup\{\langle S_i\rangle, \langle C_{i,j}\rangle, \langle E_{i,j}\rangle, \langle S'_i\rangle, \langle C'_{i,j}\rangle, \langle E'_{i,j}\rangle \\ &\quad\ \mid 1 \le i \le n+1, 1 \le j \le n+1\} \\ &\quad \cup\{\langle A_i\rangle \mid 0 \le i \le n\} \\ &\quad \cup\{\langle B_i\rangle, \langle B'_i\rangle \mid 0 \le i \le n+1\} \\ &\quad \cup\{\langle a\rangle, \langle a, 2^p\rangle\langle D, 2^p\rangle \mid -n \le a \le n^2, 0 \le p \le n\} \\ &\quad \cup\{\langle D\rangle, \langle T\rangle, \langle O\rangle\} \end{aligned}$$

- $O_E = \{\langle e_{i,j}, W\rangle \mid 1 \le i \le n, 1 \le j \le n, W \in \{T, F\}\}$
- $O_S = \{\langle V_i, A\rangle \mid 1 \le i \le n, A \in \{0, 1\}\}$
- $\mu = [\ [\ ]_2\ ]_1$
- $\omega_1 = \omega_2 = \phi$
- $R_1 = R_{1,1} \cup R_{1,4} \cup R_{1,5} \cup R_{1,6} \cup R_{1,7}$
- $R_2 = R_{2,1} \cup R_{2,2} \cup R_{2,3} \cup R_{2,4} \cup R_{2,5} \cup R_{2,6} \cup R_{2,7}$

3.4 An example of execution of the P system

Figures 2, 3 and 4 illustrate an example execution of the proposed P system Π_{UGP} for a graph in Figure 1.

In Step 1, a set of objects O_E is given from the outside region into the outer membrane. Then, two sets of evolution rules, $R_{1,1} \cup R_{2,1}$, are applied, and all objects are moved into the inner membrane. In Step 2, each inner membrane is repeatedly divided so as to create objects denoting all possible partitioning of graph applying $R_{2,2}$.

In Step 3, each inner membrane is checked if each divided two sets are equal size. In case that sizes of the two sets are

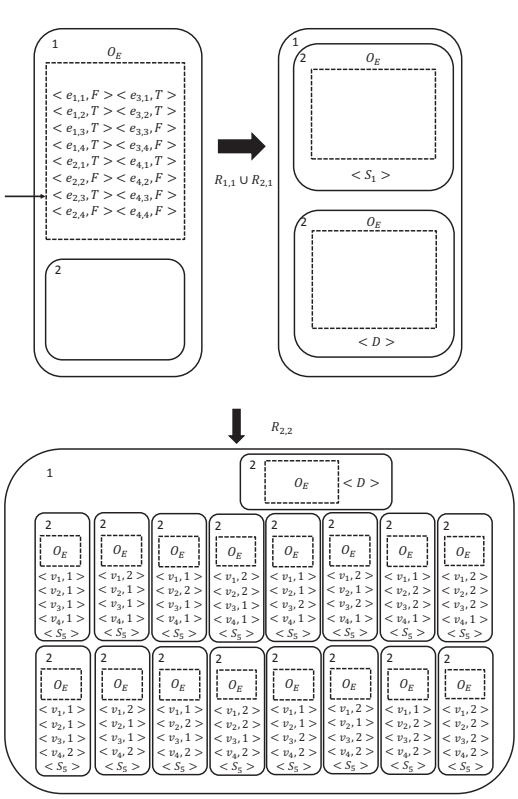


Fig. 2: An example of execution of Π_{UGP} (Steps 1,2)

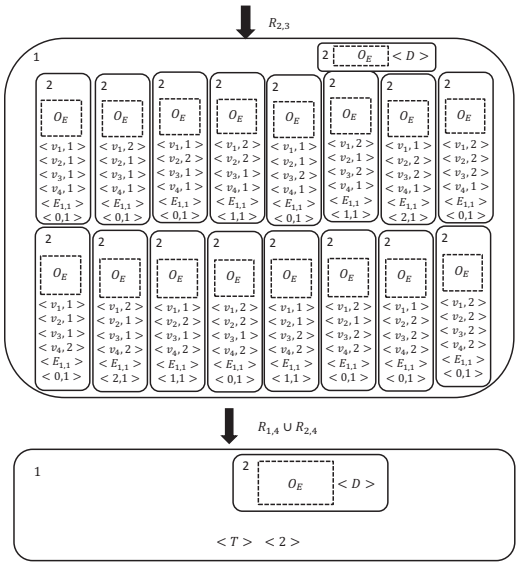


Fig. 3: An example of execution of Π_{UGP} (Steps 3,4)

not the same, $\langle 0, 1\rangle$ is created. Otherwise, object $\langle k, 1\rangle$ is is created. In this example, two kinds of objects which are $\langle 2, 1\rangle$ and $\langle 1, 1\rangle$ are created applying $R_{1,3}$. In the Step 4, all objects are deleted in the membrane, and an object $\langle k, 1\rangle$ is sent out from the membrane. The objects $\langle k, 1\rangle$ are merged into an object $\langle 2\rangle$ applying $R_{1,4}$.

In the Step 5, object $\langle 2\rangle$ is sent into the remained membrane applying $R_{1.5}$. And then, each inner membrane with $\langle 2\rangle$ is repeatedly divided again so as to create objects denoting all possible partitioning of the graph applying $R_{2,5}$. In Step 6, each inner membrane is checked if the partitioning of the graph is the uniform graph partition. If the partitioning of the graph is not uniform graph partition, all objects are deleted in the membrane applying $R_{2,6}$. In the final step, one of the inner membranes is dissolved applying $R_{1,7}$. A set of output objects $\{\langle V_1, 1\rangle, \langle V_2, 2\rangle, \langle V_3, 2\rangle, \langle V_4, 2\rangle\}$ is sent out to the outside region applying $R_{1,7}$.

3.5 Complexity

We now consider complexity of the asynchronous P system Π_{UGP}. Since $O(n^2)$ objects are moved sequentially in Step 1, both of the numbers of sequential and parallel steps in Step 1 are $O(n^2)$. In Step 2, $O(2^n)$ membranes are created, and the number of sequential and parallel steps of Step 2 are $O(2^n)$ and $O(n)$, respectively. In Step 3, both of the numbers of sequential and parallel steps in Step 3 are $O(n^2)$ since each edges is checked sequentially. In Step 4, the numbers of sequential and parallel steps are $O(n^2 \cdot 2^n)$ and $O(n^2)$ since each partitoning of the graph is checked sequentially. In Step 5 and Step 6, the procedure is almost same as the Step 2 and Step 4. Therefore, the numbers of sequential and parallel steps of Step 5 and Step 6 are the same as the numbers of sequential and parallel steps of Step 2 and Step 4, respectively. Since $O(n)$ objects are sent out as the output sequentially in Step 7, both of the numbers of sequential and parallel steps in Step 7 are $O(n)$.

Since the number of types of objects in the P system Π_{UGP} is $O(n^3)$, and $O(n^5)$ kinds of evolution rules are used, we obtain the following theorem for Π_{UGP}.

Theorem 1: The asynchronous P system Π_{UGP}, which computes the uniform graph partitioning for a graph with n vertices, works in $O(n^2 \cdot 2^n)$ sequential steps or $O(n^2)$ parallel steps using $O(n^3)$ types of objects and evolution rules of size $O(n^5)$. □

4. Other graph problems

We also propose asynchronous P systems for the other four graph problems, which are the (k, v)-balanced partitioning, maximum cut, dominating set, and connected dominating set, which are defined as follows.

- (k, v)-balanced partitioning is defined as partitioning of vertices into k sets of size at most $v(\frac{n}{k})$, while minimizing the sum of the cost of edges whose endpoints in different sets.
- Maximum cut is defined as partitioning of vertices into two sets, while maximizing the sum of the cost of edges whose endpoints in different sets.

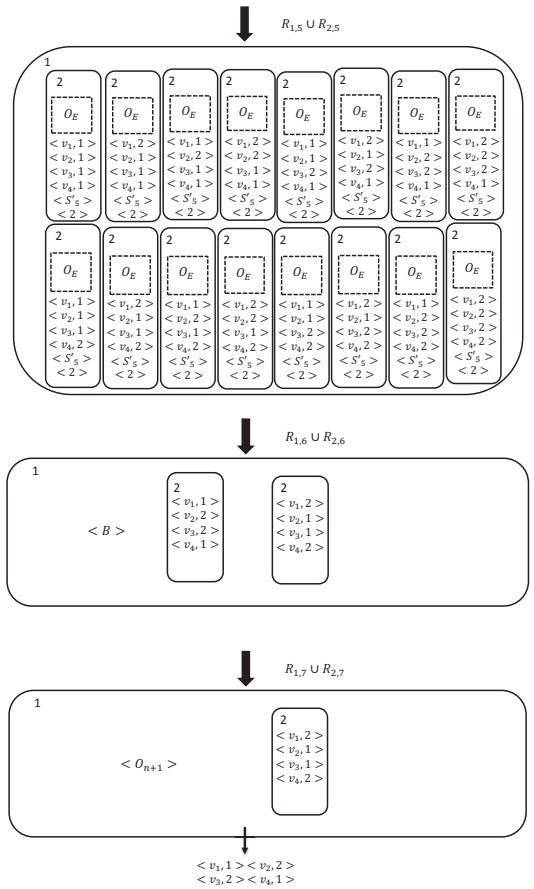


Fig. 4: An example of execution of Π_{UGP} (Steps 5,6,7)

- Dominating set for a graph $G = (V, E)$ is a subset $V' \subseteq V$ such that every vertex not in V' is adjacent to at least one member of V'. In addition, The dominating set is called connected if the induced subgraph for V' is a connected graph. The dominating set problem and the connected dominating set problem for a graph are problems that find the smallest dominating set and the smallest connected dominating set for the graph, respectively.

We propose four asynchronous P system Π_{KVP}, Π_{MAC}, Π_{DSP} and, Π_{CDP}, which solves the above four problems by modification of the asynchronous P system Π_{UGP} for the uniform graph partitioning. (We omit details of the P systems due to space limitation.)

We obtain the following theorems for the above four P systems.

Theorem 2: The asynchronous P system Π_{KVP}, which computes (k, v)-balanced partitioning problem for a graph with n vertices, works in $O(k^n \cdot n^2)$ sequential steps or $O(n^2)$ parallel steps using $O(n^3)$ types of objects and evolution rules of size $O(n^{k+1})$. □

Theorem 3: The asynchronous P system Π_{MAC}, which computes Maximum cut for a graph with n vertices, works in $O(n^2 \cdot 2^n)$ sequential steps or $O(n^2)$ parallel steps using $O(n^3)$ types of objects and evolution rules of size $O(n^5)$. □

Theorem 4: The asynchronous P system Π_{DSP}, which computes the dominating set for a graph with n vertices, works in $O(n^2 \cdot 2^n)$ sequential steps or $O(n^2)$ parallel steps using $O(n^3)$ types of objects and evolution rules of size $O(n^2)$. □

Theorem 5: The asynchronous P system Π_{CDP}, which computes the connected dominating set for a graph with n vertices, works in $O(n^2 \cdot 2^n)$ sequential steps or $O(n^2)$ parallel steps using $O(n^3)$ types of objects and evolution rules of size $O(n^2)$. □

5. Conclusions

In the present paper, we proposed P systems that solve five graph problems. The proposed P systems are fully asynchronous, i.e. any number of applicable rules may be applied in one step of the P system.

As future work, we are considering an asynchronous P system using the fewer number of membranes and evolution rules. In addition, we also consider reduction from a conventional P system to an asynchronous P system for the same problem.

References

[1] G. Păun, "Computing with membranes," *Journal of Computer and System Sciences*, vol. 61, no. 1, 2000.
[2] ——, "Introduction to membrane computing," *Springer*, 2006.
[3] A. Leporati and C. Zandron, "P systems with input in binary form," *International Journal of Foundations of Computer Science*, vol. 17, pp. 127–146, 2006.
[4] L. Pan and A. Alhazov, "Solving HPP and SAT by P systems with active membranes and separationrules," *Acta Informatica*, vol. 43, no. 2, pp. 131–145, 2006.
[5] G. Păun, "P system with active membranes: Attacking NP-complete problems," *Journal of Automata, Languages and Combinatorics*, vol. 6, no. 1, pp. 75–95, 2001.
[6] T. Tateishi and A. Fujiwara, "Logic and arithmetic operations with a constant number of steps in membrane computing," *International Journal of Foundations of Computer Science*, vol. 22, no. 3, pp. 547–564, 2011.
[7] C. Zandron, G. Rozenberg, and G. Mauri, "Solving NP-complete problems using P systems with active membranes," *Proceedings of the Second International Conference on Uncoventional Models of Computation*, pp. 289–301, 2000.
[8] R. Freund, "Asynchronous P systems and P systems working in the sequential mode," *WMC'04 Proceedings of the 5th international conference on Membrane Computing*, vol. 3365, pp. 36–62, 2005.
[9] T. Murakawa and A. Fujiwara, "Asynchronous P system for arithmetic operations and factorization," *Proceedings of 3rd International Workshop on Parallel and Distributed Algorithms and Applications*, 2011.
[10] H. Tagawa and A. Fujiwara, "Solving SAT and Hamiltonian cycle problem using asynchronous P systems," *IEICE Transaction on Fundamentals of Electronics, Communications and Computer Science*, vol. E95-D, no. 3, 2012.
[11] K. Tanaka and A. Fujiwara, "Asynchronous P systems for hard graph problems," *International Journal of Networking and Computing*, vol. 4, no. 1, pp. 2–22, 2014.

Math Words and their Dendrograms

Casanova-del-Angel, Francisco
Instituto Politécnico Nacional. México. fcasanova@ipn.mx www.franciscocasanova.pro

Abstract
This document present a hierarchical clustering algorithm based on graph theory, which, from generation of a path from a given vertex, builds a math word and calculates a cluster under an index. This is made possible due to modification of Tarry's algorithm, by exchanging path elements. When the one dimensional clustering index is added to σ, it gives us, what I have called, Tarry's hierarchy. From the definition of net word, cycle, tree, tree word and vertex, a theorem on the relationship between vertices, lines, and letters of a labyrinth is shown, which allows the generation of words and their Dendrograms with the application of Euclidian distance. The practical use of these concepts is then shown, namely, that they can provides possibilities of connections in arrangements for telephone centrals.

Keyword: Dendrograms, Tarry, Math words, connected, labyrinth

Introduction

The first person who studied the combinatory properties of schemes was Leonard Euler in 1736; he studied the network of the seven bridges in Konigsberg, figure 1. He wrote, in Berlin in 1739, "*in Königsberg Pomeranie, they have a little island named Kneiphof, the river is divided into two and around the island there are seven bridges. You can arrange a network where by you only walk over one at one time*". He continued, "Is possible for everyone, but that not everyone has the capacity to do it" Euler (1739).

The modern theorem enunciated by Euler, demonstrated the necessity of the parity of the valence in each vertex: *A connected graphic is eulerienne if all vertex have degree pair*. Since then, graph theory has developed slow but steadily. Its principal contributors are G. Tarry, who wrote about labyrinths in 1886 and 1895 (see Tarry 1886 and 1895), D. Konig (1936), C. Berge (1957), with a book about graphs and hypergraphs, W. T. Tutte, with his studies about Hamiltonian networks (1976). S. Bollobas, who wrote about Hamiltonians cycles in regular graphs, and P. Rosenstielh, with *Existence d'automates finis capables de s'accorder bien qu'arbitrairement connectés et nombreux* (1966), and *Labyrintologie mathématique* (1971), (see Rosenstiehl, 1971), and *Les graphes d'entrelacement d'un graphe* (1976). Recently, works on graph theory, like *A performance comparison between graph and hypergraph topologies for passive star WDM light wave networks*, (1998) by H. Bourdin, A. Ferreira and K. Marcus; as well as the work by Gondran and M. Minoux, entitled *Graphs et Algorithms* (1979), have gained attention. More recently, the randomized Tarry algorithm has been discussed in the article named searching a graph by Shmuel Gal (2004), and by Urretabizkaya and Rodríguez (2004), who implement the Tarry algorithm for solving mazes of known structure (2004). Today, graph theory is used in branches of mathematics such like theory of groups, topology, and theory of numbers, data analysis and clusters.

Among those who have contributed the most to the development of theory and models in telephone connections, are A. K. Erlang, who implemented the well-known *Erlang* Probability

Density Function, as well as works on *Solutions of Some Problems in the Theory of Probabilities of Significance in Automatic Telephone Exchanges*, (1917 and 1918); A. and Elldin, with his work *Switch Calculations General Survey,* published by LM Ericsson in (1955). More recent works on the subject include, *Network Flows: Theory, Algorithms and Applications*, by R. V. Ahuja, T. L. Magnanti, and J. B. Orlin, (1993), and *An Algorithm for Hierarchical network design*, by G. R. Mateurs, F. R. B. Cruz, and H.P. L. Luna (1994).

Definitions and algorithms

Definition. A graph is a pair $G = (X, U)$ where X is a finite set, of vertices of G, each element of which is incident to two elements of another finite set V named the set of vertices, see Berge (1970).

Definition. A labyrinth, ♭, is a finite set A not empty and of even cardinality, named set of words of ♭ or alphabet, supplied of one involution l without an indeterminate point (i.e: without a tendency change) called by the prime operator:

$$l \in A \text{ then } l' \in A,\ l' \neq l \text{ and } (l')' = l$$

and one relation of equivalence (called "with the same right as" where the classes are "the points" of labyrinth, the letters of same class has the same right as the point) indicated by the application of the letters which belong the letters in the set X of class:

$$d: A \longrightarrow X \quad : \quad d(l) = d(k)$$

by < the right of l is equal to right of k >, where < l has the same right as k >, see Rosenstiehl (1971).

It is possible to speak of the left of letter too, making: $g: A \longrightarrow X$ with $g(l) = d(l')\ \forall\ l \in A$, as far as the labyrinth its expressed by triad (A, i, d).

Definition. A labyrinth (A, i, d) is said to be orientated, when one part A^+ of A such that:

$$l \in A^+ \Leftrightarrow l' \notin A^+\ \forall\ l \in A$$

Definition. Based on this definition of a labyrinth (A^+, i, d), σ is called a word of this labyrinth if it belongs to some of following classes:

- $\Delta_\alpha\ \forall\ \alpha \in X$ is called empty words of ♭
- l with $l \in A$ is called word-letter of the ♭, and
- $\sigma = l_1, .., l_r, l_{r+1}, \ldots, l_p$ with $l_r \in A \cup (\Delta_\alpha)\ \forall\ \alpha \in X$ we have $d(l_r) = g(l_{r+1})\ \forall\ r = 1, .., p\text{-}1$

Therefore, given X as the set of vertex, you can have left and right applications in all word of the labyrinth ♭, defined as: $d, g: A \cup (\Delta_x)_{x \in X} \rightarrow X$ or $d, g: ♭ \rightarrow X$ with $g(\Delta_x) = g(l_1)$ and $d(\Delta_x) = d(l_p)$, i.e., $g(\Delta_x) = d(\Delta_x) = x$. And remember, σ is cyclical if $g(\sigma) = d(\sigma)$. For more elaborate definitions of tree, pure word and neutral word, see Golumbic (1980) and Bollobás (1978).

Definition. One tree A is a graph $G = (X, U)$ connected and non-cyclical.

Definition. σ of $l_\sigma = (A, i, d)$ it is a pure word if inside σ there exists a occurrence of each letter of *A*.

Definition. If $\sigma \in l_\sigma$ and $\sigma \approx \Delta_\alpha \ \forall \ \alpha \in X$, σ is a neutral word in α, if the neutral element Δ_α is a neutral word in some α-particular.

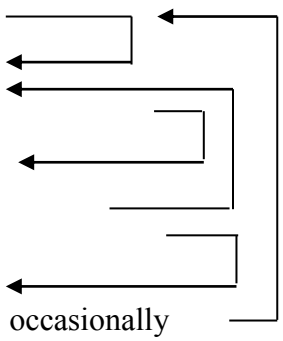
Algorithm of Tarry

Definition. If l_σ labyrinth is connected, we say Tarry's word of l_σ all pure word $\Theta \in l_\sigma$ such that the entry tree *V* of Θ and the out tree *W* of Θ are opposites, i. e., $W = V'$.

Let l_σ be the lexical of connected labyrinth $l_\sigma = (A, i, g)$, with $l_1 \in A$. Let $V(\sigma)$ be the entry tree of σ, with σ a left factor of Tarry's word Θ to apply the following algorithm, see Tarry (1895):

T.0 To put $\sigma \leftarrow l_1$

T.1 If $\sigma l \in l_\sigma$ with $l \notin \sigma$ and $l' \notin V(\sigma)$ to put $\sigma \leftarrow \sigma l$
but
T.2 If $\sigma l \in l_\sigma$ with $l \notin \sigma$ to put $\sigma \leftarrow \sigma l$
if
but
T.3 Stop. Write $\Theta = \sigma$
Where $\Theta = \sigma$ is called Tarry's word



Note: The inversion of all entry letter is an out letter in $\Theta = \sigma$, where Θ is a cyclical word, and all letter of A haves the same right as the same left of Θ, see Casanova (1982).

Attachment index

The index of aggregation or level of classification of ways connected to the orientated labyrinth l_σ, begin with the first cycle or pleat of the Tarry's word. In this case, the index level is unit. When Tarry's word σ is formed, you always have to replace σ in the first place of the first obtained cycle.

Definition. If we let l_1 and $l_2 \in l_\sigma$ stand for two paths and let α stand for all letters with the same left g(Θ), and β stand for all letter with the same right d(Θ); the minimal distance d_M, the inferior ultrametric minimal distance over a point $x_0 \in l_1$ and $x_1 \in l_2$ is:

$$d_M(l_1, l_2) = \min \{d(x_0, x_1) \mid x_0 \in l_1 \& x_1 \in l_2\}$$

Remember, the algorithm is applied after finding the Tarry's word and his cluster by couples or pleats. To finish the last pleat, begin the letter's arrangement in σ. You must begin with the first pleat.

Conclusions

Since every problem demands a full solution using links that indicate the way to construct the algorithm solution within in a formal language capable of analyzing a system by which you can recreate the transit in its construction in any direction, we have presented here a new and modern form of Tarry's algorithm. This allows us to create a hierarchy based on a vertex, the levels of aggregation for the construction of the word of the circuit or labyrinth which applied to the

something of the telephones, and lets us know configurations of connections with optimal telephone line use, fluidity and economy.

To date there has been no published algorithm of aggregation (hierarchical or otherwise), based on theory of graphs, that starting from the construction of a trajectory begun on a given vertex, which would construct a math word and calculate an aggregation under an index. The latter has been possible thanks to the modification made in Tarry's algorithm, through the exchange of elements, which has allowed a better arrangement in the union of pairs of letters and the construction of the math words. The one dimensional aggregated index applied to σ results in what I have called Tarry's hierarchy. The next step is to apply weight to aggregation algorithms.

There is no published clustering algorithm (whether hierarchical or not), based on graph theory which, from generation of a path starting on a given vertex, builds a math word and calculates clustering under an index. This has been possible by modification to Tarry's algorithm, through exchange of elements, which has allowed a better arrangement of letters coupling and the construction of a math word. The one dimensional clustering index applied to σ gives what I call Tarry's hierarchy.

References

Berge, C. 1970. *Graphes et hypergraphes*. Dunod.

Bollobás, B. (ed). 1978. *Advances in graph theory*. North-Holland. North-Holland ISBN: 0 7204 0843 1.

Bourdin, H., Ferreira, A., and Marcus, K.1998. "A Performance Comparison between Graph and Hypergraphs Topologies for Passive Star WDM Light wave Networks". *Computer Networks and ISDN Systems*, 8(30), pp. 805-819. Doi: 10.1016/S0169-7552(97)00125-6.

Casanova del Angel, F. 1982. "Introducción a la teoría de gráficas, a los laberintos y a sus palabras". *Boletín de Graduados e Investigación*, vol. I, núm. 3, pp. 23-51. IPN. México.

Euler, L. 1739. *Solutio Problematis ad geometriam situs pertinentis*. Mémoire de l'Academie des Sciences de Berlin.

Golumbic, M. Ch. 1980. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press. ISBN: 0-444-51530-5.

Gondran, M and Minoux, M. 1979. *Graphes et algorithmes*, éditions Eyrolles, coll. « Dir. Ét. &Rech.EDF », 1979 (réimpr. 1985), 546 p. EAN13: 9782212015713.

Kőnig, D. 1936. *Theorie der Endlichen und Unendlichen Graphen: kombinatorische Topologie der Streckenkomplexe*. Leipzig: Akad. Verlag.

Tarry, G. 1886. *Parcours d'un labyrinthe rentrant*. Assoc. Franç. Pour l'Avanc. Des Sciences. pp. 49-53.

Tarry, G. 1895. "Le problème des labyrinths". *Nouvelles annales de Mathematiques*, vol. XIV.

Rosenstiehl, P. 1971. "Labyrinthologie Mathematique". *Mathematique et Sciences Humaines*, 9e année, Num. 33, 5-32.

Rosenstiehl, P. 1976. *Les graphes d´entrelacement d´un graphe*. Coll. Inter. CNRS nº. 260, Problèmes Combinatories et Théorie des Graphes, Orsay (Editions du CNRS, 1976) 359-362.

Shmuel Gal. 2004. Searching a graph. A working version.

Urretabizkaya, R and Rodríguez, O. 2004. Análisis e implementación de algoritmos para la solución de laberintos de estructura conocida. Universidad Autónoma de Querétaro. México.

W T Tutte. 1976. On Hamiltonian circuits. Colloquio Internazionale sulle. Teorie Combinatoire. Atti Convegni Lincei 17. Accad Naz. Lincei. Roma I, pp. 193-199.

Improved computing performance for algorithm finding the shortest path in extended graph

Master. Lau Nguyen Dinh[1], Assoc. Prof. Chien Tran Quoc[2]. Dr.of Sc, Assoc. Prof. Thanh Le Manh[3]. Ph.D
[1] College of Transport No II, Danang, Vietnam, E-mail: launhi@gmail.com
[2] University of Danang, Danang, Vietnam, Email: tqchien@dce.udn.vn
[3] University of Hue, Hue, Vietnam, Email: lmthanh@hueuni.edu.vn

Abstract. The graph is a powerful mathematical tool applied in many fields such as transportation, communication, information technology, economy, … Up to now, in ordinary graphs the weights of edges and vertexes have only been considered independently where the length of a path is simply the sum of weights of the edges and the vertexes on this path. However, in many practical problems, weights at a vertex are not the same for all paths passing this vertex, but they depend on the coming and leaving edges. Algorithm finding the shortest path from a vertex to many vertices in the extended graph has also been studied in the paper [1, 6]. In this paper, We build *parallel algorithm to find the shortest path from a vertex to many vertices in the extended graph* to reduce computation time [2, 3, 4, 5, 7, 8, 9].

Keyword: Parallel, graph, extended graph, algorithm, the shortest path.

I. INTRODUCTION

Up to now, in ordinary graphs the weights of edges and vertexes have only been considered independently where the length of a path is simply the sum of weights of the edges and the vertexes on this path. However, in many practical problems, weights at a vertex are not the same for all paths passing this vertex, but they depend on the coming and leaving edges [1, 6].

For example, the travel time through the intersection traffic on the network depends on the direction of the vehicles: turn right, go straight or turn left. Therefore, it is necessary to build extended graph models to apply in making models for real world situations more accurately and efficiently. Algorithm finding the shortest path is the basis one used in many optimization problems in graphs and networks.

Therefore, in this paper we define extended graph model and build sequential algorithm to find the shortest path in the extended graph. Furthermore, a problem araising in applying algorithm to find maximum concurrent multi-commodity linear flow with minimal cost in extended traffic network is that in extended traffic network there are a great number of roads and a growing number of the new routes built that leads to a huge number of variables (up to thousands of variables). So the problem of finding the shortest path in the extended graph to process faster as well as take advantage of multi-core architecture, data processing with large scale with good results require the construction of parallel algorithm [2, 3, 4, 5, 7, 8, 9]. The results in this paper are basically systematized and proven..

II. THE EXTENDED GRAPH

Given extended graph G (V, E) with a set of vertices V and a set of edges E, where edges can be directed or undirected. Each edge e ∈ E is weighted $w_E(e)$. With each vertex v ∈ V, E_v is the set of edges of vertex v. Each vertex v ∈ V and each edge (e, e ') ∈ E_v x E_v, e ≠ e' is weighted w_V (v, e, e ').

The (V, E, w_E, w_V) is called extended graph.

Let p be the path from u to v through the edge e_i, i = 1, ..., h +1, and the vertices u_i, i = 1, ..., h, as following:

P=[u, e_1, u_1, e_2, u_2, …, e_h, u_h, e_{h+1}, v]

Define the length of the path p, denoted l(p), as following:

$$l(p) = \sum_{i=1}^{h+1} w_E(e_i) + \sum_{i=1}^{h} w_V(u_i, e_i, e_{i+1}) \quad (1)$$

III. ALGORITHM FINDING THE SHORTEST PATH FROM A VERTEX TO MANY VERTICES IN EXTENDED GRAPH

- *Input*. Extended graph G(V, E, w_e, w_v), vertex s∈V and set $U \subset V$.
- *Output.* l(v) is the length of the shortest path from s to v and the shortest path (if l(v) <+∞), $\forall u \in U$
- *Methods.*

The algorithm uses the following symbols:

S is a set of vertices that found the shortest path starting from s;

T=V-S;

l(v) is the length of the shortest path from s to v;

VE={(v,e)|v∈V\{s}& e∈E_V} ∪ {(s, ϕ)} is the set of vertices and edges;

SE is a set of vertex-edge excluded from VE;

TE=VE-SE;

L(v,e) is the vertex-edge pair label (v,e) ∈ VE

P(v,e) is the vertex- front edge (v, e)∈ VE

SU, the set in U found the shortest path starting from s;

TU=U-SU;

Step 1. (Initialized) Let S = ϕ ; T=V; TU=U; SU= ϕ ;

Let VE={(v,e)|v∈V\{s}& e∈E_V} ∪ {(s, ϕ)}

SE= ϕ ; TE=VE;

Set L(v,e)=∞, ∀ (v,e) ∈ VE, L(s, ϕ):=0.

Set P(v,e)= ϕ ∀ (v,e) ∈ VE

Step 2. Calculate m = min{L((v,e))| (v,e) ∈ TE}.

If m=+∞, go to Step 5.

Else, if m<+∞, choose (v_{min}, e_{min}) ∈ TE so that L(v_{min}, e_{min})=m,

Set TE=TE-{(v_{min}, e_{min})}, SE=SE ∪ {(v_{min}, e_{min})}, go to step 3.

Step 3. If $v_{min} \notin S$, then set le(v_{min})=e_{min}, S=S ∪ $\{v_{min}\}$, l(v_{min})=L(v_{min} , e_{min}), T=T-{ v_{min}}, if v_{min} ∈ TU then set SU= SU ∪ v_{min}, and TU=TU-{v_{min}}.

-if TU= ϕ , go to step 5, else go to step 4.

Step 4. For any (v,e) ∈ TE adjacent (post-adjacent) (v_{min}, e_{min}),

Set$L^{'}$(v,e)=L(v_{min},e_{min})+w_E(v_{min},v)+ w_V(v_{min},e_{min},e), if v_{min} ≠ s and $L^{'}$(v,e)=L(s, ϕ)+ w_E(v_{min},v) if v_{min} = s.

IF L(v,e)>$L^{'}$(v,e), then set L(v,e)=$L^{'}$(v,e) and P(v,e)= (v_{min}, e_{min}) Back to step 2.

Step 5. (Finding the shortest path)

For any vertex t ∈ SU, set l(t)=L(t,le(t)) is the length of the shortest path from s to t. From t tracing back the front vertex-edge, we receive the shortest path as following: let (v_1, e_1)=P(t,le(t)), (v_2, e_2)=P(v_1, e_1), …, (v_k, e_k) =P(v_{k-1}, e_{k-1}), (s, ϕ)= P(v_k, e_k).

It is Inferred that the shortest path is:

s→v_k→v_{k-1}→…→v_1→t *End.*

Theorem 1. The algorithm finding the shortest path from a vertex to many vertices in the extended graph is true.

Proof: Symbolise in turn the vertex-edge pairs come into PE is

(v_0,e_0)= (s, ϕ), (v_1,e_1), …, (v_m,e_m)= (t,le(t)).

We prove by induction that L(v_i, e_i) is the length of the shortest path from s to v_i through edge e_i, i=1, …, m.

Basic step: Obviously L(v_1, e_1) is the length of the shortest path from s to v_1 through edge e_1. This is the length of the shortest path from s to v_1, ie l(v_1)= L(v_1, e_1), since v_1 has just been put into P.

Induction step : Suppose L(v_i, e_i) is the length of the shortest path from s to v_i through edge e_i for any i<k. We prove L(v_k, e_k) is the length of the shortest path from s to v_k through egde e_k.

Let p be the shortest path from s to v_k through edge e_k, symbol l(p) is the length of p. Vertex-edge pairs on p, excluded (v_k,e_k), must belong to SE'

SE' = {(v_1,e_1), (v_2,e_2), …, (v_{k-1},e_{k-1})}

Indeed, assuming else let (v,e) be the first vertex-edge pair on p from s, that does not belong to SE'. Symbol (v_i, e_i) ∈ SE' is the vertex-edge on the path p standing in front of (v,e) we have

L(v,e) ≤ L(v_i, e_i) +w_E(v_i,v)+w_V(v,e_i,e) < l(p) ≤ L(v_k, e_k) then (v,e) must be put into SE and stand in front of v_k, i.e. (v, e) belongs to SE, and this conflicts.

Now, let (v_h, e_h) be the front vertex- edge (v_k,e_k) on p. According to the label calculation we have

L(v_k,e_k) ≤ L(v_h, e_h) +w_E(v_h,v_k)+w_V(v_h,e_h,e_k) = l(p).

inferred L(v_k,e_k) = l(p) is the length of the shortest path p from s to e_k through edge e_k.

Finally, since (t,l(e(t)) is the vertex - edge containing the vertex t first coming to SE, so l(t)=L(t,le(t)) is the length of the shortest path from s to t □

Theorem 2 . Let G be the extended graph having n vertices. The algorithm has complexity O(n^3).

Proof: Since the edges of each vertex is at most (n-1), so the set VE has more elements than n (n-1). Since each loop from step 2 to step 4 chooses a pair of vertex – edge in set VE to put into SE, so the number of the loops does not exceed n(n-1), At each loop, algorithm surveys maximumly (n-1) vertex-edge pairs which are adjacent vertices - edges considered in step 4. It is Inferred that the complexity of the algorithm is $O(n^3)$

IV. EXAMPLE

Let extended graph at fugire 1. The graph has 6 vertices, 6 directed edges and 3 undirected ones. The weights of edges w_E are represented in the graph at figure 1 and the weights of the vertices are showed in table 1

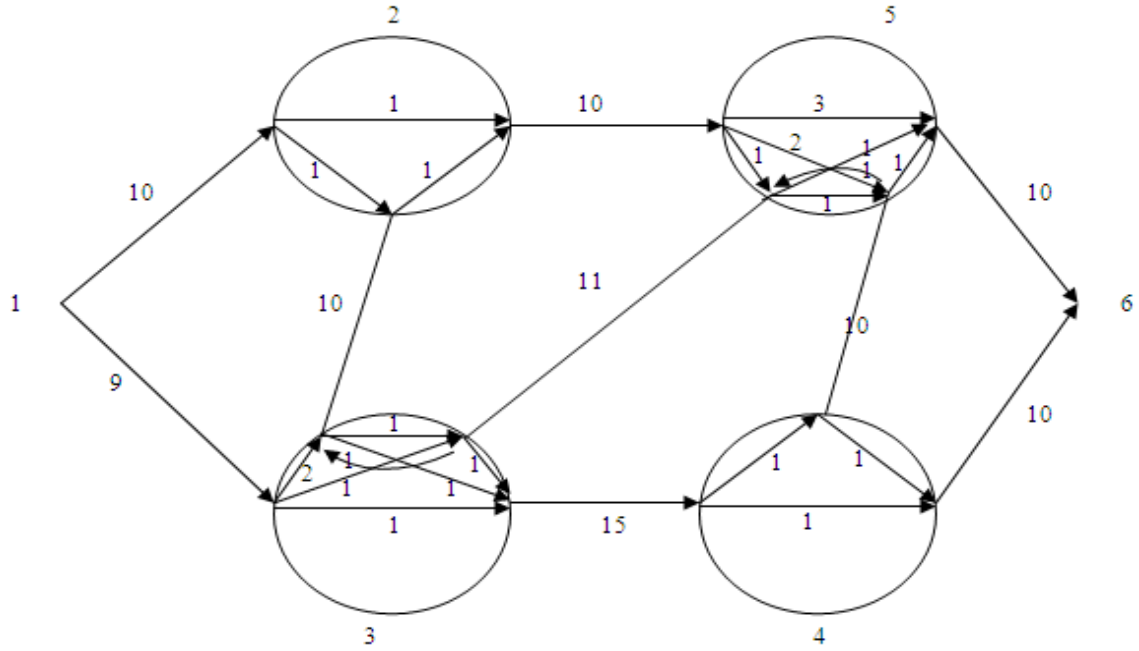
Applying the algorithm to find the shortest path from one vertex to all vertices. The process of implementation of the algorithm

It is inferred that the shortest paths from vertex 1 to all vertices as following:

From 1 to 2: 1→2, length 10. From 1 to 3: 1→3, length 9. From 1 to 4: 1→3→4, length 25. From 1 to 5: 1→2→5 or 1→3→5, length 21. From 1 to 6: 1→3→5→6, length 32

V. PARALLEL ALGORITHM FINDING THE SHORTEST PATH FROM A VERTEX TO MANY VERTICES IN EXTENDED GRAPH

Extended graph is shown in fugire 1



Fugire 1. Extended graph has the weights of vertices

- The idea of the algorithm

We build parallel algorithms on k processors (P_0, P_1,…,P_{k-1}). In k processors, there is a main processor (P_0) to manage data, divide data into k-1 sub-processors (P_1,…,P_{k-1}) Sub-processors receive the data from the main processor and find minximun L(v,e) on their vertices and send it to main processor. The main processor will find $L(v_{min}, e_{min}) = \min(L_i(v,e))$, i=0,…,k-1) which sub-processors sent. Next, The main processor will send (v_{min}, e_{min}) to sub-processors to calculate.

- Input: Extended graph G(V, E, w_E, w_V), vertex $s \in V$ and set $U \subset V$ k processors (P_0, P_1,…,P_{k-1})
- *Output:* l(v) is the length of the shortest path from s to v and the shortest path (if n l(v) $<+\infty$), $\forall u \in U$
- *Methods:*

Step 1. The main processor P_0 performs

- Divide the weights of the edges $A(w_E)$,the number of vertices T and the set of vertex- edge pairs of VE of graph for k processors
- Build T_i (i=0,…,k-1) be the set of vertices that processors P_i (i=0,…,k-1) receive

- Build VE_i (i=0,…,k-1) be the set of vertex-edge pairs that processors P_i (i=0,…,k-1) receive
- Build A_i (i=0,…,k-1) be matrix of edge weights that processors P_i (i=0,…,k-1) receive
- Initialize set T=V, S= ϕ SU= ϕ, TU=U
- Send T_i, A_i, VE_i (i=1,..,k-1) and weights w_V to k-1 processors

Step 2. k processors perform.
- Receive data from step 1
- The main processor P_0 assigns L(s, ϕ):=0;
- In k initialized processors. TE_i=VE_i (i=0,…,k-1), SE_i= ϕ ,(i=0,…,k-1)
- In k assigned processors, let L(v,e)=∞ and P(v,e)= ϕ $\forall$ (v,e) ∈ VE_i (i=0,…,k-1)

Step 3. k processors perform
- In k processors find m_i = min{L((v,e)$_j$)| (v,e)$_j$ ∈ TE_j (j=0,…,k-1) }
- The sub-processors send m_i (i=1,…,k-1) and (v,e) they have just found to the main processor

Step 4. The main processor P_0 performs
- The main processor finds m=min{m_i , i=0,…,k-1} =L(v,e)
- If m=+∞, go to step 7.
- Else, if m<+∞ send m to sub-processors.

Step 5. k sub-processors perform
- k sub-processors chooses (v_{min}, e_{min}) ∈ TE_i (i=0,…,k-1) | L(v_{min}, e_{min})=m,
set TE_i=TE_i-{(v_{min}, e_{min})}
- The main processor P_0 performs: If $v_{min} \notin S$, then set le(v_{min})=e_{min}, S=S $\cup \{v_{min}\}$, l(v_{min})=L(v_{min} , e_{min}), T=T-{ v_{min}}, if v_{min} ∈ TU then set SU= SU ∪ v_{min}, and TU=TU-{v_{min}}
- The main processor P_0 performs: If TU= ϕ, go to step 7, else go to step 6

Step 6. k processors perform
For any (v,e) ∈ TE_i adjacent (post-adjacent) (v_{min}, e_{min}), set L'(v,e)=L(v_{min}, e_{min})+ w_E(v_{min},v) +w_V(v_{min},e_{min},e) if $v_{min} \neq$ s. L'(v,e)=L(s, ϕ)+ w_E(v_{min},v) if v_{min} = s.
If L(v,e)>L'(v,e), then assign L(v,e)=L'(v,e) and P(v,e)= (v_{min}, e_{min})
Back to step 3.

Step 7.
k-1 sub-processors send results to main processor and finish, the main processor receive results from sub-processors and find the shortest path as following:

For any vertex t∈ SU, set l(t)=L(t,le(t)) is the length of the shortest path from s to t. From t tracing back the front vertex-edge, we receive the shortest path as following: Put (v_1,e_1)=P(t,le(t)), (v_2,e_2)=P(v_1,e_1), …, (v_k,e_k) =P(v_{k-1},e_{k-1}), (s, ϕ)= P(v_k,e_k).
It is inferred that the shortest path is: s→v_k→v_{k-1}→…→v_1→t *End.*

Theorem 3. The parallel algorithm has complexity

$$O(\frac{n^3}{k}) + O(n \log k)$$

Proof: In theorem 2, the complexity of the sequential algorithm is $O(n^3)$.

Number of processor is n/k (n vertices, k processors)

Since the edges of each vertex is at most (n-1), so the set VE_i has more elements than $\frac{n}{k}$(n-1). Since each loop chooses a pair of vertex – edge in set VE_i to put into SE_i, so the number of the loops does not exceed $\frac{n}{k}$(n-1).

So computation time for min value and update for min value is $O(\frac{n^2}{k})$.

Test time for v_{min} in step 6 and put set S has more elements than n. So computation time is $O(\frac{n^3}{k})$. Communication time in processors k is O(logk). When the algorithm stop, we have n communication steps.

So the complexity of the parallel algorithm in the general case is:

$$O(\frac{n^3}{k}) + O(n \log k) \qquad (2)$$

VI. AN EXAMPLE FOR APPLYING PARALLEL ALGORITHM

The extended graph is showed in figure 1. Applying the parallel algorithm for 2 processors (k=2) P_0 and P_1. In which P_0 is main processor, P_1 is sub-processor. P_0 performs calculation similar to P_1. The process of implementing the parallel algorithm as following:

Step 1. P_0 divides and sends data to P_0, P_1
Divide T_0={1,2,3} for P_0, T_1={4,5,6} for P_1

$$A_0 = \begin{bmatrix} \infty & 10 & 9 \\ \infty & \infty & 10 \\ \infty & 10 & \infty \\ \infty & \infty & \infty \\ \infty & \infty & 11 \\ \infty & \infty & \infty \end{bmatrix} \qquad A_1 = \begin{bmatrix} \infty & \infty & \infty \\ \infty & 10 & \infty \\ 15 & 11 & \infty \\ \infty & 10 & 10 \\ 10 & \infty & 11 \\ \infty & \infty & \infty \end{bmatrix}$$

Fugire 2. Matrix of edge weighst on P_0, and P_1

VE_0={ L(1, ϕ);L(2,(1,2));L(2,(3,2)); L(3,(1,3); L(3,(2,3)); L(3,(5,3))}

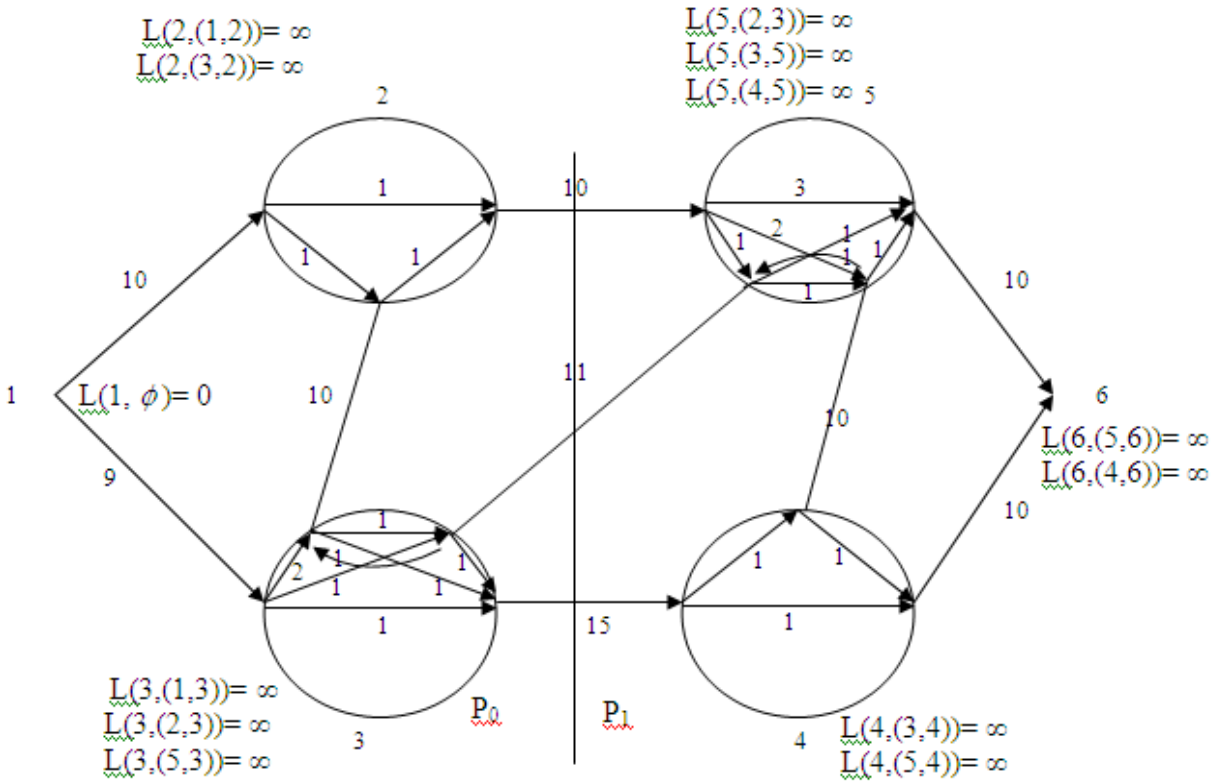
VE_1={ L(4,(3,4)); L(4,(5,4)); L(5,(2,3)); L(5,(3,5)); L(5,(4,5)); L(6,(5,6)); L(6,(4,6))}

Send w_V to P_0 and P_1

Step 2. The processors P_0 and P_1 perform

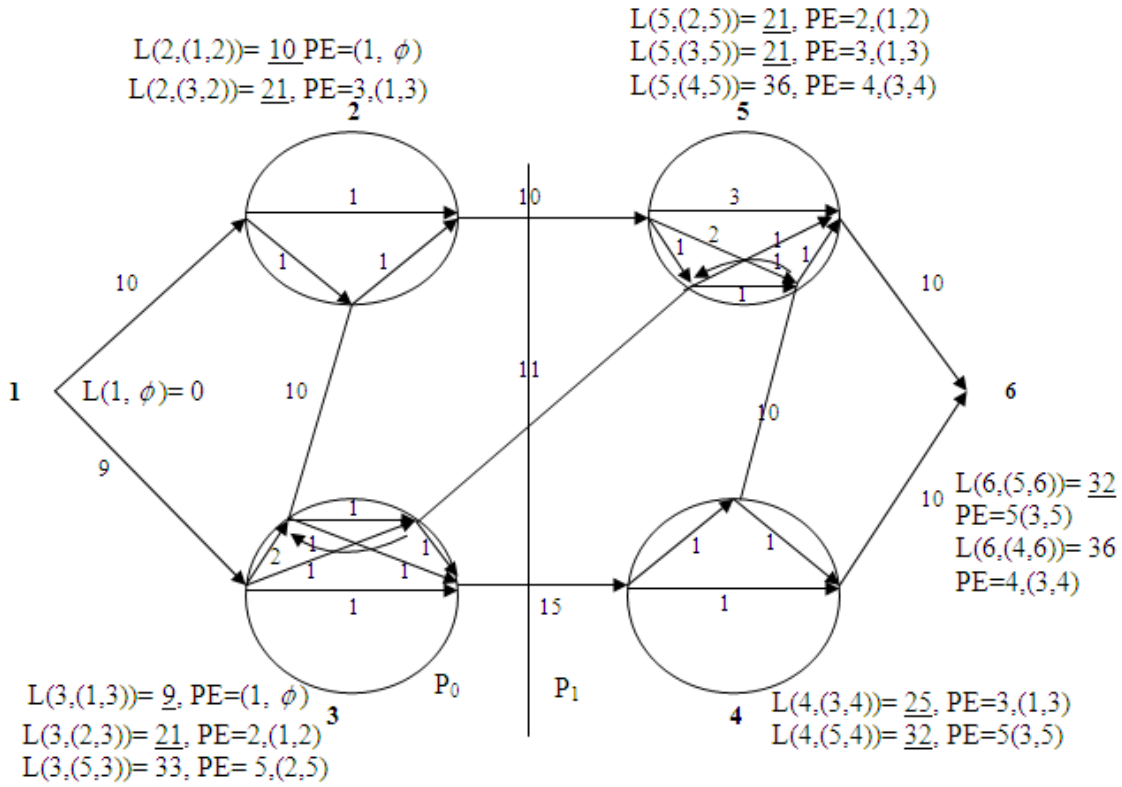
- The main processor P_0 receives T_0, A_0 (Figure 3), VE_0 and w_V. Sub-processor P_1 receives T_1, A_1 (Figure 2), VE_1 and w_V
- P_0 assigns $TE_0=VE_0$, $SE_0 = \phi$. P_1 assigns $TE_1=VE_1$, $SE_1=\phi$
- P_0 assigns L(1, ϕ)=0 ;L(2,(1,2))=;L(2,(3,2))=; L(3,(1,3)=; L(3,(2,3))=; L(3,(5,3))= ∞
- P_1 assigns L(4,(3,4))= ∞ ; L(4,(5,4)) = ∞ ; L(5,(2,3))= ∞ ; L(5,(3,5))= ∞ ; L(5,(4,5))= ∞ ; L(6,(5,6))= ∞ ; L(6,(4,6))= ∞

Results are shown in Figure 3



Fugire 3. Extended graph initialized on 2 processors

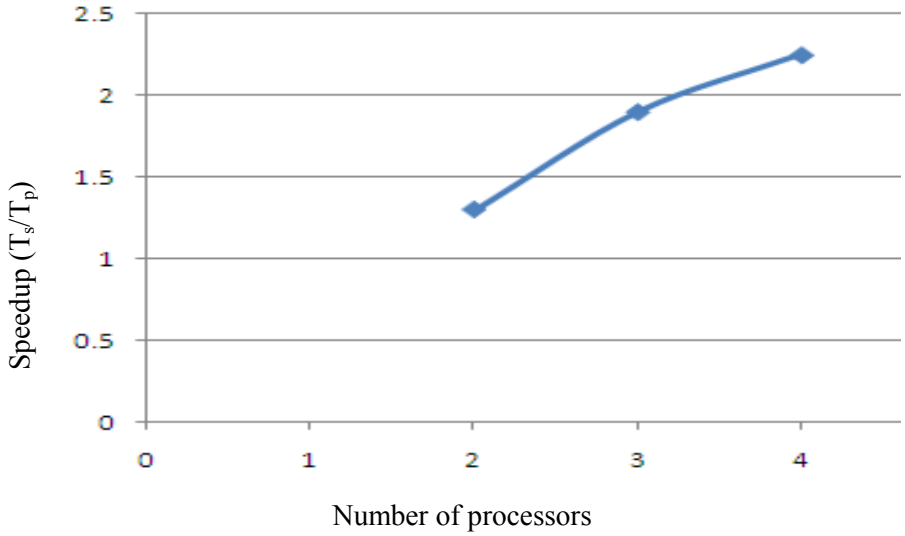
Keep do in that we have result as fugire 4 From 1 to 2: 1→2, length 10. From 1 to 3: 1→3, length 9. From 1 to 4: 1→3→4, length 25. From 1 to 5: 1→2→5 or 1→3→5, length 21. From 1 to 6: 1→3→5→6, length 32.



Fugire 4. Result graph P_0 receives

VII. THE EXPERIMENTAL RESULTS

Parallel algorithm finding the shortest path from a vertice to many vertices in the extended graph is built on k processors. The program written in Java with database administration system My SQL We experimentally sampled nodes as follows: The extended graph corresponds to 1500 nodes. The simulation result is shown in figure 5. This result demonstrates that the runtime of parallel algorithms is better than sequential algorithm.



Fugire 5. Chart performs the speedup of extended graph having 1500 nodes

VIII. CONCLUSION

In this paper, not only extended graph model is defined but also sequential and parallel algorithms finding the shortest path from a vertex to many vertices in extended transport network are presented in detail with particular experimental examples. In addition, the basic results are clearly systematized and proven.

REFERENCES

1. Chien Tran Quoc, *algorithms for finding the shortest paths in a general network*, University of DaNang, 2007. Journal of science and technology - University of DaNang, 12(61)/2012, 16-20.
2. Lau Nguyen Dinh, Tran Ngoc Viet, *Parallelizing algorithm dijkstra's finding the shortest paths from a vertex to all vertices*, Journal of science, University of Hue, 74B, 5, (2012), 81-92.
3. Lau Nguyen Dinh, Tran Ngoc Viet, *A parallel algorithm finding the shortest paths of multiple pairs of source and destination vertices in a graph*, Journal of science and technology - University of DaNang 9 (58),2012, 30-34.
4. Lau Nguyen Dinh, Tran Ngoc Viet, *Parallelizing algorithm finding the shortest paths of all vertices on computer cluster system*, Proceedings national Conference XVth "Some selected issues of Ìnormation Technology and Communications" 2012(accepted)
5. Chien Tran Quoc, Lau Nguyen Dinh, Trinh Nguyen Thi Tu, *Sequential and Parallel Algorithm by Postflow-Pull Methods to Find Maximum Flow*, Proceedings 13th International Conference on Computational Science and Its Applications, ISNB:978-0-7695-5045-9/13 $26.00 © 2013 IEEE, DOI 10.1109/ICCSA.2013.36.
6. Chien Tran Quoc, Tue Nguyen Mau, Viet Tran Ngoc, *Algorithm finding the shortest path on extended graph,* Proceeding of national Conference on Fundamental and Applied Infromation Technology Research (FAIR), Hue, Vietnam, 2013.
7. Chien Tran Quoc, Thanh Le Manh, Lau Nguyen Dinh, *Parallel algorithm to find maximum flow cost limits on extended traffic network,* Proceeding national Conference XVI "Some selected issues of Information Technology and Communications" Danang 14-15/11/2013(Accepted).
8. Chien Tran Quoc, Thanh Le Manh, Lau Nguyen Dinh, *Sequential and parallel algorithm by combined the push and pull methods to find maximum flow***,** Proceeding of national Conference on Fundamental and Applied Infromation Technology Research (FAIR), Hue, Vietnam, 2013.
9. Lau Nguyen Dinh, Chien Tran Quoc, Manh Le Thanh, P*arallel algorithm to divide optimal linear flow on extended traffic network,* Research, Development and Application on Information & Communication Technology, Ministry of Information & Communication of Vietnam, No 3, V-1, 2014.

SESSION

ALGORITHMS + FINITE STATE MACHINE + COMPUTATIONAL SCIENCE + LANGUAGES AND SYSTEMS

Chair(s)

TBA

A tail recursive scheme for exact real number computation based on LRT, GMP and FC++

J. Raymundo Marcial-Romero[1], J. A. Hernández[1], Vianney Muñoz[1], Guillermo De Ita[2], and Lourdes Rivas[1]
[1]Facultad de Ingeniería, UAEM, Toluca, México
[2]Facultad de Ciencias de la Computación, BUAP, Puebla, México

Abstract— *Language for Redundant Test (LRT) is a programming language for exact real number computation. The operational semantics of the language has been implemented in both, a functional language (Haskell) and in an Object oriented language (FC++) which provide mechanisms for lazy evaluation (also called call-by-need) and infinite lists manipulation. However, in the former, the time consumption of programs execution grows in such a way that even basic chaotic functions like the logistic map takes a thousand more time that a simple implementation in an imperative language like C. On the other hand, in FC++, the time consumption is improved considerably (equivalent to its C contrapart), at the cost of increasing the memory usage. In this paper we present a tail recursive scheme for basic operation in LRT that allows a linear growth of both time and memory.*

1. Introduction

There are several paradigms to compute with real numbers. Among them, we can cite floating point number arithmetic [17], interval analysis [16], algebraic manipulation [21] and exact real number computation [22].

The exact real number computation paradigm has several advantages compared to the others, for example, it avoids the rounding off errors that occurs in floating point arithmetic. Like interval analysis, it bounds the result, however it guarantees the computation of a smaller interval at each step of the computation which do not occur in interval analysis. Algebraic manipulations can be used in exact real number computation however, when no further reduction can be done an exact method is used to compute the solution compared to algebraic manipulation which has to turn to floating point arithmetic or interval analysis.

There have been several theoretical proposals to exact real number computations [20], [5], [11]. Most of them have succeeded to prove that the theory is sound and complete. However, when they have been implemented, none of them has achieved an efficient use of memory, polinomial execution time and a smooth translation from the theory to the practice.

To improve memory usage and execution time some implementations such as IRRAM [17], MPFR [6] and RealLib [9] have lost the elegancy of functional programming and especially, they have slightly deviated from the theory. Although we consider that faster implementations is what is required in practice, we believe that there is increased confidence in the correctness of an implementation the closer it is to the original theory. A pair of implementations which have not deviated from the theory to practice are Era [2] and a corecursive sign digit representation [3], implemented in Objective Caml and Coq respectively. However, Era, as said by the authors, is slower than IRRAM since C++ generally outputs when compiling more efficient code than Objetive Caml. Respect to the corecursive implementation, although an excellent theory is presented, the efficiency is never mentioned.

A further well established theory for exact real number computation is LRT (Language for Redundant Test) [11]. It has been proved that any computable first order function can be defined in LRT [14]. Moreover, an implementation of LRT in Haskell has been presented by Marcial et. al. [12]. However, its efficiency compared to even a sign digit representation [19] is questionable. In Marcial et. al. [13] an implementation of a basic calculator based on LRT and programmed using FC++ and GMP was presented. It was shown that the execution time improved compared to the Haskell implementations however compared to the fastest implementation now a days such as iRRAM it was thousands of time slower. The main problems have been: 1) the scheme used in recursive calls and 2) the algorithms implemented. In this paper, we show that if a tail recursive scheme is used, LRT can be as efficient as iRRAM at least for basic operations which allow to compute some chaotic functions such as the logistic map.

The paper is divided as follows, in Section 2 the language LRT is defined. In Section 3 a brief explanation of the tail recursive scheme is presented. In section 4 some results are presented. Finally the conclusions and further work is discussed.

2. The LRT Language

2.1 Syntax

The language LRT is a variant of Real PCF [5] and an extension of PCF (Programming Computable Functions) [18] with a ground type for real numbers and suitable primitive functions for real-number computation. Its raw syntax is given

by

$$\begin{array}{rcl} x & \in & Variable, \\ t & ::= & \mathtt{nat} \mid \mathtt{bool} \mid \mathtt{I} \mid t \to t, \\ P & ::= & x \mid \mathtt{n} \mid \mathtt{true} \mid \mathtt{false} \mid (+1)(P) \mid (-1)(P) \mid \\ & & (=0)(P) \mid \mathtt{if}\ P\ \mathtt{then}\ P\ \mathtt{else}\ P \mid \mathtt{cons}_{[\underline{a},\overline{a}]}(P) \mid \\ & & \mathtt{tail}_{[\underline{a},\overline{a}]}(P) \mid \mathtt{rtest}_{l,r}(P) \mid \lambda x : t.P \mid PP \mid \mathtt{Y}P, \end{array}$$

where $Variable$ is a set of variables, t represents a set of types, in this case the language has three ground types, the natural numbers type (represented by `nat`), the booleans (represented by `bool`) and the unit real number type (represented by `I` which denotes the set of intervals in $[-1,1]$, as it was shown in [10] the complete computable real line can be easily represented in this language, even more the implementation presented here considers the complete real line). The type $t \to t$ denotes higher order types. The constructs of the language (represented by P) are the variables (represented by x), the constants for natural numbers and booleans (represented by $\mathtt{n}, \mathtt{true}$ and `false`), the succesor, predecesor and equal test for zero operations for naturals numbers ((+1), (-1) and (=0)), the classical `if` operator; three operation for exact real number computation (`cons`, `tail` and `rtest`) where the subscripts of the constructs `cons` and `tail` are rational intervals (sometime written as a or $[\underline{a},\overline{a}]$) and those of `rtest` are rational numbers. The last three constructors of the languages are those of the lambda calculus ($\lambda x : t.P, PP$ and $\mathtt{Y}P$) where the first denotes abstraction, the second application and the third recursion.

Because the intention of this paper is not to present the denotational semantics of the language which is based on powerdomains [11], we just present the mathematical objects which describe the $\mathtt{cons}, \mathtt{tail}$ and `rtest` constructors. The others are the well known PCF constructors and can be consulted at [7], [18] .

Let $D \subseteq [-1,1]$, the function $\mathtt{cons}_a \colon D \to D$ is the unique increasing affine map with image the interval a, i.e.,

$$\mathtt{cons}_{[\underline{a},\overline{a}]}([\underline{x},\overline{x}]) = \left[\frac{\overline{a}-\underline{a}}{2}\underline{x} + \frac{\overline{a}+\underline{a}}{2}, \frac{\overline{a}-\underline{a}}{2}\overline{x} + \frac{\overline{a}+\underline{a}}{2}\right]$$

That is, rescale and translate the interval $[-1,1]$ so that it becomes $[\underline{a},\overline{a}]$, and define $\mathtt{cons}_{[\underline{a},\overline{a}]}([\underline{x},\overline{x}])$ to be the interval which results from applying the same rescaling and translation to $[\underline{x},\overline{x}]$. In order to keep the notation simple, when the context permits we use x to represent $[\underline{x},\overline{x}]$, meaning that the same operation is applied to both end points of the interval obtained, for example the `cons` function can be written as:

$$\mathtt{cons}_{[\underline{a},\overline{a}]}(x) = \frac{\overline{a}-\underline{a}}{2}x + \frac{\overline{a}+\underline{a}}{2} \tag{1}$$

The function $\mathtt{tail}_{[\underline{a},\overline{a}]}(x) \colon D \to D$ is a left inverse, i.e.

$$\mathtt{tail}_a(\mathtt{cons}_a(x)) = x.$$

More precisely, the following left inverse is taken, where κ_a is $\overline{a}-\underline{a}$ and τ_a is $\overline{a}+\underline{a}$:

$$\mathtt{tail}_{[\underline{a},\overline{a}]}(x) = \max(-1, \min((2x-\tau_a)/\kappa_a, 1)).$$

This definition guarantees that the range of the `tail` function is in the interval [-1,1]. The details of why this is a convenient definition can be consulted in [5]. It is worthy to mention that an infinite shrinking sequence of `cons` intervals represent a real number in the interval $[-1,1]$, the operational semantics defined below gives a rule for constructing a real number.

The definition of the function $\mathtt{rtest}_{l,r} : D \to \{\mathtt{true}, \mathtt{false}\}$, where $l < r$ are rational numbers, can be formulated as

$$\mathtt{rtest}_{l,r}(x) = \begin{cases} \mathtt{true}, & \text{if } x \subseteq (-\infty, l], \\ \mathtt{true}\ \mathit{or}\ \mathtt{false}, & \text{if } x \subseteq (l, r), \\ \mathtt{false}, & \text{if } x \subseteq [r, \infty). \end{cases} \tag{2}$$

The function $\mathrm{rtest}_{l,r}$ is operationally computable because, for any argument x given intensionally as a shrinking sequence of **cons** intervals, the computational rules systematically establish one of the semidecidable conditions $l < \overline{x}$ and $\underline{x} < r$ where l, r are rational numbers.

2.2 Operational Semantics

We consider a small-step style operational semantics for our language. We define the one-step reduction relation $\to$ to be the least relation containing the one-step reduction rules for evaluation of PCF [18] together with those given below.

We first need some preliminaries. For intervals a and b in $[-1,1]$, we define

$$ab = \mathsf{cons}_a(b),$$

where cons is the function defined previously. This operation is associative, and has the interval $[-1,1]$ (denoted by $\bot$) as its neutral element [5]:

$$(ab)c = a(bc), \qquad a\bot = \bot a = a.$$

In the interval domain literature [1], $a \sqsubseteq b$ iff $b \subseteq a$. Moreover,

$$a \sqsubseteq b \iff \exists c \in D.\ ac = b,$$

and this c is unique if a has non-zero length. In this case we denote c by

$$b \setminus a.$$

For intervals a and b, we define

$$a \le b \iff \overline{a} \le \underline{b}$$

and

$$a \uparrow b \iff \exists c.\ a \le c \text{ and } b \le c.$$

With this notation, the rules for Real PCF as defined in [5] are:

(1) $\mathbf{cons}_a(\mathbf{cons}_b M) \to \mathbf{cons}_{ab} M$

(2) $\mathbf{cons}_a M \to \mathbf{cons}_a M'$

(3) $\mathbf{tail}_a(\mathbf{cons}_b M) \to \mathbf{Ycons}_{[-1,0]}$ if $b \leq a$

(4) $\mathbf{tail}_a(\mathbf{cons}_b M) \to \mathbf{Ycons}_{[0,1]}$ if $b \geq a$

(5) $\mathbf{tail}_a(\mathbf{cons}_b M) \to \mathbf{cons}_{b\setminus a} M$ if $a \sqsubseteq b$ and $a \neq b$

(6) $\mathbf{tail}_a(M) \to \mathbf{tail}_a(M')$

(7) $\mathbf{if\ true}\ M\ N \to M$

(8) $\mathbf{if\ false}\ M\ N \to N$

(9) $\mathbf{if}\ M\ N_1\ N_2 \to \mathbf{if}\ M'\ N_1\ N_2$

For our language LRT, we add:

(10) $\mathbf{rtest}_{l,r}(\mathbf{cons}_a\ M) \to \mathbf{true}$ if $\overline{a} < r$

(11) $\mathbf{rtest}_{l,r}(\mathbf{cons}_a\ M) \to \mathbf{false}$ if $l < \underline{a}$

(12) $\mathbf{rtest}_{l,r}\ M \to \mathbf{rtest}_{l,r}\ M'$ if $M \to M'$.

Remarks:

1) Rule (1) plays a crucial role and amounts to the associativity law. The idea is that both a and b give partial information about a real number, and ab is the result of gluing the partial information together in an incremental way. See [5] for a further discussion including a geometrical interpretation.
2) Rules (2), (6), (9) and (12) are applied whenever any of the other rules are matched.
3) Rule (3) represents the fact that we already know that the rest of the real number we are looking for is an infinite sequence of the interval $[-1, 0]$, i.e.

$$\mathbf{Ycons}_{[-1,0]} = \mathbf{cons}_{[-1,0]}(\mathbf{cons}_{[-1,0]}(\ldots))$$

4) Rule (4) is similar to rule (3).
5) Rule (5) is applied when the partial information accumulated at some point contains the interval of the next input.
6) Rules (7) and (8) are the classical conditional rules.
7) Notice that if the interval a is contained in the interval $[l, r]$, rules (10) and (11) can be applied.
8) Rules (10)-(12) cannot be made deterministic given the particular computational adequacy formulation which is proved in [11].
9) In practice, one would like to avoid divergent computations by considering a strategy for application of the rules. In [11] total correctness of basic algorithms and in [15] total correctness of first order functions are shown, hence any implementation of any strategy will be correct.

For a deeper discussion of the relation between the operational and denotational semantics of LRT, the reader is referred to [11], [15].

3. The Basic Calculator

The basic calculator consists of the operations: addition, subtraction, multiplication and division. In order to describe how an algorithm in LRT works we present a particular example. The average function defined by:

$$x \oplus y = \frac{x+y}{2}$$

can be implemented in LRT as follows:

```
faverage(x, y)  =
  if rtest_{l,c} (x)
   then
    if rtest_{l,c} (y)
     then Cons_L(faverage(Tail_L x, Tail_L y))
    else
      if rtest_{c,r} (y)
       then Cons_{C_1}(faverage(Tail_L x, Tail_C y)
      else Cons_C(faverage(Tail_L x, Tail_R y))
   else
    if rtest_{c,r} (x)
    then
      if rtest_{l,c} (y)
       then Cons_{C_1}(faverage(Tail_C x, Tail_C y))
      else
        if rtest_{c,r} (y)
         then Cons_C(faverage(Tail_C x, Tail_C y))
        else Cons_{C_2}(faverage(Tail_C x, Tail_R y))
   else
      if rtest_{l,c} (y)
       then Cons_C(faverage(Tail_R x, Tail_L y))
      else
        if rtest_{c,r} (y)
         then Cons_{C_2}(faverage(Tail_R x, Tail_C y))
        else Cons_R(faverage(Tail_R x, Tail_R y))
```

where

$l = -1/2, \quad c = 0, \quad r = 1/2,$
$L = [-1, 0], \quad C = [-1/2, 1/2],$
$R = [0, 1], \quad C_1 = [-3/4, 1/4], \quad C_2 = [-1/4, 3/4].$

The idea behind this program is as follows. If both x and y are in the interval L, then we know that $x \oplus y$ is in the interval L, if both x and y are in the interval R, then we know that $x \oplus y$ is in the interval R, and so on. The boundary cases are taken care of by the `rtest` conditional.

What is interesting is that, despite the use of the multi-valued construction `rtest`, the overall result of the computation is single valued. In other words, different computation paths will give different shrinking sequences of intervals, but all of them will shrink to the same number. A proof of this fact and the correctness of the program is provided in [10]. This can be seen as follows: an unfolding of $1/2 \oplus 1/2$ gives the expresion $\mathtt{Cons_R(faverage(0 \oplus 0))}$. This means that the result of the operation is in the interval $R = [0, 1]$. A second unfolding gives $\mathtt{Cons_R Cons_C(faverage(1 \oplus 1))}$, due to it is a call by

need language, the first two conses are reduced using Equation 1. This means that the result is in the interval $[1/3, 2/3]$. A repeated unfolding gives the required result $1/2$.

A translation to the above program to the functional object oriented language FC++ is hard but achievable, however the use of memory and execution time is completely different. It can be noticed that the above program does not have a base case like most of the recursive programs have, so an implementation in an eager evaluation will never stop. This is overcome by the functionality provided to C++ in FC++ which has a mechanism to evaluate what is needed of the infinite recursive calls. However, in each recursive call, FC++ follows what C++ does, creates a stack to store the parameters of the recursive calls. So an excessive number of recursive calls together with an increasing use of potentially infinite lists make the computer memory (no matter its size) come to an end.

Tail recursion allows to "optimize" the number of recursive calls getting rid of the constant number of calls. To implement tail recursive functions in LRT it was necessary to use the precision that a computation is required to be computed at. For example $(a \oplus b, 0.0001)$ means that the average of the number a and b is required with four digits of precision.

Considering the cases of $faverage$ program, the size of each interval in the $cons$ function is one. The application of equation 1 in rule (1) of the operational semantics gives a rate of convergence of $\frac{1}{2^n}$ in all possible execution branches of the program.

To compute the number of reductions of $cons$ in the average program, we need to know n in the equation:

$$precision = \frac{1}{2^n}$$

given by

$$n = \log_2 \left(\frac{1}{precision} \right)$$

The equivalent program of the `faverage` function using tail recursion in FC++ is presented below:

```
faverage(x, y, prec) =
  i = ceil(1/log(prec));
  for (j = 1; j < i; j++){
  if rtest_{l,c} (x)
   then
    if rtest_{l,c} (y)
     then resp = resp ::Cons_L
          x =Tail_L x
          y =Tail_L y
     else
      if rtest_{c,r} (y)
       then resp = resp ::Cons_{C_1}
         x =Tail_C x
         y =Tail_C y
       else resp = resp ::Cons_C
         x =Tail_C x
         y =Tail_R y
```

```
else
 if rtest_{c,r} (x)
 then
   if rtest_{l,c} (y)
    then resp = resp ::Cons_{C_1}
      x =Tail_C x
      y =Tail_C y
    else
     if rtest_{c,r} (y)
      then resp = resp ::Cons_C
           x =Tail_C x
           y =Tail_C y
      else resp = resp ::Cons_{C_2}
           x =Tail_C x
           y =Tail_R y
      else
   if rtest_{l,c} (y)
    then resp = resp ::Cons_C
      x =Tail_R x
      y =Tail_L y
    else
     if rtest_{c,r} (y)
      then resp = resp ::Cons_{C_2}
           x =Tail_R x
           y =Tail_C y
      else resp = resp ::Cons_R
          x =Tail_R x
          y =Tail_R y
}
return resp;
```

The tail recursive program establishes the number of `conses` to be reduced in order to evaluate the expression to the required precision. Instead of creating stacks in each call, we create a variable of `cons` type an accumulate the reduction on the variables. A similar analysis was made to the basic operations (subtraction, multiplication and division) similar to the one presented in this section. In the following section, we present the execution results of the implementation.

4. Results

We do three different comparisons. The first one shows an example of a chaotic function were this implementation gives exact results compared to the standard simple and double precision of the C programming language. The second comparison consider the time taken to compute the logistic map against iRRAM. Finally, the memory usage is compared with a previous implementation of LRT without tail recursion [8]. All the comparisons were performed on a MacBook with processor of 2.4 GHz Intel Core 2 Duo and memory of 2 GB.

4.1 The logistic Map

The logistic map is a function $f : [0, 1] \to [0, 1]$ defined by

$$f(x) = ax(1 - x)$$

for a given constant a. Devaney [4] stated that it was first considered as a model of population growth by Pierre Verhulstby in 1845. For example, a value 0.5 may represent 50% of the maximum population of cattle in a given farm. The problem is, given x_0, to compute the orbit

$$x_0, f(x_0), f(f(x_0)) \dots f^n(x_0), \dots,$$

which collects the population value of succesive generations. The purpose is to compute an initial segment of the orbit for a given initial population x_0. It has been identified that choosing $a = 4$ is a chaotic case. The main problem is that its value is sensitive to small variations of its variables. The result of computing orbits for the same initial value $x_0 = 0.671875$, in simple and double precision in the C programming language is shown in Table 1. Also, Table 1 shows the exact result and the value obtained using our calculator. As it can be noticed the tables are equal up to $n = 7$. From row 8th up to 39th the double, exact and LRT column report equal results. From row 40th the C double implementations show a small deviation from the exact result and at the last 60th row this deviation is more evident. It is worth to mention that every exact real number computation implementation must produce the correct result as is the case in our implementation. The main drawback is the execution time that our implementation takes to compute the orbits. However, in this first version of our implementation, the goal is not to look for the most efficient algorithms for exact real number computation. Instead, we wanted to show that it is possible to transit from the basic LRT theory to actual practice in a smooth way.

n	Simple precision	Double precision	LRT result	Exact result
0	0.671875	0.671875	0.671875	0.671875
1	0.881836	0.881836	0.881836	0.881836
2	0.416805	0.416805	0.416805	0.416805
3	0.972315	0.972315	0.972315	0.972315
4	0.107676	0.107676	0.107676	0.107676
5	0.384327	0.384327	0.384327	0.384327
6	0.946479	0.946479	0.946479	0.946479
7	0.202625	0.202625	0.202625	0.202625
8	**0.646272**	0.646273	0.646273	0.646273
9	0.914417	0.914416	0.914416	0.914416
10	0.313033	0.313037	0.313037	0.313037
11	0.860174	0.860179	0.860179	0.860179
12	0.481098	0.481084	0.481084	0.481084
13	0.998570	0.998569	0.998569	0.998569
14	0.005708	0.005716	0.005716	0.005716
15	0.022702	0.022735	0.022735	0.022735
16	0.088747	0.088875	0.088875	0.088875
17	0.323485	0.323907	0.323907	0.323907
18	0.875370	0.875965	0.875965	0.875965
19	0.436386	0.434601	0.434601	0.434601
20	0.983813	0.982892	0.982892	0.982892
25	0.652836	0.757549	0.757549	0.757549
30	0.934926	0.481445	0.481445	0.481445
35	0.848152	0.313159	0.313159	0.313159
39	0.014638	0.006038	0.006038	0.006038
40	0.057695	**0.024007**	0.024009	0.024009
45	0.991612	0.930952	0.930881	0.930881
50	0.042173	0.629401	0.625028	0.625028
55	0.108415	0.749775	0.615752	0.615752
60	0.934518	0.757153	0.315445	0.315445

Table 1

RESULTS OF COMPUTING THE LOGISTIC MAP FOR SIMPLE AND DOUBLE PRECISION IN THE C PROGRAMMING LANGUAGE, AND OUR IMPLEMENTATION AND THE EXACT RESULT. FROM VALUES $n = 8$ AND $n = 40$ THE SIMPLE AND DOUBLE PRECISION RESPECTIVELY DEVIATE FROM THE EXACT RESULT.

4.2 Comparing against iRRAM

Table 2 shows a comparison in terms of time with an implementation of the logistic map in iRRAM. Due to both of them iRRAM and LRT follows the exact real number computation, the results output by iRRAM are also correct (no rounding error is requiered). As it can be seen both implementations give results in milliseconds a difference to a non tail recursion implementation of LRT which for $n = 60$ takes at least 15 minutes in the computation.

4.3 Memory usage

The last comparison shows the memory usage between a tail recursion and a non-tail recursion implementation of LRT. Figure 1 shows a graph representation of the memory use in the implementation of the logistic map. As it can be seen, the tail recursion implementation uses less than 10% of memory instead of the non-tail implementation which consumes 100% of memory at f^{60}.

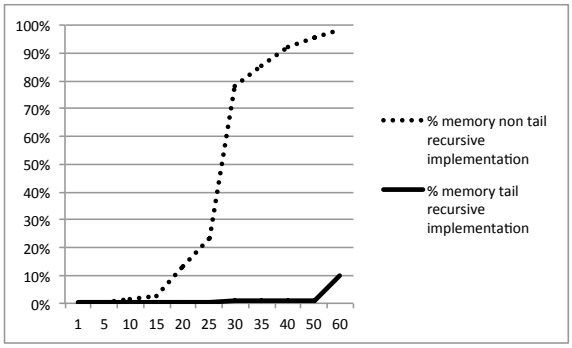


Fig. 1

USE OF MEMORY OF THE IMPLEMENTATION OF THE LOGISTIC MAP WITH AND WITHOUT TAIL RECURSION.

5. Conclusions

We have presented a tail recursive implementation of LRT in the FC++ programming language using the GMP library. Although C++ is an imperative language, FC++ is a functional C++ implementation, meaning that it allows a call by need evaluation and the definition of infinite lists. The precision

n	LRT with Tail Recursion	iRRAM	LRT with our tail recursion
0	0.2 msec	0.670 msec	2 msec.
1	4.7 msec	0.670 msec	3 msec.
2	10 msec	0.680 msec	5 msec
3	12 msec	0.690 msec	13 msec
4	13 msec	0.710 msec	22 msec
5	15 msec	0.710 msec	42 msec
6	18 msec	0.720 msec	61 msec
7	19 msec	0.720 msec	103 msec
8	20 msec	0.740 msec	128 msec
9	20 msec	0.750 msec	189 msec
10	25 msec	0.750 msec	222 msec
11	28 msec	0.760 msec	352 msec
12	31 msec	0.770 msec	607 msec
13	32 msec	0.800 msec	850 msec
14	34 msec	0.820 msec	854 msec
15	34 msec	0.900 msec	870 msec
20	51 msec	1.09 msec	3.53 sec
25	52 msec	1.16 msec	8.23 sec
30	67 msec	1.19 msec	39.49 sec
35	78 msec	1.2 msec	2.25 min
40	86 msec	1.37 msec	5.75 min
50	100 msec	1.38 msec	10.83 min
55	110 msec	1.39 msec	11.2 min
60	120 msec	1.64 msec	15.7 min

Table 2
RESULTS OF COMPUTING THE EXECUTION TIME OF LOGISTIC MAP AGAINST IRRAM, AND AN LRT IMPLEMENTATION WITHOUT TAIL RECURSION.

required to compute basic operations (addition, subtraction, multiplication and division) were established. The precision was used to implement the basic algorithms. It is well known that the basic algorithms allows the implementation of polinomial functions, i.e. of the form $a_n x^n + a_{n-i} x^{n-1} + \cdots a_1 x + a$, hence the proposal was proved with a chaotic function called the logistic map. The results shows an improvement compared with previous implementations of LRT and also an improvement compared with well established implementations like iRRAM. A further work is to establish the precision required to compute trigonometric functions possibly using Taylor series, e.g. the limit function has to be defined. The implementation of the limit function will allow to compute any first order computable function.

References

[1] S. Abramsky and A. Jung. Domain theory. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 3, pages 1–168. Clarendon Press, 1994.

[2] A. Bauer and I. Kavkler. Implementing real numbers with rz. In *Proceedings of the Fourth International Conference on Computability and Complexity in Analysis, CCA*, pages 365–384. ENTCS, 2008.

[3] A. Ciaffaglione and P. D. Gianantonio. A certified, corecursive implementation of exact real numbers. *Theoretical Computer Science*, 351(1):39–51, 2006.

[4] R. L. Devaney. *An Introduction to Chaotical Dynamical Systems*. Addison-Wesley, California, 2do edition, 1989.

[5] M. H. Escardó. PCF extended with real numbers. *Theoretical Computer Science*, 162(1):79–115, August 1996.

[6] P. Pélissier G. Hanrot, V. Lefévre and P. Zimmermann. The MPFR library. INRIA. http://mpfr.org.

[7] C. A. Gunter. *Semantics of Programming Languages*. The MIT Press, 1992.

[8] A. Lucatero, J.A. Hernández and J.Raymundo Marcial-Romero. A scientific calculator for exact real number computation based on LRT, GMP and FC++ , *Acta Universitaria*, 22 (NE-1). ISSN:0188-6266. pp: 35-41. March 2012.

[9] B. Lambov. The reallib project. BRICS, University of Aarhus. http://brics.dk/~barnie/RealLib.

[10] J. R. Marcial-Romero. *Semantics of a sequential language for exact real-number computation*. PhD thesis, University of Birmingham, December 2004.

[11] J. R. Marcial-Romero and M. H. Escardó. Semantics of a sequential language for exact real-number computation. *Theoretical Computer Science*, 379(1-2):120–141, 2007.

[12] J. R. Marcial-Romero, J. A. Hernández, and Héctor A. Montes-Venegas. Comparing implementations of a calculator for exact real number computation. *Proceedings of the Mexican International Conference on Computer Science, ENC*, pages 13–23. IEEE Computer Society Press, July 2009.

[13] J. R. Marcial-Romero, A. Lucatero, and J. A. Hernández. A GMP-FC++ implementation of a calculator for exact real number computation based on LRT. *Proceedings of the Seventh Latin American Workshop on Logic / Languages, Algorithms and New Methods of Reasoning 2011 , LANMR*, pages 71–82. CEUR Workshop Proceedings, 2011.

[14] J. R. Marcial-Romero and A. Moshier. Sequential real number computation and recursive relations. In *Proceedings of the Fourth International Conference on Computability and Complexity in Analysis, CCA*, pages 171–189. ENTCS, 2008.

[15] J. R. Marcial-Romero and A. Moshier. Sequential real number computation and recursive relations. *Mathematical Logic Quarterly*, 54(5):492–507, 2008.

[16] R. E. Moore *Interval Analysis*. Prentice-Hall, Englewood Cliffs, 1966

[17] N. M uller. iRRAM - exact arithmetic in C++. Universitat Trier. http://www.informatik.uni-trier.de/iRRAM.

[18] G. D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5(1):223–255, 1977.

[19] Dave Plume. A calculator for exact real number computation. 4th Year Project Report, Department of Computer Science and Artificial Intelligence, University of Edinburgh, 1998.

[20] P. J. Potts, A. Edalat, and M.H Escardó. Semantics of exact real arithmetic. In *In Proceedings of the Twelveth Annual IEEE Symposium on Logic In Computer Science*. IEEE Computer Society Press, 1997.

[21] A. Schalk. *Algebras for Generalized Construction*. PhD thesis, TU Darmstadt, 1993.

[22] K. Weihrauch. *Computable Analysis*. Springer, 2000.

Tackling Sequences From Prudent Self-Avoiding Walks

Shanzhen Gao, Keh-Hsun Chen
Department of Computer Science, College of Computing and Informatics
University of North Carolina at Charlotte, Charlotte, NC 28223, USA
Email: sgao3@uncc.edu, chen@uncc.edu

***Abstract*—A self-avoiding walk (SAW) is a sequence of moves on a lattice not visiting the same point more than once. A SAW on the square lattice is prudent if it never takes a step towards a vertex it has already visited. Prudent walks differ from most subclasses of SAWs that have been counted so far in that they can wind around their starting point. Some interesting problems and sequences arising from prudent walks of one-sided and two-sided are discussed in this paper. A few methods such as computational, kernel, generating function, recurrence relation and constructive method are applied to our study. Several open problems are posted.**

Keywords: Self-avoiding walk, prudent self-avoiding walk, generating function, kernel method, integer sequence

I. Introduction

A well-known long standing problem in combinatorics and statistical mechanics is to find the generating function for self-avoiding walks (SAW) on a two-dimensional lattice, enumerated by perimeter. A SAW is a sequence of moves on a square lattice which does not visit the same point more than once. It has been considered by more than one hundred researchers in the pass one hundred years, including George Polya, Tony Guttmann, Laszlo Lovasz, Donald Knuth, Richard Stanley, Doron Zeilberger, Mireille Bousquet-Mélou, Thomas Prellberg, Neal Madras, Gordon Slade, Agnes Dittel, E.J. Janse van Rensburg, Harry Kesten, Stuart G. Whittington, Lincoln Chayes, Iwan Jensen, Arthur T. Benjamin, and others. More than three hundred papers and a few volumes of books were published in this area. A SAW is interesting for simulations because its properties cannot be calculated analytically. Calculating the number of self-avoiding walks is a common computational problem [1], [2], [3].

In order to present our problems and results clearly and efficiently, we introduce some notations in the following.

East step: E or $\rightarrow$ or $(1,0)$, x-step

You can see more in the table below:

$(0,1)$	$(1,0)$	$(1,1)$	$(0,-1)$
$\uparrow$	$\rightarrow$	$\nearrow$	$\downarrow$
N	E	NE	S

$(-1,0)$	$(-1,-1)$	$(-1,1)$	$(1,-1)$
$\leftarrow$	$\swarrow$	$\nwarrow$	$\searrow$
W	SW	NW	SE

$\uparrow^{\geq k}$: k or more than k consecutive $\uparrow$ steps

$\uparrow^{=k}$: k consecutive $\uparrow$ steps

avoiding $\uparrow^{\geq k}$: no k or more than k consecutive $\uparrow$ steps

avoiding $\uparrow^{=k}$: no k consecutive $\uparrow$ steps, but can have more than or less than k consecutive $\uparrow$ steps

$\lfloor x \rfloor$: the largest integer not greater than x, $floor(x)$

$\lceil x \rceil$: is the smallest integer not less than x, $ceiling(x)$

$[x^n]f(x)$ denotes the coefficient of x^n in the power series expansion of a function $f(x)$.

$[x^m y^n]f(x,y)$ denotes the coefficient of $x^m y^n$ in the power series expansion of a function $f(x,y)$.

$\binom{n}{r}$ the number of combinations of n things r at a time.

$$\begin{aligned}\binom{n}{r} &= \frac{n!}{(n-r)!r!} \\ &= \binom{n}{n-r} \\ &= \binom{n-1}{r-1} + \binom{n-1}{r}\end{aligned}$$

$$\binom{-n}{r} = (-1)^r \binom{n+r-1}{r}$$

In the past few decades, many mathematicians have studied the following two classical problems:

Classical Problem 1

What is the number of SAWs from $(0,0)$ to $(n-1,n-1)$ in an $n \times n$ grid, taking steps from $\{\uparrow,\downarrow,\leftarrow,\rightarrow\}$?

Donald Knuth claimed that the number is between 1.3×10^{24} and 1.6×10^{24} for $n = 11$ and he did not believe that he would ever in his lifetime know the exact answer to this problem in 1975. However, after a few years, Richard Schroeppel pointed out that the exact value is $1,568,758,030,464,750,013,214,100 = 2^2 3^2 5^2 31 \times 115\,422\,379 \times 487\,148\,912\,401$ [4], [5], [6]. It is still an unsolved problem for $n > 25$.

Classical Problem 2

What is the number $f(n)$ of n-step SAWs, on the square lattice, taking steps from $\{\uparrow,\downarrow,\leftarrow,\rightarrow\}$?

The number $f(n)$ is known for $n \leq 71$ [4], [5], [7], [8].

It is clear that

$$2^n \leq f(n) \leq 4 \times 3^{n-1}$$

$$f(m+n) \leq f(m)f(n)$$

There exists a constant C such that

$$\lim_{n\to\infty} f(n)^{1/n} = \inf_n [f(n)]^{1/n} = C.$$

$C = 2.64$ (up to 71 steps have been counted).

$C = 2.638$ (up to 91 steps have been counted).

$$f(n) \approx 2.638^n$$

The number of SAWs/ the number of total walks:

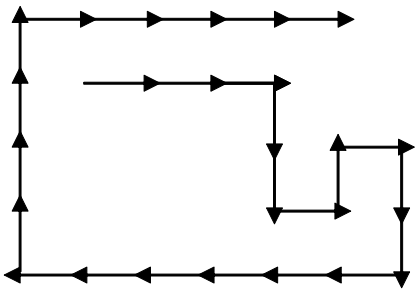
$$\frac{1}{1200} \text{ for } n = 20$$

$$\frac{1}{2.4 \times 10^8} \text{ for } n = 50$$

A recently proposed model called prudent self-avoiding walks (PSAW) was first introduced to the mathematics community in an unpublished manuscript of Préa, who called them *exterior walks*. A prudent walk is a connected path on square lattice such that, at each step, the extension of that step along its current trajectory will never intersect any previously occupied vertex. Such walks are clearly self-avoiding [9], [10], [11], [12], [13]. We will talk about some sequences arising from PSAWs in the following.

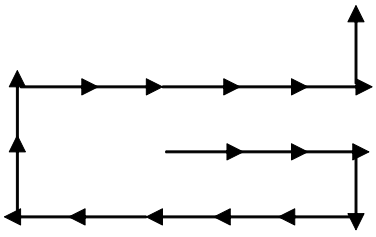
II. Prudent Self-Avoiding Walks: Definitions and Examples

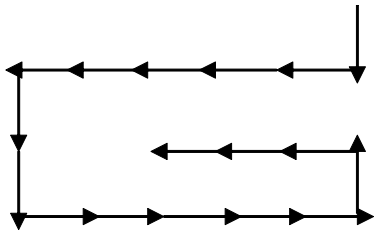
A PSAW is a proper subset of SAWs on the square lattice. The walk starts at $(0, 0)$, and the empty walk is a PSAW. A PSAW grows by adding a step to the end point of a PSAW such that the extension of this step - by any distance - never intersects the walk. Hence the name prudent. The walk is so careful to be self-avoiding that it refuses to take a single step in any direction where it can see - no matter how far away - an occupied vertex. The following walk is a PSAW.



A. Properties of a PSAW

Unlike SAW, PSAW are usually not reversible. There is such an example in the following figure.

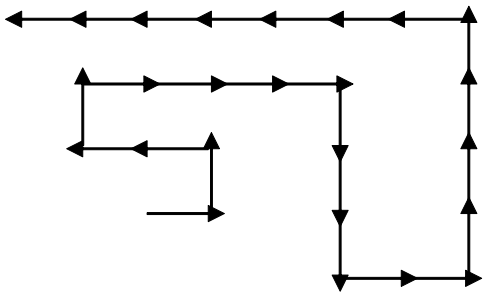




Each PSAW possesses a minimum bounding rectangle, which we call *box*. Less obviously, the endpoint of a prudent walk is always a point on the boundary of the box. Each new step either inflates the box or walks (prudently) along the border. After an inflating step, there are 3 possibilities for a walk to go on. Otherwise, only 2.

In a *one-sided* PSAW, the endpoint lies always on the top side of the box. The walk is *partially directed*.

A prudent walk is *two-sided* if its endpoint lies always on the top side, or on the right side of the box. The walk in the following figure is a two-sided PSAW.



III. Some Sequences Arising from One-sided PSAWs

Sequence 1

What is the number (say $f(n)$) of one-sided n-step prudent walks, taking steps from $\{\uparrow, \leftarrow, \rightarrow\}$?

The generating function equals

$$\sum_{n \geq 0} f(n)\, t^n = \frac{1+t}{1-2t-t^2}$$
$$= 1 + 3t + 7t^2 + 17t^3 + 41t^4 + 99t^5 + ...$$

Also,

$$f(n) = 2f(n-1) + f(n-2)$$
$$= \frac{(1-\sqrt{2})^n + (1+\sqrt{2})^n}{2}$$
$$= \begin{bmatrix} 1 & 0 \end{bmatrix} \left(\sum_{k=0}^{n+1} \binom{n+1}{k} \begin{bmatrix} 0 & 1 \\ 2 & 0 \end{bmatrix}^k \right) \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

We obtain sequence $A001333$ of the On-Line Encyclopedia of Integer Sequences.[15, A001333]

Sequence 2

The number of one-sided n-step prudent walks, starting from $(0,0)$ and ending on y-axis, taking steps from $\{\uparrow, \leftarrow, \rightarrow\}$ is

$$1+\sum_{k=1}^{\lfloor (n-1)/2 \rfloor} \sum_{i=1}^{\min\{n-2k,k\}} \binom{n-2k+1}{i}\binom{k-1}{k-i}\binom{n-k-i}{k}.$$

We obtain sequence $A136029$.[15, A136029]

Sequence 3

Consider the number of one-sided prudent walks starting from $(0,0)$ to (x,y), taking steps from $\{\uparrow, \leftarrow, \rightarrow\}$. The number of such walks with $k+x$ right $\rightarrow$ steps, k left $\leftarrow$ steps and y up $\uparrow$ steps, is

$$\sum_{i=1}^{\min\{y,k+x\}} \binom{y+1}{i}\binom{k+x-1}{k+x-i}\binom{y+k-i}{k}.$$

If $k=2$ and $x=y=n$, we obtain sequence $A119578$.[15, A119578]

Sequence 4

The number of one-sided n-step prudent walks, from $(0,0)$ to (x,y), ($n-x-y$ is even) taking steps from $\{\uparrow, \leftarrow, \rightarrow\}$ is

$$\sum_{i=0}^{\min\{y,\frac{n+x-y}{2}\}} \binom{y+1}{i}\binom{\frac{n+x-y}{2}-1}{\frac{n+x-y}{2}-i}\binom{\frac{n-x+y}{2}-i}{\frac{n-x-y}{2}}.$$

If $x=y=3$, we obtain sequence $A163761$.[15, A163761]

Sequence 5

What is the number of the one-sided n-step prudent walks, avoiding k or more consecutive east steps, $\rightarrow^{\geq k}$?

The generating function equals

$$\frac{1+t-t^k}{1-2t-t^2+t^{k+1}}$$

If $k=1$,

$$\frac{1+t-t^k}{1-2t-t^2+t^{k+1}} = \frac{1}{1-2t} = 1+2t+4t^2+8t^3+16t^4+32t^5+...$$

If $k=2$, we obtain sequence $[15, A006356]$:

$1, 3, 6, 14, 31, 70, 157, 353, 793, 1782, 4004, 8997, 20216, ...$

It also counts the number of paths for a ray of light that enters two layers of glass and then is reflected exactly n times before leaving the layers of glass.

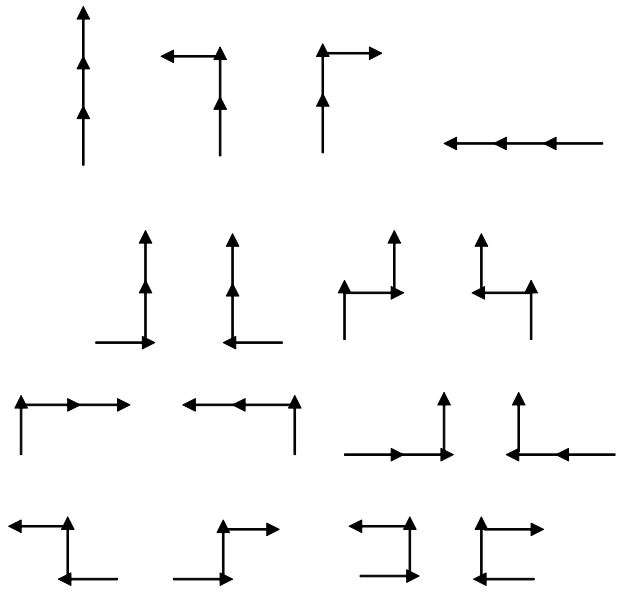
If $k=3$, we obtain sequence [15, A052967]:

$$1, 3, 7, 16, 38, 89, 209, 491, 1153, 2708, 6360, ...$$

If $k=4$, we obtain sequence [15, A190360]:

$$1, 3, 7, 17, 40, 96, 229, 547, 1306, 3119, 3119, 7448, ...$$

For the case $k=3$ in the above theorem, there are 16 walks as follows:



Sequence 6

The number of one-sided n-step prudent walks, taking steps from $\{\uparrow, \leftarrow, \rightarrow, \nearrow\}$ equals

$$\frac{5+\sqrt{17}}{2\sqrt{17}}\left(\frac{3+\sqrt{17}}{2}\right)^n - \frac{5-\sqrt{17}}{2\sqrt{17}}\left(\frac{3-\sqrt{17}}{2}\right)^n.$$

We obtain sequence $A055099$.[15, A055099]

Sequence 7

What is the number of one-sided n-step prudent walks, taking steps from $\{\rightarrow, \leftarrow, \uparrow, \nearrow, \searrow\}$?

The generating function is

$$\frac{1+t}{1-4t-3t^2}.$$

We obtain sequence $A126473$.[15, A126473]

Sequence 8

What is the number of one-sided n-step prudent walks in the first quadrant, starting from $(0,0)$ and ending on the y-axis, taking steps from $\{\uparrow, \leftarrow, \rightarrow\}$?

The generating function is

$$\frac{1}{2t^3}((1+t)(1-t)^2 - \sqrt{(1-t^4)(1-2t-t^2)}).$$

Sequence 9

What is the number of one-sided n-step prudent walks *exactly* avoiding $\leftarrow^{=k}$, taking steps from $\{\uparrow, \leftarrow, \rightarrow\}$?

The generating function equals

$$\frac{1+t-t^k+t^{k+1}}{1-2t-t^2+t^{k+1}-t^{k+2}}.$$

If $k=1$, we obtain sequence $A078061$.[15, A078061]

Sequence 10

What is the number of one-sided n-step prudent walks exactly avoiding $\leftarrow^{=k}$ and $\uparrow^{=k}$ (both at the same time)?

The generating function is

$$\frac{1+t-2t^k+2t^{k+1}}{1-2t-t^2+2t^{k+1}-2t^{k+2}}.$$

For $k=1$,

$$f(n)=\left(2^{n+2}-(-1)^{\lfloor n/2\rfloor}+2(-1)^{\lfloor (n+1)/2\rfloor}\right)/5,$$

also,

$$f(n)=2f(n-1)-f(n-2)+2f(n-3)$$
$$\text{with } f(1)=1, f(2)=3, f(3)=7.$$

This is sequence $A007909$.[15, A007909]

IV. Some Sequences Arising from Two-sided PSAWs

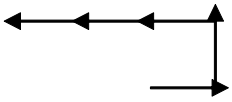
What is the number of two-sided, n-step prudent walks ending on the top side of their box avoiding both patterns $\leftarrow^{\geq 2}$, $\downarrow^{\geq 2}$ (both at the same time), taking steps from $\{\uparrow,\downarrow,\leftarrow,\rightarrow\}$?

Theorem 1. *The generating function (say $T(t,u)$) of the above two-sided prudent walks ending on the top side of their box satisfies*

$$\left(1-t^2u-\frac{tu}{u-t}\right)T(t,u)=1+tu+T(t,t)t\frac{u-2t}{u-t}, \quad (1)$$

where u counts the distance between the endpoint and the north-east (NE) corner of the box.

For instance, in the following figure, a walk takes 5 steps, and the distance between the endpoint and the north-east corner is 3. So we can use t^5u^3 to count this walk.



Outline of the proof of the theorem:

Case 1: Neither the top nor the right side has ever moved; the walk is only a west step. This case contributes 1 to the generating function.

Case 2: The last inflating step goes east. This implies that the endpoint of the walk was on the right side of the box before that step. After that east step, the walk has made a sequence of north steps to reach the top side of the box. Observe that, by symmetry, the series $T(t,u)$ also counts walks ending on the right side of the box by the length and the distance between the endpoint and the north-east corner. These two observations give the generating function for this class as $T(t,t)$.

Case 3: The last inflating step goes north. After this step, there is either a west step or a bounded sequence of East steps. This gives the generation function for this class as

$$\left(t^2u+\frac{tu}{u-t}\right)T(t,u)-\frac{t^2}{u-t}T(t,t)$$

Putting the three cases together, we get the generating function (1) for $T(t,u)$.

Solve this generating function for $T(t,u)$ using the Kernel Method:

From

$$\left(1-t^2u-\frac{tu}{u-t}\right)T(t,u)=1+tu+T(t,t)\left(t-\frac{t^2}{u-t}\right),$$

we can get

$$(1-tu)\left(u-tu-t-t^2u^2+t^3u\right)T(t,u)$$
$$=(u-t)(1-tu)(1+tu)-T(t,t)(1-tu)t(2t-u)$$

Set $(1-tu)\left(u-tu-t-t^2u^2+t^3u\right)=0$, then there is only one *power series* solution for u

$$u=\frac{1}{2t^2}\left(1-t+t^3-\sqrt{(1-t-t^3)^2-4t^4}\right).$$

Let U be this solution,

$$U=U(t)=\frac{1}{2t^2}\left(1-t+t^3-\sqrt{(1-t-t^3)^2-4t^4}\right). \quad (2)$$

Set

$$(1+tu)(u-t)(1-tu)+T(t,t)(1-tu)t(u-2t)=0,$$

and replace u by U:

$$T(t,t)=(1+tU)\frac{t-U}{t(U-2t)}. \quad (3)$$

From

$$(1-tu)\left(u-t-tu-t^2u^2+t^3u\right)T(t,u)$$
$$=(u-t)(1-tu)(1+tu)-T(t,t)(1-tu)t(2t-u)$$

get

$$T(t,u)=\frac{(t-u)(1-tu)(1+tu)}{(1-tu)(u-t-tu-t^2u^2+t^3u)}+$$
$$\frac{T(t,t)(1-tu)t(2t-u)}{(1-tu)(u-t-tu-t^2u^2+t^3u)}$$

Replace $T(t,t)$ by (3). Now

$$T(t,u)=\frac{(1+tu)(u-t)}{u-t-tu-t^2u^2+t^3u}$$
$$-\frac{(1+tU)(U-t)(1-tu)(u-2t)}{(U-2t)(1-tu)(u-t-tu-t^2u^2+t^3u)}$$

where $U(t)$ has been defined in (2).

Sequence 11

Notice that $T(t,1)$ is the generating function of the number of two-sided n-step prudent walks ending on the top side of their box avoiding both patterns $\leftarrow^{\geq 2}$, $\downarrow^{\geq 2}$, taking steps from $\{\uparrow,\downarrow,\leftarrow,\rightarrow\}$, thus $T(t,1)=$

$$\frac{1}{2t(1-2t-t^2+t^3)(1-2t-2t^3)}\times$$
$$\left((1-2t)(1-t)\sqrt{(1-t-t^3)^2-4t^4}-\right.$$
$$\left.(1+t)\left(1-7t+14t^2-11t^3+10t^4-4t^5\right)\right)$$
$$=1+3t+6t^2+15t^3+35t^4+83t^5+195t^6+...$$

Sequence 12

Note that $T(t,0)$ is the generating function of the number of two-sided n-step prudent walks ending at the north-east corner of their box avoiding both patterns $\leftarrow^{\geq 2}$, $\downarrow^{\geq 2}$, taking steps from $\{\uparrow,\downarrow,\leftarrow,\rightarrow\}$, so $T(t,0) =$

$$\frac{(1-t)\sqrt{(1-t-t^3)^2-4t^4}-1+3t-t^2+t^3+t^4}{(1-2t-2t^3)t}$$
$$= 1+2t+4t^2+10t^3+24t^4+56t^5+130t^6+304t^7+...$$

Sequence 13

Furthermore, $2T(t,1)-T(t,0)$ is the generating function of the number of two-sided n-step prudent walks ending on the top side or right side of their box avoiding both patterns $\leftarrow^{\geq 2}$, $\downarrow^{\geq 2}$, taking steps from $\{\uparrow,\downarrow,\leftarrow,\rightarrow\}$, thus $2T(t,1)-T(t,0) =$

$$\frac{1}{(1-2t-t^2+t^3)(1-2t-2t^3)}\times$$
$$(t(1-t)^2\sqrt{(1-t-t^3)^2-4t^4}+$$
$$1-t-2t^2-2t^3-2t^4+4t^5-t^6)$$
$$= 1+4t+8t^2+20t^3+46t^4+110t^5+260t^6+616t^7 \ +...$$

Open Problem 1

What is the number of two-sided n-step prudent walks, ending on the top side of their box, avoiding both $\leftarrow^{\geq k}$, and $\downarrow^{\geq k}$ $(k>2)$ taking steps from $\{\uparrow,\downarrow,\leftarrow,\rightarrow\}$?

The generating function satisfies:

$$\left(1-t^2u\frac{1-t^ku^k}{1-tu}-\frac{tu}{u-t}\right)T(t,u)$$
$$= 1+tu\frac{1-t^ku^k}{1-tu}+\frac{u-2t}{u-t}tT(t,t),$$

where u counts the distance between the endpoint and the north-east corner of the box. For $k=3$,

$$\frac{u-t-t^2u^2+t^3u-t^3u^3+t^4u^2-t^4u^4+t^5u^3-tu}{u-t}$$
$$\times T(t,u)$$
$$= 1+tu+t^2u^2+t^3u^3+\frac{u-2t}{u-t}tT(t,t)$$

i.e.,

$$(-t+\left(1+t^3-t\right)u+\left(t^4-t^2\right)u^2+\left(t^5-t^3\right)u^3+-t^4u^4)$$
$$\times T(t,u)$$
$$= (1+tu+t^2u^2+t^3u^3)(u-t)+t(u-2t)T(t,t).$$

Set $-t+\left(1+t^3-t\right)u+\left(t^4-t^2\right)u^2+\left(t^5-t^3\right)u^3-t^4u^4 = 0$, and solve for u, as a power series of t. We obtained the first one hundred terms for u, beginning with

$$u = t+t^2+t^3+t^4+2t^5+4t^6+8t^7+16t^8+33t^9+69t^{10}+...$$

Using this u, we can get many examples for the sequence.

Open Problem 2

What is the number of two-sided n-step prudent walks, ending on the top side of their box, exactly avoiding both $\leftarrow^{=2}$, $\downarrow^{=2}$, taking steps from $\{\uparrow,\downarrow,\leftarrow,\rightarrow\}$?

The generating function is

$$(1-\frac{t^2u}{1-tu}-\frac{tu}{u-t}+u^2t^3)T(t,u)$$
$$= \frac{1}{1-tu}-u^2t^2+\frac{u-2t}{u-t}tT(t,t).$$

It seems to us it is not trivial to solve this generating function.

V. Some Theorems and Proofs

Theorem 2. *The generating function of the number, say $f(n,k)$, of the one-sided n-step prudent walks, taking steps from $\{\uparrow,\leftarrow,\rightarrow\}$, avoiding k or more consecutive east steps, $\rightarrow^{\geq k}$satisfies*

$$\frac{1+t-t^k}{1-2t-t^2+t^{k+1}},$$

and for $k\geq 2$,

$$f(n,k) = \sum_{i=0}^{n}\sum_{j=0}^{i}(-1)^{\frac{n-j-i}{k-1}}2^{i-j}\binom{i}{j}\binom{j}{\frac{n-j-i}{k-1}}$$
$$+\sum_{i=0}^{n-1}\sum_{j=0}^{i}(-1)^{\frac{n-j-i-1}{k-1}}2^{i-j}\binom{i}{j}\binom{j}{\frac{n-i-j-1}{k-1}}$$
$$-\sum_{i=0}^{n-k}\sum_{j=0}^{i}(-1)^{\frac{n-j-i-k}{k-1}}2^{i-j}\binom{i}{j}\binom{j}{\frac{n-j-i-k}{k-1}}$$

$$f(n,1) = 2^n.$$

Proof: Let $F(t)$ denote the length generating function of the number of one-sided prudent walks, avoiding k or more consecutive east steps. We have the following three cases.

(1) For the walks which do not contain North steps, they can be empty walk, walks with only west steps, walks with only east steps with length at least one and at most $k-1$, the contributions are 1, $\frac{t}{1-t}$, $\frac{t(1-t^{k-1})}{1-t}$ respectively.

(2) For the walks obtained by concatenating a one-sided walk, a North step, and then a West walk, the contribution is

$$F(t)\frac{t}{1-t}.$$

(3) For the walks obtained by concatenating a one-sided walk, a North step, and then a East walk with at least 1 step and at most $k-1$ steps, the contribution is

$$F(t)\frac{t^2(1-t^{k-1})}{1-t}.$$

Adding these three contributions give the equation

$$F(t) = 1+\frac{t}{1-t}+\frac{t(1-t^{k-1})}{1-t}$$
$$+F(t)\frac{t}{1-t}+F(t)\frac{t^2(1-t^{k-1})}{1-t}.$$

Thus,

$$F(t) = \frac{1+t-t^k}{1-2t-t^2+t^{k+1}}.$$

Now, let $[t^n]F(t)$ denote the coefficient of t^n in the power series expansion of $F(t)$.

$$\begin{aligned}
&[t^n]\frac{1+t-t^k}{1-2t-t^2+t^{k+1}} \\
&= [t^n](1+t-t^k)\sum_{i=0}^{\infty}(2t+t^2-t^{k+1})^i \\
&= [t^n](1+t-t^k)\sum_{i=0}^{\infty}\sum_{j=0}^{i}\binom{i}{j}(2t)^{i-j}(t^2-t^{k+1})^j \\
&= [t^n](1+t-t^k) \\
&\times \sum_{i=0}^{\infty}\sum_{j=0}^{i}\binom{i}{j}(2t)^{i-j}\sum_{l=0}^{j}\binom{j}{l}(t^2)^{j-l}(-1)^l t^{(k+1)l} \\
&= [t^n](1+t-t^k) \\
&\times \sum_{i=0}^{\infty}\sum_{j=0}^{i}\sum_{l=0}^{j}\binom{i}{j}\binom{j}{l}(-1)^{-l}t^{i+j-l+lk}2^{i-j} \\
&= \sum_{i=0}^{n}\sum_{j=0}^{i}(-1)^{\frac{n-j-i}{k-1}}2^{i-j}\binom{i}{j}\binom{j}{\frac{n-j-i}{k-1}} \\
&+ \sum_{i=0}^{n-1}\sum_{j=0}^{i}(-1)^{\frac{n-j-i-1}{k-1}}2^{i-j}\binom{i}{j}\binom{j}{\frac{n-i-j-1}{k-1}} \\
&- \sum_{i=0}^{n-k}\sum_{j=0}^{i}(-1)^{\frac{n-j-i-k}{k-1}}2^{i-j}\binom{i}{j}\binom{j}{\frac{n-j-i-k}{k-1}}.
\end{aligned}$$

■

Theorem 3. *The number of one-sided n-step prudent walks, starting from $(0,0)$ and ending on the y-axis, taking steps from $\{\uparrow,\leftarrow,\rightarrow\}$ is*

$$1+\sum_{k=1}^{\lfloor (n-1)/2\rfloor}\sum_{i=1}^{\min\{n-2k,k\}}\binom{n-2k+1}{i}\binom{k-1}{k-i}\binom{n-k-i}{k}.$$

Proof: In our proof, we will use the following two results which could be found in some mathematics books such as [16]:

The number of ways of putting n like objects into r different cells is

$$\binom{n+r-1}{n} = \binom{n+r-1}{r-1}.$$

It is also the number of nonnegative integer solutions to the equation

$$\sum_{i=1}^{r} x_i = n.$$

The number of ways of putting n like objects into r different cells with no empty cell is

$$\binom{n-1}{r-1}.$$

It is also the number of positive integer solutions to the equation

$$\sum_{i=1}^{r} x_i = n.$$

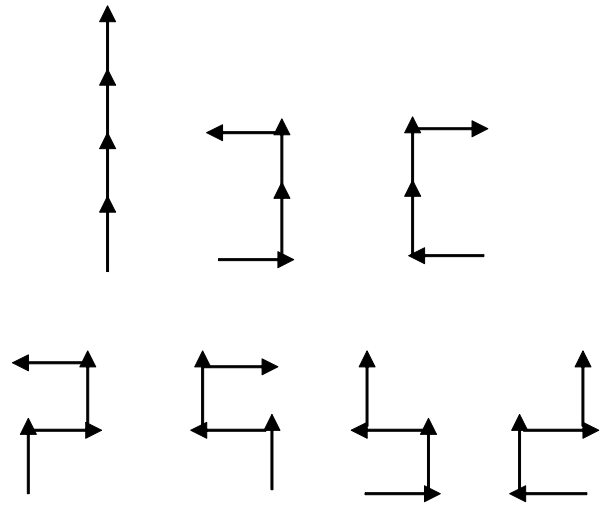
Without loss of generality, we assume that there are k East steps, k West steps and $n-2k$ North steps in a one-sided n-step prudent walks, starting from $(0,0)$ and ending on the y-axis. We also assume that $k>0$ since there is only one such walk for $k=0$. It is easy to see that $k \le \lfloor (n-1)/2 \rfloor$. The $n-2k$ North steps provide $n-2k+1$ positions (we can say $n-2k+1$ different cells) for k East steps and k West steps to be inserted. Suppose that we put k East steps into i ($1 \le i \le \min\{n-2k,k\}$) cells with no empty cell. Then there are $\binom{k-1}{k-1}$ ways of putting k East steps into i cells and $\binom{n-2k+1}{i}$ ways of choosing i cells. Now we distribute k West steps into the remaining $n-2k+1-i$ cells, which give us $\binom{n-k-i}{k}$.

Therefore, we get the number:

$$1+\sum_{k=1}^{\lfloor (n-1)/2\rfloor}\sum_{i=1}^{\min\{n-2k,k\}}\binom{n-2k+1}{i}\binom{k-1}{k-i}\binom{n-k-i}{k}.$$

■

Example: For $n=4$ in the above theorem, we have 7 such walks as follows:



Theorem 4. *The number, say $f(n)$, of generalized one-sided n-step prudent walks, taking steps from $\{\uparrow,\leftarrow,\rightarrow,\nearrow\}$ equals*

$$\begin{aligned}
&\sum_{i=0}^{n}\binom{i}{n-i}2^{n-i}(3)^{2i-n} + \sum_{i=0}^{n-1}\binom{i}{n-i-1}2^{n-i-1}(3)^{2i-n+1} \\
&= \frac{5+\sqrt{17}}{2\sqrt{17}}\left(\frac{3+\sqrt{17}}{2}\right)^n - \frac{5-\sqrt{17}}{2\sqrt{17}}\left(\frac{3-\sqrt{17}}{2}\right)^n,
\end{aligned}$$

with generating function

$$\frac{1+t}{1-3t-2t^2}.$$

Proof: Let $P(t)$ denote the length generating function of generalized one-sided prudent walks.

The contribution in $P(t)$ of walks that do not contain North steps or Northeast steps (horizontal walks) is

$$\frac{1+t}{1-t}.$$

The contribution of walks obtained by concatenating a generalized one-sided walk, a North step or Northeast step, then a horizontal walk is

$$\frac{2t(1+t)}{1-t}P(t).$$

Adding these two contributions gives a linear equation for $P(t)$:

$$P(t) = \frac{1+t}{1-t} + \frac{2t(1+t)}{1-t}P(t).$$

Therefore,

$$\begin{aligned} P(t) &= \frac{1+t}{1-3t-2t^2} \\ &= (1+t)\sum_{i=0}^{+\infty}(3t+2t^2)^i \\ &= (1+t)\sum_{i=0}^{+\infty}\binom{i}{j}(3t)^{i-j}(2t^2)^j \\ &= (1+t)\sum_{i=0}^{+\infty}\sum_{j=0}^{i}\binom{i}{j}2^j(3)^{i-j}t^{i+j} \end{aligned}$$

$$\begin{aligned} f(n) &= [t^n]P(t) \\ &= \sum_{i=0}^{n}\binom{i}{n-i}2^{n-i}(3)^{2i-n} \\ &+ \sum_{i=0}^{n-1}\binom{i}{n-i-1}2^{n-i-1}(3)^{2i-n+1}. \end{aligned}$$

The second formula of $f(n)$ can be easily derived from the length generating function. ∎

Example: For $n = 2$ in the above theorem, we have 14 such walks:

EN,NE, WN,NW,$N(NE)$,$(NE)N$,$E(NE)$,$(NE)E$, $(NE)W$, $W(NE)$, NN, WW, EE, $(NE)(NE)$.

Theorem 5. *The generating function of the number, say $f(n)$, of generalized one-sided n-step prudent walks, taking steps from $\{\rightarrow, \leftarrow, \uparrow, \nearrow, \nwarrow\}$ is*

$$\begin{aligned} &\frac{1+t}{1-4t-3t^2} \\ &= 1 + 5t + 23t^2 + 107t^3 + 497t^4 + 2309t^5 \\ &+ 10\,727t^6 + 49\,835t^7 + ... \\ f(n) &= [t^n]\,(1+t)\sum_{k\geq 0}t^k\,(4+3t)^k \\ &= [t^n]\,(1+t)\sum_{k\geq 0}\sum_{m=0}^{k}\binom{k}{m}4^{k-m}3^m t^{m+k} \\ &= \sum_{k=0}^{n}\left[\binom{k+1}{n-k}3 + \binom{k}{n-1-k}\right]4^{2k-n}3^{n-1-k}. \end{aligned}$$

Proof: Let $P(t)$ denote the length generating function of generalized one-sided prudent walks.

The contribution in $P(t)$ of walks that do not contain North steps or Northeast steps, or Northwest step (horizontal walks) is

$$\frac{1+t}{1-t}.$$

The contribution of walks obtained by concatenating a generalized one-sided walk, a North step or Northeast step or a Northwest step, then a horizontal walk is

$$\frac{3t(1+t)}{1-t}P(t).$$

Adding these two contributions gives a linear equation for $P(t)$, from which we can get $P(t)$. ∎

REFERENCES

[1] M. Neal and S. Gordon, The Self-Avoiding Walk. Birkhäuser (1996).
[2] L.F. Gregory, Intersections of Random Walks. Birkhäuser (1996).
[3] A.J. Guttmann (Ed.), Polygons, Polyominoes and Polycubes, Springer (2009).
[4] M. Bousquet-Mélou, A.J. Guttmann and I. Jensen, Self-avoiding walks crossing a square, J. Phys. A **38** (2005) 9159–9181.
[5] J. Iwan, Enumeration of self-avoiding walks on the square lattice J. Phys. A **37** (2004) 5503-5524.
[6] K.E. Donald, Science **194** (1976) 1235.
[7] E.J. Janse van Rensburg, Monte Carlo methods for the self-avoiding walk, (Topical Review) J. Phys. A **42** (2009), 323001.
[8] A.R. Conway and A.J. Guttmann, Square lattice self-avoiding walks and corrections to scaling, Phys. Rev. Lett. **77** (1996), 5284–5287.
[9] E. Duchi, On some classes of prudent walks. In Proceedings of the FPSAC'05, Taormina, Italy, (2005).
[10] P. Préa, Exterior self-avoiding walks on the square lattice, (1997) (manuscript).
[11] J.C. Dethridge and A.J. Guttmann, Prudent self-avoiding walks, Entropy **10** (2008), 309–318.
[12] T.M. Garoni, A.J. Guttmann, I. Jensen and J.C. Dethridge, Prudent walks and polygons, J. Phys. A **42** (2009), 095205.
[13] M. Bousquet-Mélou, Families of prudent self-avoiding walks, J. of Combinatorial Theory, Series A **117** (2010) 313–344.
[14] R. P. Stanley, Enumerative Combinatorics I, Wadsworth & Brooks/Cole (1986).
[15] The Online Encyclopedia of Integer Sequences (2014), published electronically at http://oeis.org.
[16] John Riordan, An Introduction to Combinatorial Analysis (originally published: New York: John Wiley 1958), Dover Publications (2002).

Model-based Automated Testing using a Record-Replay Mechanism

Junwon Kang[1], Hyunmin Yoon[2], Jaemoon Seo[1], Jungwon Roh[1], and Minsoo Ryu[1]
[1]Department of Computer Science and Engineering, Hanyang University, Seoul, Korea
[2]Department of Electronics Computer Engineering, Hanyang University, Seoul, Korea

Abstract – *Automated testing is increasingly becoming an essential part of software development as it significantly reduces laborious and time-consuming manual efforts. By automatically constructing and executing test cases with minimal manual work, automated testing promises high productivity, better coverage, and reduced cost. However, there still exist many obstacles that must be addressed to successfully incorporate automated testing into a real software development process. In this paper, we present a novel approach to automated test case creation and execution, called MoReT (Model-based Replay Testing), for event-driven systems. MoReT accepts a finite state machine model for a target system under test and generates a series of test cases based on the notion of Chow's* n*-switch coverage. Specifically, for a fixed* n*, MoReT finds an* n*-switch set cover that covers every sequence of consecutive transitions of length* n*+1 in a given state transition diagram. MoReT also allows us to automatically execute the generated test cases using a deterministic replay mechanism, called RT-Replayer, that can automatically generate and execute test cases, i.e., event sequences, during a test execution phase.*

Keywords: Model-based testing, automated testing, deterministic replay mechanism, event-driven systems, finite state machine.

1 Introduction

Automated testing is increasingly becoming an essential part of software development as it significantly reduces laborious and time-consuming manual efforts. By automatically constructing and executing test cases with minimal manual work, automated testing promises high productivity, better coverage, and reduced cost. However, there still exist many obstacles, such as lack of adequate tool support and underestimation of the automation cost, that must be addressed to successfully incorporate automated testing into a real software development process.

One of key problems is that it is hard to automatically construct effective test cases for event-driven systems. Event-driven systems usually have various sources of external events including the user and a number of I/O devices such as sensors and network interface cards. As a result, there would be a huge number of possible permutations of events that will lead to different system responses. This makes exhaustive testing infeasible, and thus requires us to determine effective test cases that can satisfy appropriate test coverage criteria.

Another important problem is that it is hard to automatically execute test cases against an event-driven system. To test an event-driven system, we should be able to precisely generate desired I/O events such as sensor data acquisitions and network data arrivals during a test execution. This requires us to build a sophisticated test environment setting so that we can precisely control the generation of various I/O events. Unfortunately, this task could be overly difficult to manage because of the diversity and complexity of event sources and complicated interdependency between events.

In this paper, we present a novel automated testing approach, called MoReT (Model-based Replay Testing), for event-driven systems. MoReT addresses the above problems by providing a model-based test case creation method and an automated test case execution technique based on deterministic execution replay. First, MoReT provides a simple, efficient model-based method for automatic test case generation. It assumes a finite state machine model for a target system under test and generates a series of test cases based on the notion of Chow's *n*-switch coverage [1]. Specifically, for a fixed *n*, MoReT finds an *n*-switch set cover that covers every sequence of consecutive transitions of length *n*+1 in a given state transition diagram. Second, MoReT provides a deterministic replay mechanism for automated test case execution. Specifically, MoReT can precisely generate and execute test cases, i.e., event sequences, during a test execution phase. MoReT executes those events automatically in pure software, completely obviating the need for human intervention and any external environment arrangement. Therefore, MoReT allows us to achieve automated test case execution for event-driven systems.

2 Automated Test Case Creation

For model-based test case creation, MoReT uses a FSM (finite state machine) model that can serve as a useful representation of an event-driven system where events govern the system's operation and dynamic behavior. For example, a sequence of events that trigger a particular execution path in a FSM model can be viewed as a meaningful test case against the target event-driven system.

Given a FSM model, MoReT automatically generates event sequences as test cases against the target system. Since the number of event sequences is often unbounded in many cases, MoReT uses the Chow's *n*-switch coverage to guide the test case selection. For a fixed *n*, an *n*-switch cover set consists of every sequence of consecutive transitions of length *n*+1 in the FSM diagram. For example, a *0*-switch cover set consists of every possible state transition and a *1*-switch cover set consists of every sequence of two consecutive transitions. Thus, we can easily control the test coverage by specifying the value of *n*.

MoReT first generates the *0*-switch cover set by transforming a given FSM model into th □ e *0*-switch cover tree that contains every possible state transition of the FSM. Figure 1 shows an example FSM model and its *0*-switch cover tree. In this example, the *0*-switch cover set is the set of all the paths from the root node to leaf nodes.

Based on the *0*-switch cover tree, MoReT can generate the *n*-switch coverage set. As mentioned earlier, an "*n*-switch" is a sequence of consecutive branches of length *n* + 1. Thus, the *n*-switch cover set is the set of all the paths from the root node to leaf nodes in the *n*-switch cover tree. Fortunately, we can obtain an *n*-switch cover tree by adding one more possible transition to every leaf node of (*n-1)*-switch. Thus, starting from the *0*-switch cover tree, we can find the *n*-switch cover tree via iterative tree expansion.

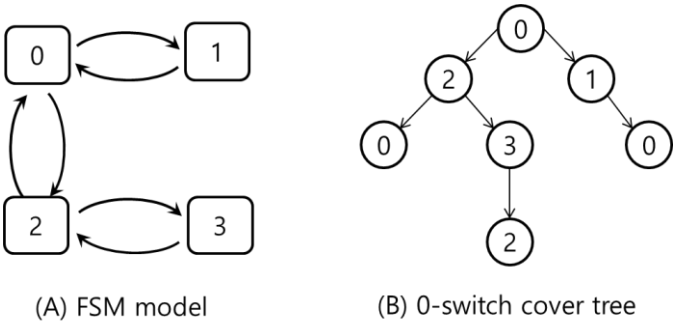


Figure 1. FSM model and its *0*-switch cover tree.

3 Automated Test Case Execution

Once test cases, i.e., event sequences, have been obtained, MoReT allows us to automatically execute the test cases using its replay mechanism, called RT-Replayer. RT-Replayer is a software mechanism that provides two key functions, event logging and event replay. For event logging, RT-Replayer monitors every I/O event and records them in an event log. When the kernel terminates, RT-Replayer stores the event log in safe non-volatile storage. For event recording, RT-Replayer can precisely emulate the recorded events consulting the event log during a re-execution.

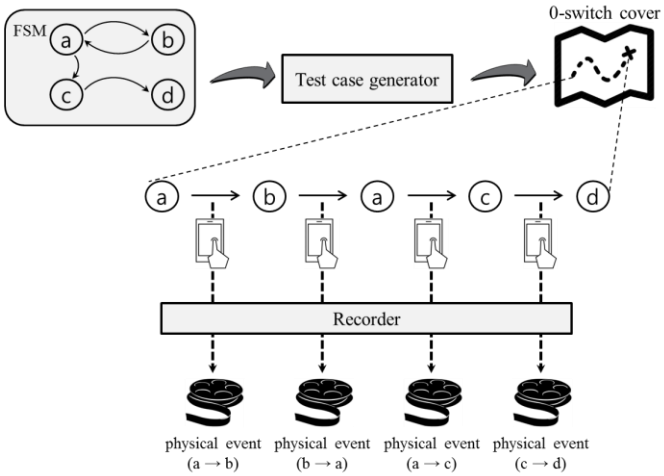


Figure 2. Manual test recording with a *0*-switch cover set.

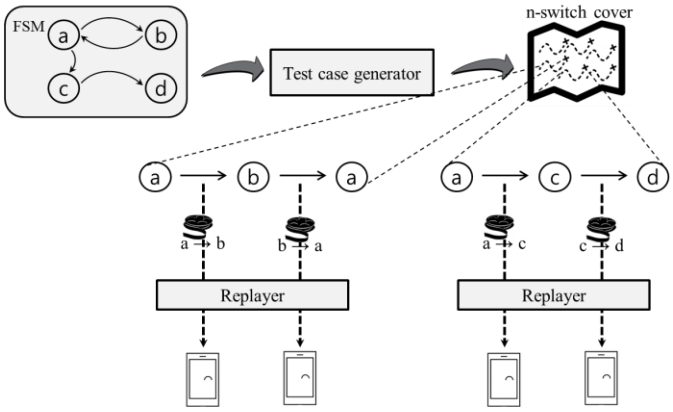


Figure 3. Automated test execution with an *n*-switch cover set.

Note that since a FSM model describes the target system at a higher level of abstraction, the event sequences derived from the logical FSM model cannot be directly executed against the physical target system due to the difference of abstraction levels between the FSM model and the physical system.

MoReT addresses this problem by introducing a two-step test execution process. The first step is manual test recording. In this step, we manually test the target system traversing every possible state transition, i.e., the *0*-switch cover set, in the given FSM. In doing so, we use the event logging mechanism of RT-Replayer to record physical events and match them with the corresponding abstract events. The

second step is automated test execution. In this step, we can automatically execute the *n*-switch cover set using the event replay mechanism of RT-Replayer. RT-Replayer can automatically emulate all physical I/O events in pure software, completely obviating the need for human intervention and any external environment arrangement.

4 Conclusion

We have presented a novel approach to automated test case creation and execution, called MoReT (Model-based Replay Testing), for event-driven systems. The contributions of this work are two-fold. First, MoReT allows us to automatically create test cases from a FSM model for a target event-driven system. Second, MoReT provides a deterministic replay mechanism for automated test case execution.

5 Acknowledgment

This work was supported partly by Seoul Creative Human Development Program (HM120006), partly by Mid-career Researcher Program through NRF (National Research Foundation) grant funded by the MEST (Ministry of Education, Science and Technology) (NRF-2011-0015997), partly by the IT R&D Program of MKE/KEIT [10035708, "The Development of CPS (Cyber-Physical Systems) Core Technologies for High Confidential Autonomic Control Software"], and partly by the MSIP(Ministry of Science, ICT&Future Planning), Korea, under the CITRC (Convergence Information Technology Research Center) support program (NIPA-2013-H0401-13-1008) supervised by the NIPA (National IT Industry Promotion Agency).

6 References

[1] T. S. Chow, "Testing software design modeled by finite-state machines," *IEEE Transactions on software engineering*, vol. SE-4, no. 3, pp. 178-187, May 1978.

[2] J. Kim, H. Yoon, and M. Ryu, "Test case generation and execution based on record-replayer Mechanism," *The 2013 international conference on foundations of computer science*, July 2013.

[3] D. Geels, G. Altekar, S. Shenker, and I. Stoica, "Replay debugging for distributed applications," *Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, pp. 27-27, May 30-June 03, 2006.

[4] T. J. LeBlanc, J. M. Mellor-Crummey, "Debugging parallel programs with instant replay," *IEEE Transactions on Computers*, v.36 no.4, pp. 471-482, pAr 1987.

[5] S. Narayanasamy, G. Pokam, B. Calder, "BugNet: Continuously Recording Program Execution for Deterministic Replay Debugging," *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, pp. 284-295, Jun 2005.

[6] E. N. (Mootaz) Elnozahy , L. Alvisi , Y. Wang , D. B. Johnson, "A survey of rollback-recovery protocols in message-passing systems," *ACM Computing Surveys (CSUR)*, v.34 no.3, pp. 375-408, Sep 2002.

[7] D. J. Sorin, M. M. K. Martin, M. D. Hill, D. A. Wood, "SafetyNet: improving the availability of shared memory multiprocessors with global checkpoint/recovery," *Proceedings of the 29th annual international symposium on Computer architecture*, pp. 123, May 2002

[8] G. W. Dunlap, S. T. King, S. Cinar, M. A. Basrai, P. M. Chen, "ReVirt: enabling intrusion analysis through virtual-machine logging and replay," *ACM SIGOPS Operating Systems Review*, v.36 no.SI, winter 2002.

[9] J. Chow, T. Garfinkel, P. M. Chen, "Decoupling dynamic program analysis from execution in virtual environments," *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pp. 1-14, Jun 2008.

[10] M. Grötschel and Y. Yuan, "Euler, Mei-Ko Kwan, Königsberg, and a Chinese Postman," *2010 Mathematics Subject Classification 00-02*, 01A05, 05C38, 90-03, pp. 43-50.

[11] Z. C. Hlaing and M. A. Khine, Member, IACSIT, "Solving Traveling Salesman Problem by Using Improved Ant Colony Optimization Algorithm," *international journal of information and education technology*, Vol. 1, No. 5, pp. 404-409, Dec. 2011.

Counting Lattice Paths and Walks with Several Step Vectors

Shanzhen Gao, Keh-Hsun Chen
Department of Computer Science, College of Computing and Informatics
University of North Carolina at Charlotte, Charlotte, NC 28223, USA
Email: sgao3@uncc.edu, chen@uncc.edu

***Abstract*—Many famous researchers in computer science, mathematics and other areas have studied enumerative problems in lattice path and walks which could be applied to many fields. We will discuss some new enumerative problems including some pattern avoidance problems in lattice paths and walks with several step vectors. Results on stretches and turns are presented and several open problems are posted. A few approaches are used in this paper such as computational, generating function, closed formula and constructional method. You will observe many interesting integer sequences as well.**

Keywords: Lattice path, prudent self-avoiding walk, generating function, pattern avoidance, stretch

I. Introduction, Notations and Preliminaries

In order to present our problems and results clearly and efficiently, we introduce some notations in the following.

East step: E or $\rightarrow$ or $(1,0)$, x-step

You can see more in the table below:

$(0,1)$	$(1,0)$	$(1,1)$	$(0,-1)$
$\uparrow$	$\rightarrow$	$\nearrow$	$\downarrow$
N	E	NE	S

$(-1,0)$	$(-1,-1)$	$(-1,1)$	$(1,-1)$
$\leftarrow$	$\swarrow$	$\nwarrow$	$\searrow$
W	SW	NW	SE

$\uparrow^{\geq k}$: k or more than k consecutive $\uparrow$ steps

$\uparrow^{=k}$: k consecutive $\uparrow$ steps

avoiding $\uparrow^{\geq k}$: no k or more than k consecutive $\uparrow$ steps

avoiding $\uparrow^{=k}$: no k consecutive $\uparrow$ steps, but can have more than or less than k consecutive $\uparrow$ steps

$\lfloor x \rfloor$: the largest integer not greater than x, $floor(x)$

$\lceil x \rceil$: is the smallest integer not less than x, $ceiling(x)$

$[x^n]f(x)$ denotes the coefficient of x^n in the power series expansion of a function $f(x)$.

$[x^m y^n]f(x,y)$ denotes the coefficient of $x^m y^n$ in the power series expansion of a function $f(x,y)$.

$\binom{n}{r}$, the number of combinations of n things r at a time.

$$\binom{n}{r} = \frac{n!}{(n-r)!r!} = \binom{n}{n-r} = \binom{n-1}{r-1} + \binom{n-1}{r}$$

$$\binom{-n}{r} = (-1)^r \binom{n+r-1}{r}$$

A lattice path is a path from the lattice point (x_1, y_1) to the lattice point (x_2, y_2), $x_1 \leq x_2$, $y_1 \leq y_2$, we mean a directed path from (x_1, y_1) to (x_2, y_2) which passes through lattice points with movements parallel to the positive direction of either axis. Here, we refer to two types steps, viz., $x-$steps and $y-$steps, where an x (y)$-$step is a directed line segment parallel to the x (y) axis going right (up) joining two neighboring points. For counting purposes we may, without loss of generality, consider lattice paths from the origin to (m,n) and observe that each such path is characterized by having exactly m horizontal steps and n vertical steps. If we denote by $f(m,n)$ the number of paths from $(0,0)$ to (m,n), elementary reasoning gives the results

$$f(m,n) = \binom{m+n}{n}.$$

Lattice paths are encountered in a natural way in various problems, e.g., ballot problems, compositions, random walks, fluctuations, queues, and the tennis ball problem.

The number of lattice paths from the origin to (m,n), $m > n+t$, not touching the line $x = y + t$, where t is a nonzero integer satisfies [18]

$$\binom{m+n}{n} - \binom{m+n}{m-t}.$$

When $t = 0$, the paths have to touch the line $x = y$ at the origin, and therefore the number paths from the origin to (m,n) that do not touch the line $x = y$ except at the origin is given by

$$\frac{m-n}{m+n}\binom{m+n}{n}.$$

The number of lattice paths from (r,s) to (m,n) that never rise above the line $y = x$ is [1]

$$\binom{n+m-r-s}{m-r} - \binom{n+m-r-s}{m-s+1}.$$

Then the number of lattice paths from $(0,0)$ to (m,n) that never rise above the line $y = x$ is

$$\binom{n+m}{m} - \binom{n+m}{m+1}.$$

The of n-step lattice paths starting from $(0,0)$ that never rise above the line $y = x$ is

$$\sum_{i=\lceil n/2 \rceil}^{n} \frac{n!(2i+1-n)}{(i+1)!(n-i)!} = \binom{n}{\lfloor n/2 \rfloor}.$$

The number of paths from $(0,0)$ to (n,n) that never rise above the line $y = x$ is the nth Catalan number, denoted by

C_n, and define $C_0 = 1$.

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \sum_{i=0}^{n}\binom{n}{i}^2$$

with generating function

$$\frac{1-\sqrt{1-4x}}{2x}.$$

Also,

$$C_{n+1} = \sum_{i=0}^{n} C_i C_{n-1} = \frac{2(2n+1)}{n+2} C_n.$$

The tennis ball problem was presented on pages $304-305$ of the book " Sweet Reason: A Field Guide to Modern Logic" by Tom Tymoczko and Jim Henle in 1995. Their presentation deals with adding numbered books to a stack on a table, then removing some, infinitely many times. Motivated by that presentation, Ralph P. Grimaldi and Joseph G. Moser deal with performing the process a finite number of times. Since then more mathematicians have studied the problem, such as C. L. Mallows and L. Shapiro, R.J. Chapman, T.Y. Chow, A. Khetan, D.P. Moulton, R.J. Waters, J. E. Bonin, Anna de Mier, M. Noy, H. Niederhausen, J. Fallon, and S. Gao. [2], [4], [5], [6], [7], [8], [9], [11], [13], [14], [16], [17] However, it is still wildly open and might challenge more people in the future.

The number of ways of putting n like objects into r different cells is $\binom{n+r-1}{n} = \binom{n+r-1}{r-1}$. [19] It is also the number of nonnegative integer solutions to the equation $\sum_{i=1}^{r} x_i = n$. The number of ways of putting n like objects into r different cells with no cell is empty is $\binom{n-1}{r-1}$. It is also the the number of positive integer solutions to the equation $\sum_{i=1}^{r} x_i = n$.

If p is a prime, then $\binom{p}{i}$ is divisible by p for $1 \le i \le p-1$. [21]

Fibonacci number: F_n is defined as $F_0 = 0$, $F_1 = 1$, $F_n = F_{n-2} + F_{n-1}$ for $n \ge 2$.

II. Pattern Avoidance in Lattice Paths and Walks

The number of n-step walks with steps $(0,1)$, $(1,0)$ and $(-1,-1)$ is

$$\frac{(3n)!}{(n!)^3}.$$

Theorem 1. *The number of $3n$-step walks from $(0,0)$ to $(0,0)$, taking steps from {E,N, SW}, and staying above the line $y = x$ (i.e., any point (x,y) along the path satisfies $y \ge x$) is given by*

$$\frac{(3n)!}{(n!)^2(n+1)!}.$$

Example: $n = 1$, three walks: $NE(SW)$, $(SW)NE$, $N(SW)E$.

This is the sequence [22, A007004]:

$$1, 3, 30, 420, 6930, 126126, ...$$

Proof: It is clear that such a $3n$-step walk contains n copies of north, east and southwest step, respectively.
It is also true that the total number of north and east steps is greater or equal to the number of southwest steps at any lattice point on a walk. Now we arrange n north and east steps (total is n) with n southwest steps to get a $2n$-step walk according to: the total number of the chosen steps is greater or equal to the number of southwest steps at any lattice point on a walk, which gives C_n (We do not consider the difference of north steps and east steps at this moment). Next, we have $2n+1$ positions to insert the remaining n steps of the north steps and east steps into the $2n$-step walk, giving $\binom{3n}{n}$ ways. Now combine them:

$$C_n\binom{3n}{n} = \frac{(3n)!}{(n!)^2(n+1)!}.$$

■

This theorem also could be proved by using *André's Reflection Method*:

$$\binom{3n}{n,n,n} - \binom{3n}{n+1,n-1,n} = \frac{(3n)!}{(n!)^2(n+1)!}.$$

Theorem 2. *The number of $3n$-step walks from $(0,0)$ to $(0,0)$, taking steps from {W,S,NE}, and staying within the first quadrant (i.e., any point (x,y) along the walk satisfies $x, y \ge 0$) is given by [12]*

$$\frac{4^n(3n)!}{(n+1)!(2n+1)!}.$$

Example: $n = 1$, two walks: $(NE)SW$, $(NE)WS$.
This is the sequence [22, A006335]:

$$1, 2, 16, 192, 2816, 46592, 835584, ...$$

Theorem 3. *The number of lattice paths avoiding $\uparrow^{\ge 2}$, from $(0,0)$ to (m,n) is*

$$\binom{m+1}{n}.$$

Proof: The m East steps provide $m+1$ positions (we can say $m+1$ different cells) for n North steps to be inserted with each cell containing at most one element (North step). Then there are $\binom{m+1}{n}$ ways to choose n cells. ■

Corollary 4. *The number of lattice paths from $(0,0)$ to $(ns+1, nt-1)$, avoiding $\uparrow^{\ge 2}$is*

$$\binom{ns+2}{nt-1}.$$

Corollary 5. *The number of n-step paths with east and north steps and with two consecutive north steps forbidden is equal to [3]*

$$\sum_{i=0}^{\lceil n/2 \rceil}\binom{n+1-i}{i} = F_{n+2}.$$

Theorem 6. *The number of walks from* $(0,0)$ *to* (m,n) *(* $m \geq n$*) take steps from {E,N,NE} is*

$$\sum_{k=0}^{n} \binom{m+n-2k}{n-k} \binom{m+n-k}{k}.$$

Proof: Without loss of generality, we assume that there are k Northwest steps, $m-k$ East steps and $n-k$ North steps in a walk from $(0,0)$ to (m,n). It is clear that $0 \leq k \leq n$.

Firstly, we only consider the number of arrangements of $m-k$ East steps and $n-k$ North steps, which give us $\binom{m+n-2k}{n-k}$ ways.

Secondly, $m-k$ East steps and $n-k$ North steps provide $m+n-2k+1$ positions (we can say $m+n-2k+1$ different cells) for k Northwest steps to be inserted, which give $\binom{m+n-k}{k}$.

Therefore, we get the number:

$$\sum_{k=0}^{n} \binom{m+n-2k}{n-k} \binom{m+n-k}{k}.$$

■

We obtain sequence $[22, A001850]$ for $m = n$:

$$1, 3, 13, 63, 321, 1683, 8989, 48639, ...$$

Sequence $[22, A002002]$ for $m = n+1$:

$$1, 5, 25, 129, 681, 3653, 19825, ...$$

The number of walks from $(0,0)$ to $(n, n-1)$ ($m \geq n$) take steps from {E,N,NE}.

Sequence $[22, A026002]$ for $m = n+2$:

$$1, 7, 41, 231, 1289, 7183, 40081, ...$$

Sequence $[22, A190666]$ for $m = n+3$:

$$9, 61, 377, 2241, 13073, 75517, 433905, ...$$

Example 7. *There are* 13 *walks in the above theorem for* $m = n = 2$*:* 6 *walks with* 2 *East steps and* 2 *North steps,* 1 *walk with two Northeast steps,* 6 *walks with* 1 *Northeast step,* 1 *East step and* 1 *North step: (NE)NE, (NE)EN, NE(NE), EN(NE),E(NE)N, N(NE)E.*

Corollary 8. *The number of walks from* $(0,0)$ *to* (n,n) *take steps from {E,N,NE} is*

$$\sum_{k=0}^{n} \frac{(n+k)!}{(n-k)!(k!)^2}.$$

Theorem 9. *The number of lattice paths from* $(0,0)$ *to* (m,n) *avoiding* $\uparrow^{\geq 3}$*is*

$$\sum_{i=0}^{\lfloor n/2 \rfloor} \binom{m+1}{n-i} \binom{n-i}{i}.$$

Proof: Without loss of generality, we assume that there are i copies of double North Steps, $n-2i$ copies of single North step in a lattice path from $(0,0)$ to (m,n) and avoiding $\uparrow^{\geq 3}$. It is clear that $0 \leq i \leq \lfloor n/2 \rfloor$. The m East steps provide $m+1$ positions (we can say $m+1$ different cells) for n North steps to be inserted with each cell containing at most one element ($\uparrow$ or $\uparrow^2$). There are $\binom{m+1}{n-i}$ ways to choose n cells for the i copies $\uparrow^2$ and $n-2i$ copies of $\uparrow$. We have $\binom{n-i}{i}$ ways to distribute the i copies $\uparrow^2$. Therefore, we can get the number. ■

Corollary 10. *The number of lattice paths from* $(0,0)$ *to* $(ns+1, nt-1)$ *avoiding* $\uparrow^{\geq 3}$*is*

$$\sum_{i=0}^{\lfloor (nt-1)/2 \rfloor} \binom{ns+2}{nt-1-i} \binom{nt-1-i}{i}.$$

Theorem 11. *The number of lattice paths from* $(0,0)$ *to* (m,n) *avoiding* $\uparrow^{\geq 2}$ *and* $\rightarrow^{\geq 3}$*is*

$$\binom{n+1}{m-n-1} + 2\binom{n}{m-n} + \binom{n-1}{m-n+1}.$$

Theorem 12. *The number of lattice paths from* $(0,0)$ *to* (m,n) *avoiding* $\uparrow^{\geq 3}$*and* $\rightarrow^{\geq 3}$*is*

$$\begin{aligned} &2\sum_{i=m-n}^{\lfloor m/2 \rfloor} \binom{m-i}{i} \binom{m-i}{n-m+i} \\ &+ \sum_{i=m-n-1}^{\lfloor m/2 \rfloor} \binom{m-i}{i} \binom{m-i-1}{n-m+i+1} \\ &+ \sum_{i=m-n+1}^{\lfloor m/2 \rfloor} \binom{m-i}{i} \binom{m-i+1}{n-m+i-1}. \end{aligned}$$

The generating function of the above numbers is

$$\begin{aligned} &[x^m y^n]\big(\left(1+x+x^2+x^3\right)\left(1+y+y^2+y^3\right) \\ &(1-xy-xy^2-xy^3-x^2y-x^2y^2 \\ &-x^2y^3-x^3y-x^3y^2-x^3y^3)^{-1}\big). \end{aligned}$$

Corollary 13. *The number of lattice paths from* $(0,0)$ *to* (n,n) *avoiding* $\uparrow^{\geq 3}$*and* $\rightarrow^{\geq 3}$*is*

$$2\sum_{i=0}^{\lfloor n/2 \rfloor} \binom{n-i}{i} \binom{n-i}{i} + 2\sum_{j=1}^{\lfloor \frac{n}{2} \rfloor} \binom{n-j}{j} \binom{n-j+1}{j-1}.$$

Theorem 14. *The generating function for the number of lattice paths from* $(0,0)$ *to* (n,n) *avoiding* $\uparrow^{\geq 3}$*and* $\rightarrow^{\geq 3}$*, is*

$$\begin{aligned} &\frac{(1-t)^2\sqrt{(1+t+t^2)(1-3t+t^2)} - (1-3t+t^2)(1+t^2)}{t^2(1-3t+t^2)} \\ &= 2t + 6t^2 + 14t^3 + 34t^4 + 84t^5 + 208t^6 + 518t^7 + \ldots . \end{aligned}$$

A proof of this theorem involved finite operator calculus is in [10].

Example 15. *For* $m \leq 7$ *and* $n \leq 8$*, the number of lattice paths from* $(0,0)$ *to* (m,n) *avoiding* $\uparrow^{\geq 3}$*and* $\rightarrow^{\geq 3}$*is*

as follows:

n=8				1	15	87
n=7				4	30	114
n=6			1	10	43	113
n=5			3	16	45	**84**
n=4		1	6	18	**34**	45
n=3		2	7	**14**	18	16
n=2	1	3	**6**	7	6	2
n=1	1	**2**	3	2	1	
n=0		1	1			
	m=0	m=1	m=2	m=3	m=4	m=5

from $(0,0)$ *to* (m,n) *avoiding* $\uparrow^{\geq 3}$ *and* $\longrightarrow^{\geq 3}$

Theorem 16. *Let* $f(m,n)$ *be the number of lattice paths from* $(0,0)$ *to* (n,n) *avoiding* $\uparrow^k$ *and* $\longrightarrow^k$*,taking steps from* $\{\uparrow, \longrightarrow\}$*. Then*

$$f(m,n) = f(m-1,n) + f(m,n-1) - f(m-k,n-1) - f(m-1,n-k) + f(m-k,n-k).$$

Corollary 17. *The number of lattice paths from* $(0,0)$ *to* $((ns+1), nt-1)$ *avoiding* $\uparrow^{\geq 3}$*and* $\rightarrow^{\geq 3}$*is*

$$2\sum_{i=(ns+1)-n}^{\lfloor (ns+1)/2 \rfloor} \binom{m-i}{i}\binom{m-i}{n-m+i} + \sum_{i=m-n-1}^{\lfloor m/2 \rfloor} \binom{m-i}{i}\binom{m-i-1}{n-m+i+1} + \sum_{i=m-n+1}^{\lfloor m/2 \rfloor} \binom{m-i}{i}\binom{m-i+1}{n-m+i-1}.$$

Theorem 18. *The number of lattice path from* $(0,0)$ *to* (n,n) *avoiding* $\uparrow^{\geq 4}, \rightarrow^{\geq 4}$*is*

$$2\sum_{i=0}^{\lfloor n/3 \rfloor}\sum_{j=0}^{\lfloor (n-3i)/2 \rfloor} \binom{n-2i-j}{i}\binom{n-3i-j}{j} \sum_{s=0}^{\min\{\lfloor n/3 \rfloor, \lfloor \frac{2i+j}{2} \rfloor\}} \binom{n-2i-j}{s}\binom{n-2i-j-s}{2i+j-2s} + 2\sum_{i=0}^{\lfloor n/3 \rfloor}\sum_{j=0}^{\lfloor (n-3i)/2 \rfloor} \binom{n-2i-j}{i}\binom{n-3i-j}{j} \sum_{s=0}^{\min\{\lfloor n/3 \rfloor, \lfloor \frac{2i+j+1}{2} \rfloor\}} \binom{n-2i-j-1}{s}\binom{n-2i-j-s-1}{2i+j+1-2s}.$$

The above numbers equal

$$[x^n y^n]\left(\frac{\left(1+x+x^2+x^3\right)\left(1+y+y^2+y^3\right)}{1-xy\left(1+y+y^2\right)\left(1+x^2+x\right)}\right).$$

Theorem 19. *The generating function of the number of lattice paths from* $(0,0)$ *to* (n,n) *avoiding* $\uparrow^{\geq i}, \rightarrow^{\geq j}$*satisfies*

$$\frac{(\sum_{k=1}^{i} x^{k-1})(\sum_{k=1}^{j} y^{k-1})}{1-(\sum_{k=2}^{i} x^{k-1})(\sum_{k=2}^{j} y^{k-1})}.$$

Problem 20. *How to find the number of lattice paths from* $(0,0)$ *to* (n,n) *avoiding* $\uparrow^{\geq i}, \rightarrow^{\geq j}$*, and weakly above the the diagonal* $y = x$ *. And how to find a good generating function for this problem?*

III. Stretches and Turns [15]

A. Lattice Path with East, North Steps:

Consider the paths from $(0,0)$ to (m,n) with s level-stretches (a stretch is one or some unextendable continues level steps),k right turns and h up-stretches.

Let $f_1(m,n,k)$ denote number of walks from $(0,0)$ to (m,n) with k right turns, then

$$f_1(m,n,k) = \binom{m}{k}\binom{n}{k}.$$

Example: $f_1(1,1,1) = 1$, $f_1(1,1,0) = 1$, $f_1(2,2,1) = 4$.

Let $f_2(m,n,s)$ denote the number of walks from $(0,0)$ to (m,n) with s level-stretches, then

$$f_2(m,n,s) = \binom{m-1}{s-1}\binom{n+1}{s}.$$

Example: $f_2(1,1,1) = 2, f_2(1,2,1) = 3, f_2(3,2,2)$=6.

Let $f_3(m,n,h)$ denote the number of walks from $(0,0)$ to (m,n) with h up-stretches, then

$$f_3(m,n,h) = \binom{m+1}{h}\binom{n+1}{h-1}.$$

Example: $f_3(1,1,1) = 2, f_3(1,2,2) = 3, f_3(3,3,2) = 24$.

Let $f_4(m,n,t)$ denote the number of walks from $(0,0)$ to (m,n) with t stretches, then $f_4(m,n,t) =$

$$2\binom{m-1}{t/2-1}\binom{n-1}{t/2-1} \text{ when } t \text{ is even,}$$

$$\binom{m-1}{(t-1)/2}\binom{n-1}{(t-1)/2-1} + \binom{m-1}{(t+1)/2-2}\binom{n-1}{(t+1)/2-1}$$

when t is odd.

Example:$f_4(1,1,2) = 2, f_4(3,3,4) = 8, f_4(2,1,3) = 1, f_4(3,4,5) = 9$.

Let $f_5(m,n)$ denote the number of walks from $(0,0)$ to (m,n). It is well known that

$$f_5(m,n) = \binom{m+n}{n} = \binom{m+n}{m}.$$

We also have

$$f_5(m,n)=\sum_{i\geq 1}\binom{m+1}{i}\binom{n-1}{i-1}.$$

Example: $f_5(1,1)=2, f_5(1,2)=3, f_5(2,2)=6$.

B. ENW Walks

Counting walks which start at the origin $(0,0)$ and take unit steps $(1,0)$, $(0,1)$, and $(-1,0)$ with the restriction that no E step immediately follows a W step and vice verse. The restriction has the effect of making the walks self-avoiding. It is a major unsolved problem to enumerate all self-avoiding walks. We start by counting walks which start at the origin $(0,0)$ and take unit steps $(1,0)$,and $(0,1)$. Let $p(m,n)$ denote the number of ENW walks from $(0,0)$ the (m,n). We have

$$p(m,n)=p(m,n-1)+2\sum_{i>0}p(m-j,n-1)$$
$$p(m+n)=\sum_{i=0}p(m+i,n-i).$$

Let $p_1(m,n,h)$ denote the number of walks from $(0,0)$ to (m,n) with h up-stretches, then

$$p_1(m,n,h)=2^{h+1}\binom{m}{h}\binom{n-1}{h-1}+2^{h-1}\binom{m-1}{h-2}\binom{n-1}{h-1}.$$

Example:$p_1(1,1,1)=4, p_1(3,4,2)=78$.

Let $p_2(m,n,t)$ denote the number of walks from $(0,0)$ to (m,n) with t stretches, then

$$p_2(m,n,t)=$$

$$2^{(t/2+1)}\binom{m-1}{t/2-1}\binom{n-1}{t/2-1}\text{ when } t \text{ is even;}$$

$$2^{(t+1)/2}\binom{m-1}{(t-1)/2}\binom{n-1}{(t-1)/2-1}+$$
$$2^{(t-1)/2}\binom{m-1}{(t+1)/2-2}\binom{n-1}{(t+1)/2-1}$$
when t is odd.

Example:$p_2(1,1,2)=4, p_2(4,3,4)=48, p_2(1,2,3)=2$,$p_2(3,4,5)=48$.

Let $p_2(N,t)$ denote the number of walks with length N and t stretches, then $p_2(N,t)=$

$$\sum_{n=t/2}^{N-t/2}2^{t/2+1}\binom{N-n+1}{t/2-1}\binom{n}{t/2-1}\text{ for even } t$$
$$\sum_{n=(t-1)/2}^{N-(t+1)/2}2^{(t+1)/2}\binom{N-n-1}{(t-1)/2}\binom{n-1}{(t-1)/2-1}$$
$$+\sum_{n=(t+1)/2}^{N-(t-1)/2}2^{(t-1)/2}\binom{N-n-1}{(t+1)/2-2}\binom{n-1}{(t+1)/2-1}$$
for odd t.

Example: $p_2(2,2)=4, p_2(3,2)=8$, $p_2(1,1)=3$, $p_2(3,3)=6$.

Let $p(N)$ be the number of walks of length N, then

$$p(N)=3+\sum_{t=1}^{N}p_2(N,t).$$

C. ENW Walks without Ending With a W Step

We now consider ENW walks from $(0,0)$ to (m,n) with the additional restriction no walk ends with a W step. Let $q_1(m,n,h)$ denote the number of walks from $(0,0)$ to (m,n) with h up-stretches, then

$$q_1(m,n,h)=2^{h-1}\binom{n-1}{h-1}(\binom{m}{h-1}+2\binom{m}{h}).$$

Example:$q_1(1,1,1)=3, q_1(2,2,1)=5$,$q_1(3,4,2)=54$.

Let $q_2(m,n,t)$ denote the number of walks from $(0,0)$ to (m,n) with t stretches, then $q_2(m,n,t)=$

$$3\times 2^{t/2-1}\binom{m-1}{t/2-1}\binom{n-1}{t/2-1}\text{ for even } t$$
$$2^{(t-1)/2}\binom{m-1}{(t+1)/2-2}\binom{n-1}{(t+1)/2-1}$$
$$+2^{(t-1)/2}\binom{m-1}{(t-1)/2}\binom{n-1}{(t-1)/2-1}\text{ for odd } t.$$

Example:$q_2(1,1,2)=3, q_2(4,3,4)=36, q_2(1,2,3)=2$,$q_2(3,4,5)=36$.

Let $q_2(N,t)$ denote the number of walks with length N and t stretches, then $q_2(N,t)=$

$$\sum_{n=t/2}^{N-t/2}3\times 2^{t/2-1}\binom{N-m-1}{t/2-1}\binom{n-1}{t/2-1}\text{ for even } t$$

$$\sum_{n=(t-1)/2}^{N-(t-1)/2}(2^{(t-1)/2}\binom{N-n-1}{(t+1)/2-2}\binom{n-1}{(t+1)/2-1}$$
$$+2^{(t-1)/2}\binom{N-n-1}{(t-1)/2}\binom{n-1}{(t-1)/2-1})$$
for odd t.

Example: $q_2(2,2)=3$,$q_2(3,2)=6, q_2(3,3)=4, q_2(5,3)=24$.

Let $q(N)$ be the number of walks of length N, then

$$q(N)=\sum_{t=1}^{N}p_2(N,t).$$

Let $q(m,n)$ denote the number of walks from $(0,0)$ to (m,n), then

$$q(m,n)=\sum_{h=1}^{n}2^{h-1}\binom{n-1}{h-1}(\binom{m}{h-1}+2\binom{m}{h}).$$

Example:$q(1,1)=3, q(1,2)=5, q(3,4)=129$.

$$q(N)=\sum_{n=1}^{N}(\sum_{h=1}^{n}2^{h-1}\binom{n-1}{h-1}(\binom{N-n}{h-1}+2\binom{N-n}{h}))+1.$$

Example: $q(0)=1, q(1)=2, q(2)=5, q(3)=12$,$q(4)=29, q(5)=70, q(6)=169$.

D. END Walks

We now consider walks from $(0,0)$ to (m,n) and taking unit steps $(1,0)=E$(east) and $(0,1)=N$(north) and double east steps of length 2 denoted by D.

Let $c(m,n,h,d)$ denote the number of walks from $(0,0)$ to (m,n) with h up-stretches and d copies of D, then

$$c(m,n,h,d)=\binom{n-1}{h-1}\binom{m-d}{d}\binom{m-d+1}{h}.$$

Example: $c(1,1,1,0)=2, c(2,1,1,1)=2$,$c(6,6,4,2)=300, c(6,6,4,3)=10$.

Let $c(m,n,h)$ denote the number of walks from $(0,0)$ to (m,n) with h up-stretches, then

$$c(m,n,h)=\sum_{d=0}^{m}\binom{n-1}{h-1}\binom{m-d}{d}\binom{m-d+1}{h}.$$

Example: $c(1,1,1)=2, c(2,1,1)=5, c(4,3,2)=62$.

Let $c(m,n)$ denote the number of walks from $(0,0)$ to (m,n),then

$$c(m,n)=\sum_{h=0}^{n}\sum_{d=0}^{m}\binom{n-1}{h-1}\binom{m-d}{d}\binom{m-d+1}{h}.$$

Example: $c(1,1)=2, c(1,2)=3, c(3,3)=40$.

Also,

$$\begin{aligned}c(m,n)&=\sum_{h=0}^{n}\sum_{d=0}^{m}\binom{n-1}{h-1}\binom{m-d}{d}\binom{m-d+1}{h}\\&=\sum_{d=0}^{m}\binom{m-d}{d}\sum_{h=0}^{n}\binom{n-1}{h-1}\binom{m-d+1}{h}\\&=\sum_{d=0}^{m}\binom{m-d}{d}\binom{n+m-d}{m-d}.\end{aligned}$$

Let $c(N)$ denote the number of walks of length N, then

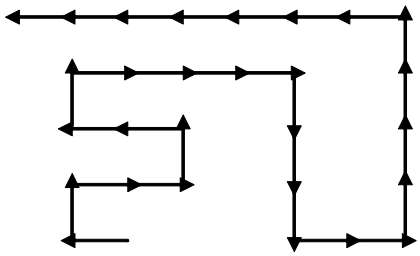
$$\begin{aligned}c(N)&=\sum_{n=0}^{N}\sum_{d=0}^{N-n}\binom{N-n-d}{d}\binom{N-d}{N-n-d}\\&=\sum_{n=0}^{N}\sum_{d=0}^{N-n}\binom{N-n-d}{d}\binom{N-d}{n}.\end{aligned}$$

Example:$c(1)=2, c(2)=5, c(3)=12, c(4)=29$.

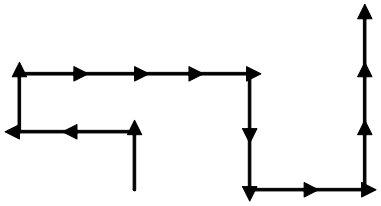
IV. Open Problems in Prudent Self-avoiding Walks

We will talk about two-sided prudent self-avoiding walks (PSAWs) in the following:

A well-known long standing problem in combinatorics and statistical mechanics is to find the generating function for self-avoiding walks (SAW) on a two-dimensional lattice, enumerated by perimeter. A SAW is a sequence of moves on a square lattice which does not visit the same point more than once. It has been considered by more than one hundred researchers in the pass one hundred years, including George Polya, Tony Guttmann, Laszlo Lovasz, Donald Knuth, Richard Stanley, Doron Zeilberger, Mireille Bousquet-Mélou, Thomas Prellberg, Neal Madras, Gordon Slade, Agnes Dittel, E.J. Janse van Rensburg, Harry Kesten, Stuart G. Whittington, Lincoln Chayes, Iwan Jensen, Arthur T. Benjamin, and others. More than three hundred papers and a few volumes of books were published in this area. A SAW is interesting for simulations because its properties cannot be calculated analytically. Calculating the number of self-avoiding walks is a common computational problem. A PSAW is a proper subset of SAWs on the square lattice. The walk starts at $(0,0)$, and the empty walk is a PSAW. A PSAW grows by adding a step to the end point of a PSAW such that the extension of this step - by any distance - never intersects the walk. Hence the name prudent. The walk is so careful to be self-avoiding that it refuses to take a single step in any direction where it can see - no matter how far away - an occupied vertex. The following walk is a PSAW.



In a *one-sided* PSAW, the endpoint lies always on the top side of the box. The walk is *partially directed*. A prudent walk is *two-sided* if its endpoint lies always on the top side, or on the right side of the box. The walk in the following figure is a two-sided PSAW.



We studied the pattern avoidance problems on one-sided and two-sided PSAWs and obtained:

(1) The generating function of the number, say $f(n,k)$, of one-sided n-step prudent walks exactly avoiding $\leftarrow^{=k}$ and $\uparrow^{=k}$

(both at the same time) satisfies

$$\frac{1+t-2t^k+2t^{k+1}}{1-2t-t^2+2t^{k+1}-2t^{k+2}}, \text{ and}$$

$$f(n,k) = g(n,k) + g(n-1,k) - 2g(n-k,k) + 2g(n-k-1,k), \text{ where}$$

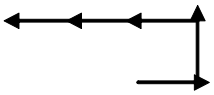
$$g(n,k) = \sum_{i=0}^{n}\sum_{j=0}^{i}\sum_{l=0}^{j}\binom{i}{j}\times \binom{j}{l}\binom{l}{-i-j+l-kl+n}(-1)^{i+j+kl-n}2^{i-j+l}.$$

(2) The generating function of the number of two-sided n-step prudent walks, ending on the top side of their box, avoiding both $\leftarrow^{\geq k}$, and $\downarrow^{\geq k}$ $(k > 2)$ taking steps from $\{\uparrow,\downarrow,\leftarrow,\rightarrow\}$satisfies:

$$\left(1-t^2u\frac{1-t^ku^k}{1-tu}-\frac{tu}{u-t}\right)T(t,u) = 1+tu\frac{1-t^ku^k}{1-tu}+\frac{u-2t}{u-t}tT(t,t)$$

where u counts the distance between the endpoint and the north-east corner of the box.

For instance, in the following figure, a walk takes 5 steps, and the distance between the endpoint and the north-east corner is 3. So we can use t^5u^3 to count this walk.



(3) The generating function of the number of two-sided n-step prudent walks, ending on the top side of their box, exactly avoiding both $\leftarrow^{=2}$, $\downarrow^{=2}$, taking steps from $\{\uparrow,\downarrow,\leftarrow,\rightarrow\}$, equals

$$(1-\frac{t^2u}{1-tu}-\frac{tu}{u-t}+u^2t^3)T(t,u) = \frac{1}{1-tu}-u^2t^2+\frac{u-2t}{u-t}tT(t,t).$$

Then we come to two open problems and two new results here:

Problem 21. *How to enumerate the number of two-sided n-step prudent walks, ending on the top side of their box, avoiding both $\leftarrow^{\geq i}$, and $\downarrow^{\geq j}$ $(i > j > 2)$ taking steps from $\{\uparrow,\downarrow,\leftarrow,\rightarrow\}$?*

Problem 22. *How to enumerate the number of two-sided n-step prudent walks, ending on the top side of their box, exactly avoiding both $\leftarrow^{=i}$, $\downarrow^{=j}(i > j > 2)$ taking steps from $\{\uparrow,\downarrow,\leftarrow,\rightarrow\}$?*

Theorem 23. *The number of one-sided n-step prudent walks in the first quadrant, starting from $(0,0)$ and ending on the y-axis, taking steps from $\{\uparrow,\leftarrow,\rightarrow\}$, avoiding $\leftarrow^{\geq 2}$ and $\rightarrow^{\geq 2}$is*

$$\sum_{j=0}^{\lfloor\frac{n+1}{4}\rfloor}\frac{1}{j+1}\binom{2j}{j}\binom{n-2j+1}{2j}.$$

Proof: We suppose that there are j copies of East steps, j copies of West steps and $n-2j$ copies of North steps.

Now we arrange j copies of East steps and j copies of West steps according to: the total number of the East steps is greater or equal to the total number of West steps from $(0,0)$ to any lattice point on a walk, which gives C_j, the jth Catalan number.

The $n-2j$ North steps in a walk provide $n-2j+1$ positions (i.e., cells) for j East steps and j West steps to be distributed, with each cell containing at most 1 East step or 1 West step. Then $0 \leq j \leq \lfloor\frac{n+1}{4}\rfloor$. There are $\binom{n-2j+1}{2j}$ way to choose $2j$ cells for the j East steps and j West steps.

Therefore,

$$\sum_{j=0}^{\lfloor\frac{n+1}{4}\rfloor}\frac{1}{j+1}\binom{2j}{j}\binom{n-2j+1}{2j}.$$

■

REFERENCES

[1] B. Bollobás, The Art of Mathematics: Coffee Time in Memphis, Cambridge University Press, 2006.

[2] J. E.Bonin, A. de Mier, and M. Noy, Lattice path matroids: Enumerative aspects and Tutte polynomials, J. Combin. Theory Ser. A **104** (2003), 63-94.

[3] L. Carlitz, Zero-one sequences and Fibonacci numbers of higher order, The Fibonacci Quarterly, Vol. **12** (1974) 1–10.

[4] R. Chapman, T. Chow, A. Khetan, D. P. Moulton, and R. J. Waters, Simple formulas for lattice paths avoiding certain periodic staircase boundaries, arXiv:0705.2888v1 (2007).

[5] A. de Mier, A natural family of flag matroids, SIAM Journal on Discrete Mathematics, **21** (2007),130-140.

[6] A. de Mier, and M. Noy, A solution to the tennis ball problem, Theoret. Comput. Sci. **346** (2005), 254-264.

[7] J. Fallon, S. Gao, and H. Niederhausen, A finite operator approach to the tennis ball problem, Congr. Numer. **184** (2007)**,** 5-10.

[8] J. Fallon, S. Gao, and H. Niederhausen, Proof of a lattice paths conjecture connected to the tennis ball problem, Journal of Statistical Planning and Inference, **140** (2010), 2227–2229.

[9] R. Grimaldi, and J. Moser, The Catalan numbers and a tennis ball problem, Congr. Numer. **125** (1997), 65-71.

[10] S. Gao, Using finite operator calculus solving pattern avoidance problems in lattice paths and walks (in preparation)

[11] M. Jani, and M. Zeleke, A bijective proof of a tennis ball problem, Bulletin of the ICA, **41 (2004)**, 89-95.

[12] G. Kreweras, Sur une class de problèmes de dénombrement liés au treillis des partitions d'entiers." Cahiers Buro 6, 2-107(1965).

[13] C. Mallows, and L. Shapiro, Balls on the lawn, J. Integer Seq. **2** (1999).

[14] D. Merlini, R. Sprugnoli, and M.C. Verri, The tennis ball problem, J. Combin. Theory Ser. A **99** (2002), 307-344.

[15] A. Nkwanta, and L. Shapiro, Pell walks and Riordan matrices,The Fibonacci Quarterly,Vol **43**.(2005),170-180.

[16] L. Shapiro, Catalan trigonometry, Congr. Numer. **156** (2002), 129-136.

[17] T. Tymoczko, and J. Henle, Sweet Reason: A Field Guide to Modern Logic, New York: W.H.Freeman and Company (1995).

[18] S. G. Mohanty, Lattice Path Counting and Applications, New York: Academic Press(1979).

[19] J. Riordan, An Introduction to Combinatorial Analysis (originally published: New York: John Wiley 1958), Dover Publications (2002).

[20] G. C. Rota, D. Kahaner, and A. Odlyzko, On the foundations of combinatorial theory VIII: finite operator calculus. J. Math. Anal. Appl. **42** (1973) 684-760.

[21] H.J. Ryser, Combinatorial Mathematics, Carus mathematical monographs **14** (Math. Assoc. Am.) 1963.

[22] The Online Encyclopedia of Integer Sequences (2014), published electronically at http://oeis.org.

Fixed Share Scheduling via Dynamic Weight Adjustment in Proportional Share Scheduling Systems

Hyungwoo Kim[1], Hyunmin Yoon[2], Peng Wu[2], and Minsoo Ryu[1]
[1]Department of Computer Science and Engineering, Hanyang University, Seoul, Korea
[2]Department of Electronics Computer Engineering, Hanyang University, Seoul, Korea

Abstract - *There exist many compute-intensive soft real-time applications, such as video encoding/decoding and data encryption/decryption, that require a fixed percentage of CPU cycles to maintain an acceptable QoS (Quality of Service). Unfortunately, major operating systems do not support well such compute-intensive soft real-time applications, since they commonly rely on a proportional share policy. In this paper, we present a novel scheduling policy, called FSS (fixed share scheduling), to enable a proportional share scheduler to support compute-intensive soft real-time applications as well as non-real-time applications. The goal of FSS is to guarantee an absolute, constant share of CPU cycles for soft real-time tasks regardless of workload conditions, whereas traditional proportional share schedulers focus on relative proportional guarantees. To do so, working on top of a proportional share scheduler, FSS dynamically changes the weight value of each soft real-time task to match the demanded amount of CPU share under varying workload conditions. The weighted fairness mechanism of the underlying proportional share scheduler will then provide the demanded amount of CPU share for each soft real-time task. To demonstrate the efficacy of FSS, we have implemented a fixed shared scheduling prototype in the Linux CFS (completely fair scheduler) and conducted experiments to show the correctness and efficiency of the FSS scheme.*

Keywords: Fixed Share Scheduling, Proportional Share Scheduler, QoS, soft real-time.

1 Introduction

There exist many compute-intensive soft real-time applications, such as video encoding/decoding and data encryption/decryption, that require a fixed percentage of CPU cycles to maintain an acceptable QoS (Quality of Service). For example, a HVEC (High Efficiency Video Coding) decoder demands around 60% of CPU utilization on ARM Cortex A9 processor at 1.5 GHz. When the CPU cycle demand cannot be met, the provided QoS will be significantly degraded.

Unfortunately, such compute-intensive soft real-time applications are not well supported by major operating systems. Most operating systems use a proportional share scheduling policy, often combined with a priority-based scheduling policy. The common goal of these schedulers is to achieve fairness and responsiveness as a general purpose operating system scheduler, rather than providing any resource guarantees for real-time applications.

In this paper, we present a novel scheduling policy, called FSS (fixed share scheduling), to enable a proportional share scheduler to support compute-intensive soft real-time applications as well as non-real-time applications. The goal of FSS is to guarantee an absolute, constant share of CPU cycles for soft real-time tasks regardless of workload conditions, whereas traditional proportional share schedulers focus on relative proportional guarantees. To do so, working on top of a proportional share scheduler, FSS dynamically changes the weight value of each soft real-time task to match the demanded amount of CPU share under varying workload conditions. The weighted fairness mechanism of the underlying proportional share scheduler will then provide the demanded amount of CPU share for each soft real-time task.

To demonstrate the efficacy of FSS, we have implemented a fixed shared scheduling prototype in the Linux CFS (completely fair scheduler). We have also conducted experiments to show the correctness and efficiency of the FSS scheme.

2 Fixed Share Guarantees with Proportional Share Schedulers

Proportional share scheduling provides a useful abstraction for multiplexing resources among different tasks [10]. The key idea of proportional share scheduling is to allocate resources to tasks proportional to their weights. The ideal model of proportional share resource allocation is the Generalized Processor Sharing (GPS) scheme. GPS assumes a fluid-style resource model and guarantees perfect fairness based on an infinitesimal fluid resource model. For real systems where resources cannot be provided infinitesimally, approximate scheduling schemes like WFQ (Weighted Fair Queuing) and PGPS (Packet by Packet GPS) have been proposed. Figure 1 shows the ideal proportional share scheduling and approximate proportional scheduling.

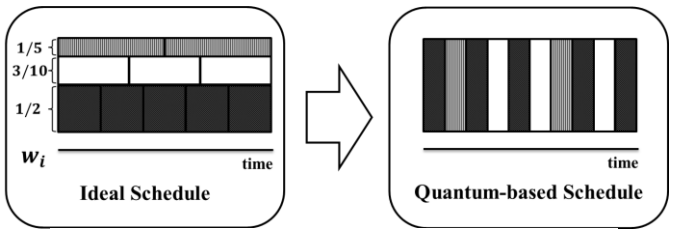


Figure 1. Proportional share scheduling.

Let $Weight_i$ be the weight of task τ_i and let Φ be the set of all active tasks at time t. The share $S_i(t)$ of a task τ_i at time t is defined as below.

$$S_i(t) = \frac{Weight_i}{\sum_{j\in\Phi} Weight_j} \quad (1)$$

As the number of tasks can change at runtime, the resource share $S_i(t)$ also changes. For example, when a new task arrives, the total weight $\sum_{j\in\Phi} Weight_j$ of tasks will increase, decreasing the share $S_i(t)$ of task τ_i. Therefore, a proportional share scheduler only guarantees a relative share of CPU cycles depending on the workload condition, rather than an absolute, fixed share of CPU time.

The goal of FSS is to guarantee a constant amount of CPU utilization for soft real-time tasks regardless of dynamic changes in workload. This idea can be easily implemented in a proportional share scheduling system by simply changing the weight values of soft real-time tasks during runtime. Let U_i be the CPU utilization demanded by a soft real-time task τ_i. The actual CPU utilization allocated to task τ_i is determined by weight values of other non-real-time tasks as follows.

$$U_i = \frac{Weight_i}{\sum_{j\in\Phi, j\neq i} Weight_j + Weight_i} \quad (2)$$

A simple arithmetic manipulation gives the following equation.

$$Weight_i = \frac{U_i}{1 - U_i} \times \sum_{j\in\Phi, j\neq i} Weight_j \quad (3)$$

Therefore, we can determine the new weight value of task τ_i using Eq. (3) for a given value of fixed share demand U_i.

3 Incorporating Fixed Share Scheduling into Linux CFS

In this section, we demonstrate that the proposed idea can be incorporated into the Linux completely fair scheduler (CFS).

CFS has been introduced since Linux 2.6.23 to provide weighted fairness. CFS uses virtual runtime for each task to keep track of the actual CPU time the task has received and the ideal CPU time the task should have received. Let $\mathrm{VR}(\tau_i, t)$ be the virtual runtime of task τ_i at time t and $A(\tau_i, t)$ be the CPU time consumed by the task τ_i by time t. Let $Weight_0$ be the weight for nice value 0.

$$\mathrm{VR}(\tau_i, t) = \frac{Weight_0}{Weight_i} \times A(\tau_i, t) \quad (4)$$

```
static cons int prio_to_weight[40] = {
 /* -20 */     88761,     71755,     56483,     46273,     36291,
 /* -15 */     29154,     23254,     18705,     14949,     11916,
 /* -10 */      9548,      7620,      6100,      4904,      3906,
 /*  -5 */      3121,      2501,      1991,      1586,      1277,
 /*   0 */      1024,       820,       655,       526,       423,
 /*   5 */       335,       272,       215,       172,       137,
 /*  10 */       110,        87,        70,        56,        45,
 /*  15 */        36,        29,        23,        18,        15,
};
```

Figure 2. Mapping between nice and weight values.

Note that $Weight_i$ is determined by the nice value of task τ_i. The weight value has a range from 88761 (nice level -20) to 15 (nice level 19) where small nice values have large weights. The nice values are stored in the *prio_to_weight[]* array in "sched.h" of the Linux kernel source code as shown in Figure 2.

Note that CFS guarantees weighted fairness by allocating different time slices to tasks of different weights. Specifically, the length of the time slice TS_i of τ_i is proportional to its weight as below.

$$TS_i = \frac{Weight_i}{\sum_{j\in\Phi} Weight_j} \times \mathrm{P} \quad (5)$$

where P is the length of round in which the scheduler executes every task only once scanning the run queue. It is defined by the following.

$$P = \begin{cases} sysctl\_sched\_latency, if\ n < sched\_nr\_latency \\ sysctl\_sched\_min\_granularity \times n, otherwise \end{cases} \quad (6)$$

where n is the total number of tasks in the run queue, $sysctl\_sched\_latency$ and $sysctl\_sched\_min\_granularity$ are constants specified as 6ms and 0.75ms, respectively, in our Linux setting, and $sched\_nr\_latency$ is defined by Eq. (7).

$$sched\_nr\_latency = \frac{sysctl\_sched\_latency}{sysctl\_sched\_min\_granularity} \quad (7)$$

which becomes 8 as $sysctl\_sched\_latency$ is 6ms and $sysctl\_sched\_min\_granularity$ is 0.75ms.

A fixed share scheduling scheme can be easily implemented in Linux by slightly modifying the scheduler to perform weight recalculation. Since CFS can also be viewed as an approximate implementation of GPS, the implementation of fixed share scheduling in CFS is not much

different from implementing it in other proportional share schedulers. A major difference is that CFS uses discrete weight values ranging from 15 to 88761, which requires us to find the nearest weight value that is greater than the recalculated weight value. For example, when a new weight value should be 8100, we need to choose 9548 and assign -10 as a new nice value.

Hardware	***CPU***	Samsung Exynos5 Octa big.LITTLE processor
	RAM	2 GByte LPDDR3 RAM
Software	***Operating System***	Android 4.2.2 Jelly Bean, Kernel Version : 3.4.5
	GNU gcc	arm-eabi-gcc (GCC) 4.6.x-google 20120106

Figure 3. Implementation environment.

4 Experimental Evaluation

We have implemented and incorporated FSS into Linux CFS. Figure 3 summarizes the development environment. We then conducted a set of simple experiment to validate the correct behavior of our fixed share scheduling scheme.

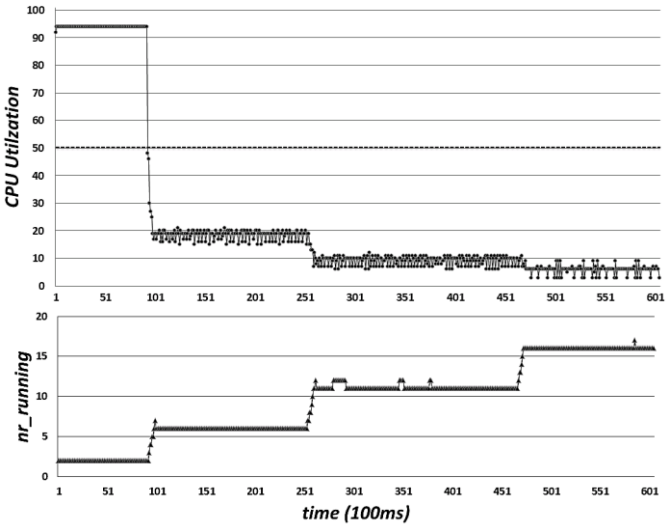


Figure 4. CPU utilization and *nr_running* in the original Linux CFS scheduler

In the first set of experiments, we observed the utilization of a target task with nice value of 0 under the original CFS scheme. Figure 4 shows that the utilization of target task decreases as the number of tasks, *nr_running*, increases.

In the second set of experiments, we observed the utilization of a target task under the proposed fixed share scheduling scheme. The fixed share demand of target task was set to be 50% of CPU utilization. Figure 5 shows that the actual utilization of target task is maintained around 50% even though the number of tasks varies dynamically.

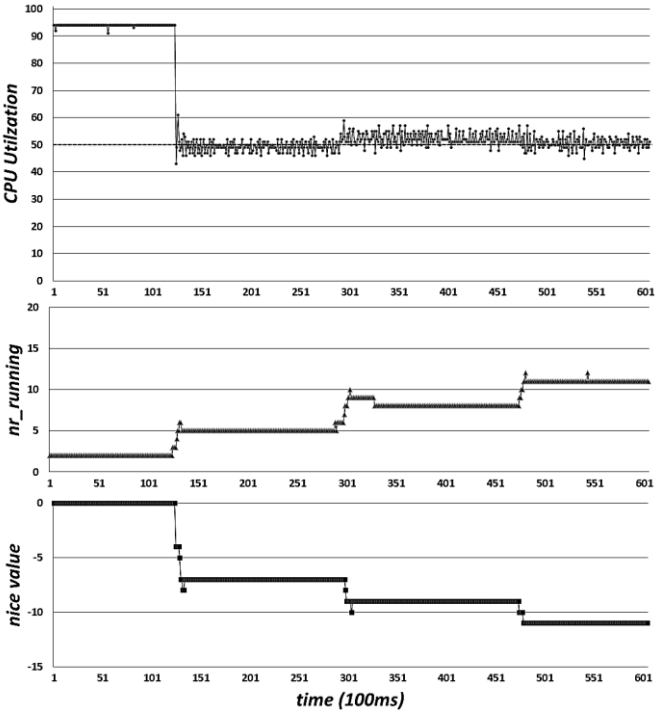


Figure 5. CPU utilization, *nr_running*, and nive value in the proposed FSS scheme

5 Conclusion

In this paper, we have presented a FSS (fixed share scheduling) that can guarantee an absolute, constant share of CPU cycles for soft real-time tasks. FSS works on top of a traditional proportional share scheduler and dynamically changes the weight value of each soft real-time task to guarantee the demanded amount of CPU share under varying workload conditions. To demonstrate the efficacy of FSS, we have also implemented a fixed shared scheduling prototype in the Linux CFS (completely fair scheduler). Our evaluation shows that FSS provides accurate fixed share guarantees under varying workload conditions.

6 Acknowledgment

This work was supported partly by Seoul Creative Human Development Program (HM120006), partly by Mid-career Researcher Program through NRF (National Research Foundation) grant funded by the MEST (Ministry of Education, Science and Technology) (NRF-2011-0015997), partly by the IT R&D Program of MKE/KEIT [10035708, "The Development of CPS (Cyber-Physical Systems) Core Technologies for High Confidential Autonomic Control Software"], and partly by the MSIP(Ministry of Science,

ICT&Future Planning), Korea, under the CITRC (Convergence Information Technology Research Center) support program (NIPA-2013-H0401-13-1008) supervised by the NIPA (National IT Industry Promotion Agency).

7 References

[1] K. Nahrstedt, A. Arefin, and Z. Yang, "QoS and resource management in distributed interactive multimedia environments," *Multimedia Tools and Applications*, Vol. 51, Issue 1, pp. 99-132, January 2011.

[2] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single-node case," *IEEE/ACM Transactions on Networking*, Vol. 1, Issue 3, pp. 344-357, June 1993.

[3] C. S. Pabla, "Completely fair scheduler," *Linux Journal*, Vol. 2009, Issue 184, August 2009.

[4] S. Huh, J. Yoo, M. Kim, and S. Hong, "Providing Fair Share Scheduling on Multicore Cloud Servers via Virtual Runtime-based Task Migration Algorithm," *2012 IEEE 32nd International Conference on Distributed Computing Systems*, pp. 606-614, June 2012.

[5] D. Ok, B. Song, H. Yoon, P. Wu, J. Lee, J. Park, and M. Ryu, "Lag-Based Load Balancing for Linux-based Multicore Systems," *The 2013 International Conference on Foundations of Computer Science*, July 2013.

[6] M. T. Jones, "Inside the Linux 2.6 Completely Fair Scheduler," December 2009, [Online]. Available: http://www.ibm.com/developerworks/library/l-completely-fair-scheduler/

[7] Wikipedia, "Red-black tree," [Online]. Available: http://en.wikipedia.org/wiki/Red-black_tree

[8] M. J. Bach, "The Design of the UNIX Operating System," *Prentice Hall*, 1986.

[9] T. Li, D. Baumberger, and S. Hahn, "Efficient and scalable multiprocessor fair scheduling using distributed weighted round-robin," *ACM SIGPLAN Notices*, Vol. 44, Issue 4, pp. 65-74, April 2009.

[10] J. Nieh, C. Vaill, and H. Zhong, "Virtual-Time Round-Robin: An O(1) Proportional Share Scheduler," *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, pp. 245-259, June 2001.

SESSION

QUANTUM COMPUTING

Chair(s)

TBA

Quantum-Intersection Equivalents of the Orthomodularity Law in Quantum Logic: Part 1

Jack K. Horner
P. O. Box 266
Los Alamos, New Mexico 87544 USA

FCS 2014

Abstract

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of quantum-mechanical behaviors and operations. In much the same way that the structure of conventional propositional (Boolean) logic (BL) is the logic of the description of the behavior of classical physical systems and is isomorphic to a Boolean algebra (BA), so also the algebra, C(H), of closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space is a logic of the descriptions of the behavior of quantum mechanical systems and is a model of an ortholattice (OL). An OL can thus be thought of as a kind of "quantum logic" (QL). C(H) is also a model of an orthomodular lattice, which is an OL conjoined with the orthomodularity axiom (OMA). The rationalization of the OMA as a claim proper to physics has proven problematic, motivating the question of whether the OMA and its equivalents are required in an adequate characterization of QL. Here I provide an automated deduction of three quantum-intersection-based equivalents of the OMA. The proofs may be novel.

Keywords: automated deduction, quantum computing, orthomodular lattice, Hilbert space

1.0 Introduction

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of the description of quantum-mechanical behaviors ([1], [17], [18], [20]). In much the same way that conventional propositional (Boolean) logic (BL,[12]) is the logical structure of description of the behavior of classical physical systems (e.g. "the measurements of the position and momentum of particle P are commutative", i.e., can be measured in either other, yielding the same results) and is isomorphic to a Boolean lattice ([10], [11], [19]), so also the algebra, *C(H)*, of the closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space *H* ([1], [4], [6], [9], [13]) is *a* logic of the descriptions of the behavior of quantum mechanical systems (e.g., "the measurements of the position and momentum of particle P are *not* commutative") and is a model ([10]) of an ortholattice (OL; [4]). An OL can thus be thought of as a kind of "quantum logic" (QL; [19]). *C(H)* is also a model of (i.e., isomorphic to a set of sentences which hold in) an orthomodular lattice (OML; [4], [7]), which is an OL conjoined with the orthomodularity axiom (OMA; see Figure 1). The rationalization of the OMA as a claim proper to physics has proven problematic ([13], Section 5-6), motivating the question of whether the OMA is required in an adequate characterization of QL. Thus formulated, the question suggests that the OMA and its equivalents are specific to an OML, and that as a consequence, banning the OMA from QL yields a "truer" quantum logic.

Lattice axioms

```
      x = c(c(x))                       (AxLat1)
      x v y = y v x                     (AxLat2)
      (x v y) v z = x v (y v z)          (AxLat3)
      (x ^ y) ^ z = x ^ (y ^ z)          (AxLat4)
      x v (x ^ y) = x                   (AxLat5)
      x ^ (x v y) = x                   (AxLat6)
```

Ortholattice axioms

```
      c(x) ^ x = 0                      (AxOL1)
      c(x) v x = 1                      (AxOL2)
      x ^ y = c(c(x) v c(y))            (AxOL3)
```

Orthomodularity axiom

```
      y v (c(y) ^ (x v y)) = x v y      (OMA)
```

Definitions of implications and partial order

```
   i1(x,y) = c(x) v (x ^ y).
   i2(x,y) = i1(c(y), c(x).
   i3(x,y) = (c(x) ^ y) v (c(x) ^ c(y)) v i1(x,y).
   i4(x,y) = i3(c(y), c(x)).
   i5(x,y) = (x ^ y) v (c(x) ^ y) v (c(x) ^ c(y)).
   le(x,y) = (x = (x ^ y)).
```

```
Definitions of indexed intersection
   int1(x,y) = c(i1(x,c(y)))
   int2(x,y) = c(i2(x,c(y)))
   int3(x,y) = c(i3(x,c(y)))
   int4(x,y) = c(i4(x,c(y)))
   int5(x,y) = c(i5(x,c(y)))
```

```
where
   x, y are variables ranging over lattice nodes
   ^ is lattice meet
   v is lattice join
   c(x) is the orthocomplement of x
   i1(x,y) means x →1 y (Sasaki implication)
   i2(x,y) means x →2 y (Dishkant implication)
   i3(x,y) means x →3 y (Kalmbach implication)
   i4(x,y) means x →4 y (non-tollens implication)
   i5(x,y) means x →5 y (relevance implication)
   le(x,y) means x ≤ y
   <-> means if and only if
   =  is equivalence ([12])
   1 is the maximum lattice element (= x v c(x))
   0 is the minimum lattice element (= c(1))
```

Figure 1. Lattice, ortholattice, orthomodularity axioms, and some definitions.

There are at least 21 nominal equivalents (in the sense that ortholattice theory, together with these "equivalents", imply the OMA, and vice versa) of the OMA in quantum logic ([5], Theorem 2.5); as nominal equivalents, they are thus of import to optimizing quantum circuit design. Among these is the Proposition shown in Figure 2:

$$x \cap_i y \quad <-> \quad x \perp y$$

where

$x \cap_i y$ means $c(x \rightarrow_i c(y))$
$x \perp y$ means $le(x, c(y))$
$i = 1,2,3,4,5$

Figure 2. Proposition 2.12 of [5]

2.0 Method

The OML axiomatizations of Megill, Pavičić, and Horner ([5], [14], [15], [16], [21], [22]) were implemented in a *prover9* ([2]) script ([3]) configured to derive OMA from Proposition 2.12 of [5], for each of i = 1,2,3 together with ortholattice theory (orthomodular lattice theory, without the OMA), then executed in that framework on a Dell Inspiron 545 with an Intel Core2 Quad CPU Q8200 (clocked @ 2.33 GHz) and 8.00 GB RAM, running under the *Windows Vista Home Premium /Cygwin* operating environment.

3.0 Results

Figure 3 shows the proofs, generated by [3] on the platform described in Section 2.0, that Proposition 2.12 (for each of i = 1,2,3), together with ortholattice theory, imply the OMA.

```
============================== PROOF =================================

% Proof 1 at 88.30 (+ 2.62) seconds: "OMA".
% Length of proof is 42.
% Level of proof is 11.

1 le(x,y) <-> x = x ^ y # label("Df: less than") # label(non_clause).  [assumption].
2 perp(x,y) <-> le(x,c(y)) # label("Df: perpendicular") # label(non_clause).
[assumption].
4 int1(x,y) = 0 <-> perp(x,y) # label("Hypothesis for Proposition 2.10int1") #
label(non_clause).  [assumption].
5 y v (c(y) ^ (x v y)) = x v y # label("OMA") # label(non_clause) # label(goal).  [goal].
7 -perp(x,y) | le(x,c(y)) # label("Df: perpendicular").  [clausify(2)].
8 int1(x,y) != 0 | perp(x,y) # label("Hypothesis for Proposition 2.10int1").
[clausify(4)].
12 x = c(c(x)) # label("AxL1").  [assumption].
13 c(c(x)) = x.  [copy(12),flip(a)].
14 x v y = y v x # label("AxL2").  [assumption].
15 (x v y) v z = x v (y v z) # label("AxL3").  [assumption].
17 x v (x ^ y) = x # label("AxL5").  [assumption].
19 c(x) ^ x = 0 # label("AxOL1").  [assumption].
20 c(x) v x = 1 # label("AxOL2").  [assumption].
21 x v c(x) = 1.  [copy(20),rewrite([14(2)])].
22 x ^ y = c(c(x) v c(y)) # label("AxOL3").  [assumption].
24 i1(x,y) = c(x) v (x ^ y) # label("Df: i1").  [assumption].
25 i1(x,y) = c(x) v c(c(x) v c(y)).  [copy(24),rewrite([22(3)])].
44 int1(x,y) = c(i1(x,c(y))) # label("Df: int1").  [assumption].
45 int1(x,y) = c(c(x) v c(c(x) v y)).  [copy(44),rewrite([25(3),13(5)])].
54 -le(x,y) | x ^ y = x # label("Df: less than").  [clausify(1)].
55 -le(x,y) | c(c(x) v c(y)) = x.  [copy(54),rewrite([22(2)])].
58 c1 v (c(c1) ^ (c2 v c1)) != c2 v c1 # label("OMA") # answer("OMA").  [deny(5)].
59 c1 v c(c1 v c(c1 v c2)) != c1 v c2 # answer("OMA").
[copy(58),rewrite([14(6),22(7),13(4),14(12)])].
60 int1(x,y) != 0 | le(x,c(y)).  [resolve(8,b,7,a)].
61 c(c(x) v c(c(x) v y)) != 0 | le(x,c(y)).  [copy(60),rewrite([45(1)])].
64 c(1) = 0.  [back_rewrite(19),rewrite([22(2),13(2),21(2)])].
```

```
66 x v c(c(x) v c(y)) = x.  [back_rewrite(17),rewrite([22(1)])].
68 x v (y v z) = y v (x v z).  [para(14(a,1),15(a,1,1)),rewrite([15(2)])].
75 x v (y v c(x v y)) = 1.  [para(21(a,1),15(a,1)),flip(a)].
80 c(x v c(x v y)) != 0 | le(c(x),c(y)).  [para(13(a,1),61(a,1,1,1)),rewrite([13(2)])].
103 x v c(c(x) v y) = x.  [para(13(a,1),66(a,1,2,1,2))].
143 x v c(y v c(x)) = x.  [para(14(a,1),103(a,1,2,1))].
145 x v (y v c(c(x v y) v z)) = x v y.  [para(103(a,1),15(a,1)),flip(a)].
154 x v (y v c(y v x)) = 1.  [para(14(a,1),75(a,1,2,2,1))].
207 x v (y v c(z v c(x v y))) = x v y.  [para(143(a,1),15(a,1)),flip(a)].
214 c(x v c(y v x)) != 0 | le(c(x),c(y)).  [para(14(a,1),80(a,1,1,2,1))].
1657 c(x v (c(y v x) v c(c(y v x) v z))) != 0 | le(c(x v c(c(y v x) v z)),c(y)).
[para(145(a,1),214(a,1,1,2,1)),rewrite([14(8),68(8)])].
24240 le(c(x v c(x v c(y v x))),c(y)).
[para(154(a,1),1657(a,1,1)),rewrite([64(2),14(6)]),xx(a)].
24245 c(x v c(x v c(y v x))) = c(y v x).
[hyper(55,a,24240,a),rewrite([13(7),13(7),14(6),207(6)]),flip(a)].
24252 x v c(x v c(y v x)) = y v x.
[para(24245(a,1),13(a,1,1)),rewrite([13(3)]),flip(a)].
24330 x v c(x v c(x v y)) = y v x.  [para(14(a,1),24252(a,1,2,1,2,1))].
24348 $F # answer("OMA").  [back_rewrite(59),rewrite([24330(9),14(3)]),xx(a)].

============================== end of proof ==========================


============================== PROOF =================================

% Proof 1 at 33.74 (+ 0.87) seconds: "OMA".
% Length of proof is 50.
% Level of proof is 12.

1 le(x,y) <-> x = x ^ y # label("Df: less than") # label(non_clause).  [assumption].
2 perp(x,y) <-> le(x,c(y)) # label("Df: perpendicular") # label(non_clause).
[assumption].
4 int2(x,y) = 0 <-> perp(x,y) # label("Hypothesis for Proposition 2.10int2") #
label(non_clause).  [assumption].
5 y v (c(y) ^ (x v y)) = x v y # label("OMA") # label(non_clause) # label(goal).  [goal].
7 -perp(x,y) | le(x,c(y)) # label("Df: perpendicular").  [clausify(2)].
8 int2(x,y) != 0 | perp(x,y) # label("Hypothesis for Proposition 2.10int2").
[clausify(4)].
12 x = c(c(x)) # label("AxL1").  [assumption].
13 c(c(x)) = x.  [copy(12),flip(a)].
14 x v y = y v x # label("AxL2").  [assumption].
15 (x v y) v z = x v (y v z) # label("AxL3").  [assumption].
17 x v (x ^ y) = x # label("AxL5").  [assumption].
18 x ^ (x v y) = x # label("AxL6").  [assumption].
19 c(x) ^ x = 0 # label("AxOL1").  [assumption].
20 c(x) v x = 1 # label("AxOL2").  [assumption].
21 x v c(x) = 1.  [copy(20),rewrite([14(2)])].
22 x ^ y = c(c(x) v c(y)) # label("AxOL3").  [assumption].
26 i2(x,y) = c(c(y)) v (c(y) ^ c(x)) # label("Df: i2").  [assumption].
27 i2(x,y) = y v c(y v x).  [copy(26),rewrite([13(3),22(4),13(3),13(3)])].
46 int2(x,y) = c(i2(x,c(y))) # label("Df: int2").  [assumption].
47 int2(x,y) = c(c(y) v c(c(y) v x)).  [copy(46),rewrite([27(3)])].
54 -le(x,y) | x ^ y = x # label("Df: less than").  [clausify(1)].
55 -le(x,y) | c(c(x) v c(y)) = x.  [copy(54),rewrite([22(2)])].
58 c1 v (c(c1) ^ (c2 v c1)) != c2 v c1 # label("OMA") # answer("OMA").  [deny(5)].
59 c1 v c(c1 v c(c1 v c2)) != c1 v c2 # answer("OMA").
[copy(58),rewrite([14(6),22(7),13(4),14(12)])].
60 int2(x,y) != 0 | le(x,c(y)).  [resolve(8,b,7,a)].
61 c(c(x) v c(c(x) v y)) != 0 | le(y,c(x)).  [copy(60),rewrite([47(1)])].
64 c(1) = 0.  [back_rewrite(19),rewrite([22(2),13(2),21(2)])].
65 c(c(x) v c(x v y)) = x.  [back_rewrite(18),rewrite([22(2)])].
66 x v c(c(x) v c(y)) = x.  [back_rewrite(17),rewrite([22(1)])].
68 x v (y v z) = y v (x v z).  [para(14(a,1),15(a,1,1)),rewrite([15(2)])].
75 x v (y v c(x v y)) = 1.  [para(21(a,1),15(a,1)),flip(a)].
80 c(x v c(x v y)) != 0 | le(y,x).  [para(13(a,1),61(a,1,1,1)),rewrite([13(2),13(8)])].
93 c(0 v c(x)) = x.  [para(21(a,1),65(a,1,1,2,1)),rewrite([64(3),14(3)])].
99 x v c(c(x) v y) = x.  [para(13(a,1),66(a,1,2,1,2))].
104 x v x = x.  [para(64(a,1),66(a,1,2,1,2)),rewrite([14(3),93(4)])].
116 x v (x v y) = x v y.  [para(104(a,1),15(a,1,1)),flip(a)].
135 x v c(y v c(x)) = x.  [para(14(a,1),99(a,1,2,1))].
```

```
137 x v (y v c(c(x v y) v z)) = x v y.  [para(99(a,1),15(a,1)),flip(a)].
150 x v (y v c(y v x)) = 1.  [para(14(a,1),75(a,1,2,2,1))].
196 c(x) v c(y v x) = c(x).  [para(13(a,1),135(a,1,2,1,2))].
198 x v (y v c(z v c(x v y))) = x v y.  [para(135(a,1),15(a,1)),flip(a)].
217 c(x v c(y v x)) != 0 | le(y,x).  [para(14(a,1),80(a,1,1,2,1))].
322 c(x v y) v c(x v (z v y)) = c(x v y).  [para(68(a,1),196(a,1,2,1))].
1123 c(x v (c(y v x) v c(c(y v x) v z))) != 0 | le(y,x v c(c(y v x) v z)).
[para(137(a,1),217(a,1,1,2,1)),rewrite([14(8),68(8)])].
3899 c(x v y) v c(x v c(z v c(x v y))) = c(x v c(z v c(x v y))).
[para(198(a,1),322(a,1,2,1)),rewrite([14(9)])].
23785 le(x,y v c(y v c(x v y))).
[para(150(a,1),1123(a,1,1)),rewrite([64(2),14(6)]),xx(a)].
23791 le(x,y v c(y v c(y v x))).  [para(14(a,1),23785(a,2,2,1,2,1))].
23802 le(x v y,x v c(x v c(x v y))).  [para(116(a,1),23791(a,2,2,1,2,1))].
23810 x v c(x v c(x v y)) = x v y.  [hyper(55,a,23802,a),rewrite([3899(9),13(7)])].
23811 $F # answer("OMA").  [resolve(23810,a,59,a)].

============================== end of proof ==========================


============================== PROOF =================================

% Proof 1 at 25.24 (+ 0.73) seconds: "OMA".
% Length of proof is 55.
% Level of proof is 14.

1 le(x,y) <-> x = x ^ y # label("Df: less than") # label(non_clause).  [assumption].
2 perp(x,y) <-> le(x,c(y)) # label("Df: perpendicular") # label(non_clause).
[assumption].
4 int3(x,y) = 0 <-> perp(x,y) # label("Hypothesis for Proposition 2.10int3") #
label(non_clause).  [assumption].
5 y v (c(y) ^ (x v y)) = x v y # label("OMA") # label(non_clause) # label(goal).  [goal].
7 -perp(x,y) | le(x,c(y)) # label("Df: perpendicular").  [clausify(2)].
8 int3(x,y) != 0 | perp(x,y) # label("Hypothesis for Proposition 2.10int3").
[clausify(4)].
12 x = c(c(x)) # label("AxL1").  [assumption].
13 c(c(x)) = x.  [copy(12),flip(a)].
14 x v y = y v x # label("AxL2").  [assumption].
15 (x v y) v z = x v (y v z) # label("AxL3").  [assumption].
17 x v (x ^ y) = x # label("AxL5").  [assumption].
18 x ^ (x v y) = x # label("AxL6").  [assumption].
19 c(x) ^ x = 0 # label("AxOL1").  [assumption].
20 c(x) v x = 1 # label("AxOL2").  [assumption].
21 x v c(x) = 1.  [copy(20),rewrite([14(2)])].
22 x ^ y = c(c(x) v c(y)) # label("AxOL3").  [assumption].
28 i3(x,y) = ((c(x) ^ y) v (c(x) ^ c(y))) v (c(x) v (x ^ y)) # label("Df: i3").
[assumption].
29 i3(x,y) = c(x v y) v (c(x v c(y)) v (c(x) v c(c(x) v c(y)))).
[copy(28),rewrite([22(3),13(3),22(7),13(6),13(6),14(7),22(9),15(14)])].
48 int3(x,y) = c(i3(x,c(y))) # label("Df: int3").  [assumption].
49 int3(x,y) = c(c(x v c(y)) v (c(x v y) v (c(x) v c(c(x) v y)))).
[copy(48),rewrite([29(3),13(6),13(10)])].
54 -le(x,y) | x ^ y = x # label("Df: less than").  [clausify(1)].
55 -le(x,y) | c(c(x) v c(y)) = x.  [copy(54),rewrite([22(2)])].
58 c1 v (c(c1) ^ (c2 v c1)) != c2 v c1 # label("OMA") # answer("OMA").  [deny(5)].
59 c1 v c(c1 v c(c1 v c2)) != c1 v c2 # answer("OMA").
[copy(58),rewrite([14(6),22(7),13(4),14(12)])].
60 int3(x,y) != 0 | le(x,c(y)).  [resolve(8,b,7,a)].
61 c(c(x v c(y)) v (c(x v y) v (c(x) v c(c(x) v y)))) != 0 | le(x,c(y)).
[copy(60),rewrite([49(1)])].
64 c(1) = 0.  [back_rewrite(19),rewrite([22(2),13(2),21(2)])].
65 c(c(x) v c(x v y)) = x.  [back_rewrite(18),rewrite([22(2)])].
66 x v c(c(x) v c(y)) = x.  [back_rewrite(17),rewrite([22(1)])].
68 x v (y v z) = y v (x v z).  [para(14(a,1),15(a,1,1)),rewrite([15(2)])].
70 c(c(x) v (c(x v y) v (c(x v c(y)) v c(c(x) v y)))) != 0 | le(x,c(y)).
[back_rewrite(61),rewrite([68(11),68(12),68(11)])].
77 x v (y v c(x v y)) = 1.  [para(21(a,1),15(a,1)),flip(a)].
89 c(c(x) v c(y v x)) = x.  [para(14(a,1),65(a,1,1,2,1))].
91 c(0 v c(x)) = x.  [para(21(a,1),65(a,1,1,2,1)),rewrite([64(3),14(3)])].
92 1 v x = 1.  [para(64(a,1),65(a,1,1,1)),rewrite([91(6)])].
```

```
97 x v c(c(x) v y) = x.  [para(13(a,1),66(a,1,2,1,2))].
101 x v 0 = x.  [para(21(a,1),66(a,1,2,1)),rewrite([64(2)])].
102 x v x = x.  [para(64(a,1),66(a,1,2,1,2)),rewrite([14(3),91(4)])].
103 x v (y v c(x)) = y v 1.  [para(21(a,1),68(a,1,2)),flip(a)].
112 x v 1 = 1.  [para(92(a,1),14(a,1)),flip(a)].
113 x v (y v c(x)) = 1.  [back_rewrite(103),rewrite([112(5)])].
114 0 v x = x.  [para(101(a,1),14(a,1)),flip(a)].
126 x v (x v y) = x v y.  [para(102(a,1),15(a,1,1)),flip(a)].
128 x v (y v x) = y v x.  [para(102(a,1),15(a,2,2)),rewrite([14(2)])].
141 c(x) v (y v x) = 1.  [para(13(a,1),113(a,1,2,2))].
149 c(x v c(y v x)) != 0 | le(c(x),c(y v x)).
[para(141(a,1),70(a,1,1,2,1,1)),rewrite([13(2),64(2),89(6),13(3),128(3),114(5),126(4)])].
150 x v c(y v c(x)) = x.  [para(14(a,1),97(a,1,2,1))].
152 x v (y v c(c(x v y) v z)) = x v y.  [para(97(a,1),15(a,1)),flip(a)].
174 x v (y v c(y v x)) = 1.  [para(14(a,1),77(a,1,2,2,1))].
735 c(x v (c(y v x) v c(c(y v x) v z))) != 0 | le(c(x v c(c(y v x) v z)),c(y v x)).
[para(152(a,1),149(a,1,1,2,1)),rewrite([14(8),68(8),152(23)])].
20284 le(c(x v c(x v c(y v x))),c(y v x)).
[para(174(a,1),735(a,1,1)),rewrite([64(2),14(6)]),xx(a)].
25500 c(x v c(x v c(y v x))) = c(y v x).
[hyper(55,a,20284,a),rewrite([13(7),13(8),14(7),68(7),150(6),128(2)]),flip(a)].
25619 x v c(x v c(y v x)) = y v x.
[para(25500(a,1),13(a,1,1)),rewrite([13(3)]),flip(a)].
25715 x v c(x v c(x v y)) = y v x.  [para(14(a,1),25619(a,1,2,1,2,1))].
25754 $F # answer("OMA").  [back_rewrite(59),rewrite([25715(9),14(3)]),xx(a)].

============================== end of proof ==========================
```

Figure 3. Summary of a *prover9* ([2]) proof of Proposition 2.11, for each of i = 1,2,3. The proofs assume the default inference rules of *prover9*. The detailed syntax and semantics of these notations can be found in [2]. All *prover9* proofs are by default proofs by contradiction.

The total time to produce the proofs in Figure 3 on the platform described in Section 2.0 was approximately 150 seconds.

4.0 Discussion

The results of Section 3.0 motivate several observations:

1. With the exception of Proposition 2.12, the proofs in Figure 3 for i=2 and i=3 both use L1, L2, L3, L5. L6, OL1, OL2, and OL3. The proof for i=1, in contrast, uses L1, L2, L3, L5, OL1, OL2, and OL3.

2. The proofs in Section 3.0 may be novel.

3. Companion papers provide proofs for i = 4,5, and for orthomodular lattice theory implies Propositions 2.12i, i = 1,2,3,4,5, and for the converse propositions.

4. Proposition 2.12 can be regarded as a definition of quantum intersection; thus, this paper together with the papers mentioned in (3), constitutes a proof that the definition of quantum intersection is equivalent to the OMA in orthomodular quantum logic. Companion papers derive equivalences for the OMA with definitions of quantum-identity and quantum-union. Collectively, these papers provide a theory of equivalence of the OMA with the quantum connectives. In light of these equivalences, QL without the OMA would hardly qualify as a logic.

5.0 Acknowledgements

This work benefited from discussions with Tom Oberdan, Frank Pecchioni, Tony Pawlicki, and the late John K. Prentice, whose passion for foundations of physics inspired those of us privileged to have known him. For any infelicities that remain, I am solely responsible.

6.0 References

[1] von Neumann J. *Mathematical Foundations of Quantum Mechanics*. 1936. Translated by R. T. Beyer. Princeton. 1983.
[2] McCune WW. *prover9 and mace4*. URL http://www.cs.unm.edu/~mccune/prover9/. 2009.
[3] Horner JK. MHT *prover9* scripts. 2011. Available from the author on request.
[4] Dalla Chiara ML and Giuntini R. *Quantum Logics*. URL http://xxx.lanl.gov/abs/quant-ph/0101028. 2004.
[5] Megill ND and Pavičić M. Orthomodular lattices and quantum algebra. *International Journal of Theoretical Physics* 40 (2001), pp. 1387-1410.
[6] Akhiezer NI and Glazman IM. *Theory of Linear Operators in Hilbert Space. Volume I*. Translated by M. Nestell. Frederick Ungar. 1961.
[7] Holland, Jr. SS Orthomodularity in infinite dimensions: a theorem of M. Solèr. *Bulletin of the American Mathematical Society* 32 (1995), pp. 205-234.
[8] Marsden EL and Herman LM. *A condition for distribution in orthomodular lattices.* Kansas State University Technical Report #40. 1974.
[9] Knuth DE and Bendix PB. Simple word problems in universal algebras. In J. Leech, ed. *Computational Problems in Abstract Algebra.* Pergamon Press. 1970. pp. 263-297.
[10] Chang CC and Keisler HJ. *Model Theory*. North-Holland. 1990. pp. 38-39.
[11] Birkhoff G. *Lattice Theory*. Third Edition. American Mathematical Society. 1967.
[12] Church A. *Introduction to Mathematical Logic. Volume I.* Princeton. 1956.
[13] Jauch J. *Foundations of Quantum Mechanics.* Addison-Wesley. 1968.
[14] Megill ND. *Metamath*. URL http://us.metamath.org/qlegif/mmql.html#unify. 2004.
[15] Horner JK. An automated deduction system for orthomodular lattice theory. *Proceedings of the 2005 International Conference on Artificial Intelligence.* CSREA Press. 2005. pp. 260-265.
[16] Horner JK. An automated equational logic deduction of join elimination in orthomodular lattice theory. *Proceedings of the 2007 International Conference on Artificial Intelligence.* CSREA Press. 2007. pp. 481-488.
[17] Messiah A. *Quantum Mechanics*. Dover. 1958.
[18] Horner JK. Using automated theorem-provers to aid the design of efficient compilers for quantum computing. *Los Alamos National Laboratory Quantum Institute Workshop*. December 9–10, 2002. URL http://www.lanl.gov/science/centers/quantum/qls_pdfs/horner.pdf.
[19] Birkhoff G and von Neumann J. The logic of quantum mechanics. *Annals of Mathematics* 37 (1936), 823-243.
[20] Nielsen MA and Chuang L . *Quantum Computation and Quantum Information*. Cambridge. 2000.
[21] Pavičić M and Megill N. Quantum and classical implicational algebras with primitive implication. *International Journal of Theoretical Physics* 37 (1998), 2091-2098. ftp://m3k.grad.hr/pavicic/quantum-logic/1998-int-j-theor-phys-2.ps.gz.
[22] Horner JK. An automated deduction of the relative strength of orthomodular and weakly orthomodular lattice theory. *Proceedings of the 2009 International Conference on Artificial Intelligence*. CSREA Press. 2009. pp. 525-530.

Quantum-Intersection Equivalents of the Orthomodularity Law in Quantum Logic: Part 2

Jack K. Horner
P. O. Box 266
Los Alamos, New Mexico 87544 USA
jhorner@cybermesa.com

FCS 2014

Abstract

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of quantum-mechanical behaviors and operations. In much the same way that the structure of conventional propositional (Boolean) logic (BL) is the logic of the description of the behavior of classical physical systems and is isomorphic to a Boolean algebra (BA), so also the algebra, C(H), of closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space is a logic of the descriptions of the behavior of quantum mechanical systems and is a model of an ortholattice (OL). An OL can thus be thought of as a kind of "quantum logic" (QL). C(H) is also a model of an orthomodular lattice, which is an OL conjoined with the orthomodularity axiom (OMA). The rationalization of the OMA as a claim proper to physics has proven problematic, motivating the question of whether the OMA and its equivalents are required in an adequate characterization of QL. Here I provide an automated deduction of two quantum-intersection-based equivalents of the OMA. The proofs may be novel.

Keywords: automated deduction, quantum computing, orthomodular lattice, Hilbert space

1.0 Introduction

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of the description of quantum-mechanical behaviors ([1], [17], [18], [20]). In much the same way that conventional propositional (Boolean) logic (BL,[12]) is the logical structure of description of the behavior of classical physical systems (e.g. "the measurements of the position and momentum of particle P are commutative", i.e., can be measured in either other, yielding the same results) and is isomorphic to a Boolean lattice ([10], [11], [19]), so also the algebra, *C(H)*, of the closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space *H* ([1], [4], [6], [9], [13]) is *a* logic of the descriptions of the behavior of quantum mechanical systems (e.g., "the measurements of the position and momentum of particle P are *not* commutative") and is a model ([10]) of an ortholattice (OL; [4]). An OL can thus be thought of as a kind of "quantum logic" (QL; [19]). *C(H)* is also a model of (i.e., isomorphic to a set of sentences which hold in) an orthomodular lattice (OML; [4], [7]), which is an OL conjoined with the orthomodularity axiom (OMA; see Figure 1). The rationalization of the OMA as a claim proper to physics has proven problematic ([13], Section 5-6), motivating the question of whether the OMA is required in an adequate characterization of QL. Thus formulated, the question suggests that the OMA and its equivalents are specific to an OML, and that as a consequence, banning the OMA from QL yields a "truer" quantum logic.

Lattice axioms

```
      x = c(c(x))                       (AxLat1)
      x v y = y v x                     (AxLat2)
      (x v y) v z = x v (y v z)         (AxLat3)
      (x ^ y) ^ z = x ^ (y ^ z)         (AxLat4)
      x v (x ^ y) = x                   (AxLat5)
      x ^ (x v y) = x                   (AxLat6)
```

Ortholattice axioms

```
      c(x) ^ x = 0                      (AxOL1)
      c(x) v x = 1                      (AxOL2)
      x ^ y = c(c(x) v c(y))            (AxOL3)
```

Orthomodularity axiom

```
      y v (c(y) ^ (x v y)) = x v y      (OMA)
```

Definitions of implications and partial order

```
   i1(x,y) = c(x) v (x ^ y).
   i2(x,y) = i1(c(y), c(x).
   i3(x,y) = (c(x) ^ y) v (c(x) ^ c(y)) v i1(x,y).
   i4(x,y) = i3(c(y), c(x)).
   i5(x,y) = (x ^ y) v (c(x) ^ y) v (c(x) ^ c(y)).
   le(x,y) = (x = (x ^ y)).
```

```
Definitions of indexed intersection
   int1(x,y) = c(i1(x,c(y)))
   int2(x,y) = c(i2(x,c(y)))
   int3(x,y) = c(i3(x,c(y)))
   int4(x,y) = c(i4(x,c(y)))
   int5(x,y) = c(i5(x,c(y)))
```

```
where
   x, y are variables ranging over lattice nodes
   ^ is lattice meet
   v is lattice join
   c(x) is the orthocomplement of x
   i1(x,y) means x →1 y (Sasaki implication)
   i2(x,y) means x →2 y (Dishkant implication)
   i3(x,y) means x →3 y (Kalmbach implication)
   i4(x,y) means x →4 y (non-tollens implication)
   i5(x,y) means x →5 y (relevance implication)
   le(x,y) means x ≤ y
   <-> means if and only if
   =  is equivalence ([12])
   1 is the maximum lattice element (= x v c(x))
   0 is the minimum lattice element (= c(1))
```

Figure 1. Lattice, ortholattice, orthomodularity axioms, and some definitions.

There are at least 21 nominal equivalents (in the sense that ortholattice theory, together with these "equivalents", imply the OMA, and vice versa) of the OMA in quantum logic ([5], Theorem 2.5); as nominal equivalents, they are thus of import to optimizing quantum circuit design. Among these is the Proposition shown in Figure 2:

$$x \cap_i y \quad <-> \quad x \perp y$$

where

$x \cap_i y$ means $c(x \rightarrow_i c(y))$
$x \perp y$ means $le(x,c(y))$
$i = 1,2,3,4,5$

Figure 2. Proposition 2.12 of [5]

2.0 Method

The OML axiomatizations of Megill, Pavičić, and Horner ([5], [14], [15], [16], [21], [22]) were implemented in a *prover9* ([2]) script ([3]) configured to derive the OMA from Proposition 2.12 of [5], for each of $i = 4,5$ together with ortholattice theory (orthomodular lattice theory, without the OMA), then executed in that framework on a Dell Inspiron 545 with an Intel Core2 Quad CPU Q8200 (clocked @ 2.33 GHz) and 8.00 GB RAM, running under the *Windows Vista Home Premium /Cygwin* operating environment.

3.0 Results

Figure 3 shows the proofs, generated by [3] on the platform described in Section 2.0, that Proposition 2.12 of [5] (for each of $i = 4,5$), is implied by orthomodular lattice theory.

```
============================== PROOF =================================

% Proof 1 at 11.87 (+ 0.34) seconds: "OMA".
% Length of proof is 57.
% Level of proof is 13.

1 le(x,y) <-> x = x ^ y # label("Df: less than") # label(non_clause).  [assumption].
2 perp(x,y) <-> le(x,c(y)) # label("Df: perpendicular") # label(non_clause).
[assumption].
4 int4(x,y) = 0 <-> perp(x,y) # label("Hypothesis for Proposition 2.10int4") #
label(non_clause).  [assumption].
5 y v (c(y) ^ (x v y)) = x v y # label("OMA") # label(non_clause) # label(goal).  [goal].
7 -perp(x,y) | le(x,c(y)) # label("Df: perpendicular").  [clausify(2)].
8 int4(x,y) != 0 | perp(x,y) # label("Hypothesis for Proposition 2.10int4").
[clausify(4)].
12 x = c(c(x)) # label("AxL1").  [assumption].
13 c(c(x)) = x.  [copy(12),flip(a)].
14 x v y = y v x # label("AxL2").  [assumption].
15 (x v y) v z = x v (y v z) # label("AxL3").  [assumption].
17 x v (x ^ y) = x # label("AxL5").  [assumption].
18 x ^ (x v y) = x # label("AxL6").  [assumption].
19 c(x) ^ x = 0 # label("AxOL1").  [assumption].
20 c(x) v x = 1 # label("AxOL2").  [assumption].
21 x v c(x) = 1.  [copy(20),rewrite([14(2)])].
22 x ^ y = c(c(x) v c(y)) # label("AxOL3").  [assumption].
30 i4(x,y) = ((c(c(y)) ^ c(x)) v (c(c(y)) ^ c(c(x)))) v (c(c(y)) v (c(y) ^ c(x))) #
label("Df: i4").  [assumption].
31 i4(x,y) = y v (c(y v x) v (c(c(y) v x) v c(c(y) v c(x)))).
[copy(30),rewrite([13(3),22(3),13(4),13(6),13(6),22(5),13(11),22(12),13(11),13(11),14(13)
,15(13)])].
50 int4(x,y) = c(i4(x,c(y))) # label("Df: int4").  [assumption].
51 int4(x,y) = c(c(y) v (c(c(y) v x) v (c(y v x) v c(y v c(x))))).
[copy(50),rewrite([31(3),13(7),13(9)])].
```

```
54 -le(x,y) | x ^ y = x # label("Df: less than").  [clausify(1)].
55 -le(x,y) | c(c(x) v c(y)) = x.  [copy(54),rewrite([22(2)])].
58 c1 v (c(c1) ^ (c2 v c1)) != c2 v c1 # label("OMA") # answer("OMA").  [deny(5)].
59 c1 v c(c1 v c(c1 v c2)) != c1 v c2 # answer("OMA").
[copy(58),rewrite([14(6),22(7),13(4),14(12)])].
60 int4(x,y) != 0 | le(x,c(y)).  [resolve(8,b,7,a)].
61 c(c(x) v (c(c(x) v y) v (c(x v y) v c(x v c(y))))) != 0 | le(y,c(x)).
[copy(60),rewrite([51(1)])].
64 c(1) = 0.  [back_rewrite(19),rewrite([22(2),13(2),21(2)])].
65 c(c(x) v c(x v y)) = x.  [back_rewrite(18),rewrite([22(2)])].
66 x v c(c(x) v c(y)) = x.  [back_rewrite(17),rewrite([22(1)])].
68 x v (y v z) = y v (x v z).  [para(14(a,1),15(a,1,1)),rewrite([15(2)])].
70 c(c(x) v (c(x v y) v (c(c(x) v y) v c(x v c(y))))) != 0 | le(y,c(x)).
[back_rewrite(61),rewrite([68(11)])].
77 x v (y v c(x v y)) = 1.  [para(21(a,1),15(a,1)),flip(a)].
89 c(c(x) v c(y v x)) = x.  [para(14(a,1),65(a,1,1,2,1))].
91 c(0 v c(x)) = x.  [para(21(a,1),65(a,1,1,2,1)),rewrite([64(3),14(3)])].
92 1 v x = 1.  [para(64(a,1),65(a,1,1,1)),rewrite([91(6)])].
97 x v c(c(x) v y) = x.  [para(13(a,1),66(a,1,2,1,2))].
101 x v 0 = x.  [para(21(a,1),66(a,1,2,1)),rewrite([64(2)])].
102 x v x = x.  [para(64(a,1),66(a,1,2,1,2)),rewrite([14(3),91(4)])].
103 x v (y v c(x)) = y v 1.  [para(21(a,1),68(a,1,2)),flip(a)].
112 x v 1 = 1.  [para(92(a,1),14(a,1)),flip(a)].
113 x v (y v c(x)) = 1.  [back_rewrite(103),rewrite([112(5)])].
114 0 v x = x.  [para(101(a,1),14(a,1)),flip(a)].
127 x v (x v y) = x v y.  [para(102(a,1),15(a,1,1)),flip(a)].
129 x v (y v x) = y v x.  [para(102(a,1),15(a,2,2)),rewrite([14(2)])].
142 c(x) v (y v x) = 1.  [para(13(a,1),113(a,1,2,2))].
150 c(x v c(y v x)) != 0 | le(y v x,x).
[para(142(a,1),70(a,1,1,2,1,1)),rewrite([13(2),64(2),13(3),129(3),89(8),14(4),114(5),127(
4),13(9)])].
151 x v c(y v c(x)) = x.  [para(14(a,1),97(a,1,2,1))].
153 x v (y v c(c(x v y) v z)) = x v y.  [para(97(a,1),15(a,1)),flip(a)].
174 x v (y v c(y v x)) = 1.  [para(14(a,1),77(a,1,2,2,1))].
215 c(x) v c(y v x) = c(x).  [para(13(a,1),151(a,1,2,1,2))].
217 x v (y v c(z v c(x v y))) = x v y.  [para(151(a,1),15(a,1)),flip(a)].
724 c(x v (c(y v x) v c(c(y v x) v z))) != 0 | le(y v x,x v c(c(y v x) v z)).
[para(153(a,1),150(a,1,1,2,1)),rewrite([14(8),68(8),153(17)])].
1721 c(x v y) v c(y v c(z v c(x v y))) = c(y v c(z v c(x v y))).
[para(217(a,1),215(a,1,2,1)),rewrite([14(9)])].
19730 le(x v y,y v c(y v c(x v y))).
[para(174(a,1),724(a,1,1)),rewrite([64(2),14(7)]),xx(a)].
19735 x v c(x v c(y v x)) = y v x.  [hyper(55,a,19730,a),rewrite([1721(9),13(7)])].
19744 x v c(x v c(x v y)) = y v x.  [para(14(a,1),19735(a,1,2,1,2,1))].
19808 $F # answer("OMA").  [back_rewrite(59),rewrite([19744(9),14(3)]),xx(a)].

============================== end of proof ==========================



============================== PROOF =================================

% Proof 1 at 46.16 (+ 0.94) seconds: "OMA".
% Length of proof is 54.
% Level of proof is 14.


1 le(x,y) <-> x = x ^ y # label("Df: less than") # label(non_clause).  [assumption].
2 perp(x,y) <-> le(x,c(y)) # label("Df: perpendicular") # label(non_clause).
[assumption].
4 int5(x,y) = 0 <-> perp(x,y) # label("Hypothesis for Proposition 2.10int5") #
label(non_clause).  [assumption].
5 y v (c(y) ^ (x v y)) = x v y # label("OMA") # label(non_clause) # label(goal).  [goal].
7 -perp(x,y) | le(x,c(y)) # label("Df: perpendicular").  [clausify(2)].
8 int5(x,y) != 0 | perp(x,y) # label("Hypothesis for Proposition 2.10int5").
[clausify(4)].
12 x = c(c(x)) # label("AxL1").  [assumption].
13 c(c(x)) = x.  [copy(12),flip(a)].
14 x v y = y v x # label("AxL2").  [assumption].
15 (x v y) v z = x v (y v z) # label("AxL3").  [assumption].
17 x v (x ^ y) = x # label("AxL5").  [assumption].
```

```
18 x ^ (x v y) = x # label("AxL6").  [assumption].
19 c(x) ^ x = 0 # label("AxOL1").  [assumption].
20 c(x) v x = 1 # label("AxOL2").  [assumption].
21 x v c(x) = 1.  [copy(20),rewrite([14(2)])].
22 x ^ y = c(c(x) v c(y)) # label("AxOL3").  [assumption].
32 i5(x,y) = ((x ^ y) v (c(x) ^ y)) v (c(x) ^ c(y)) # label("Df: i5").  [assumption].
33 i5(x,y) = c(x v y) v (c(x v c(y)) v c(c(x) v c(y))).
[copy(32),rewrite([22(2),22(7),13(7),14(9),22(12),13(11),13(11),14(12)])].
52 int5(x,y) = c(i5(x,c(y))) # label("Df: int5").  [assumption].
53 int5(x,y) = c(c(x v c(y)) v (c(x v y) v c(c(x) v y))).
[copy(52),rewrite([33(3),13(6),13(9)])].
54 -le(x,y) | x ^ y = x # label("Df: less than").  [clausify(1)].
55 -le(x,y) | c(c(x) v c(y)) = x.  [copy(54),rewrite([22(2)])].
58 c1 v (c(c1) ^ (c2 v c1)) != c2 v c1 # label("OMA") # answer("OMA").  [deny(5)].
59 c1 v c(c1 v c(c1 v c2)) != c1 v c2 # answer("OMA").
[copy(58),rewrite([14(6),22(7),13(4),14(12)])].
60 int5(x,y) != 0 | le(x,c(y)).  [resolve(8,b,7,a)].
61 c(c(x v c(y)) v (c(x v y) v c(c(x) v y))) != 0 | le(x,c(y)).
[copy(60),rewrite([53(1)])].
64 c(1) = 0.  [back_rewrite(19),rewrite([22(2),13(2),21(2)])].
65 c(c(x) v c(x v y)) = x.  [back_rewrite(18),rewrite([22(2)])].
66 x v c(c(x) v c(y)) = x.  [back_rewrite(17),rewrite([22(1)])].
68 x v (y v z) = y v (x v z).  [para(14(a,1),15(a,1,1)),rewrite([15(2)])].
70 c(c(x v y) v (c(x v c(y)) v c(c(x) v y))) != 0 | le(x,c(y)).
[back_rewrite(61),rewrite([68(10)])].
77 x v (y v c(x v y)) = 1.  [para(21(a,1),15(a,1)),flip(a)].
89 c(c(x) v c(y v x)) = x.  [para(14(a,1),65(a,1,1,2,1))].
91 c(0 v c(x)) = x.  [para(21(a,1),65(a,1,1,2,1)),rewrite([64(3),14(3)])].
92 1 v x = 1.  [para(64(a,1),65(a,1,1,1)),rewrite([91(6)])].
97 x v c(c(x) v y) = x.  [para(13(a,1),66(a,1,2,1,2))].
101 x v 0 = x.  [para(21(a,1),66(a,1,2,1)),rewrite([64(2)])].
102 x v x = x.  [para(64(a,1),66(a,1,2,1,2)),rewrite([14(3),91(4)])].
103 x v (y v c(x)) = y v 1.  [para(21(a,1),68(a,1,2)),flip(a)].
113 x v 1 = 1.  [para(92(a,1),14(a,1)),flip(a)].
114 x v (y v c(x)) = 1.  [back_rewrite(103),rewrite([113(5)])].
115 0 v x = x.  [para(101(a,1),14(a,1)),flip(a)].
129 x v (y v x) = y v x.  [para(102(a,1),15(a,2,2)),rewrite([14(2)])].
142 c(x) v (y v x) = 1.  [para(13(a,1),114(a,1,2,2))].
150 c(x v c(y v x)) != 0 | le(c(x),c(y v x)).
[para(142(a,1),70(a,1,1,1,1)),rewrite([64(2),89(6),13(3),129(3),115(5)])].
151 x v c(y v c(x)) = x.  [para(14(a,1),97(a,1,2,1))].
153 x v (y v c(c(x v y) v z)) = x v y.  [para(97(a,1),15(a,1)),flip(a)].
175 x v (y v c(y v x)) = 1.  [para(14(a,1),77(a,1,2,2,1))].
1036 c(x v (c(y v x) v c(c(y v x) v z))) != 0 | le(c(x v c(c(y v x) v z)),c(y v x)).
[para(153(a,1),150(a,1,1,2,1)),rewrite([14(8),68(8),153(23)])].
27473 le(c(x v c(x v c(y v x))),c(y v x)).
[para(175(a,1),1036(a,1,1)),rewrite([64(2),14(6)]),xx(a)].
27476 c(x v c(x v c(y v x))) = c(y v x).
[hyper(55,a,27473,a),rewrite([13(7),13(8),14(7),68(7),151(6),129(2)]),flip(a)].
27770 x v c(x v c(y v x)) = y v x.
[para(27476(a,1),13(a,1,1)),rewrite([13(3)]),flip(a)].
27822 x v c(x v c(x v y)) = y v x.  [para(14(a,1),27770(a,1,2,1,2,1))].
27860 $F # answer("OMA").  [back_rewrite(59),rewrite([27822(9),14(3)]),xx(a)].

============================== end of proof ==========================
```

Figure 3. Summary of a *prover9* ([2]) proof of Proposition 2.11, for each of i = 4,5. The proofs assume the default inference rules of *prover9*. The general form of a line in this proof is "*line_number conclusion [derivation]*", where *line_number* is a unique identifier of a line in the proof, and *conclusion* is the result of applying the *prover9* inference rules (such as *para*modulation, *copy*ing, and *rewrit*ing), noted in square brackets (denoting the *derivation*), to the lines cited in those brackets. Note that some of "logical" proof lines in the above have been transformed to two text lines, with the *derivation* appearing on a text line following a text line containing the first part of that logical line. The detailed syntax and semantics of these notations can be found in [2]. All *prover9* proofs are by default proofs by contradiction.

The total time to produce the proofs in Figure 3 on the platform described in Section 2.0 was approximately 60 seconds.

4.0 Discussion

The results of Section 3.0 motivate several observations:

1. With the exception of Proposition 2.12, the proofs in Figure 3 use L1, L2, L3, L5, L6, OL1, OL2, and OL3.

2. The proofs in Section 3.0 may be novel.

3. Companion papers provide proofs for i = 1,2,3, and for orthomodular lattice theory implies Propositions 2.12i, i = 1,2,3,4,5, and for the converse propositons.

4. Proposition 2.12 can be regarded as a definition of quantum intersection; thus, this paper together with the papers mentioned in (3), constitute a proof that the definition of quantum intersection is equivalent to the OMA in orthomodular quantum logic. . Companion papers derive equivalences for the OMA with definitions of quantum-identity and quantum-union. Collectively, these papers provide a theory of equivalence of the OMA with the quantum connectives. In light of these equivalences, QL without the OMA would hardly qualify as a logic.

5.0 Acknowledgements

This work benefited from discussions with Tom Oberdan, Frank Pecchioni, Tony Pawlicki, and the late John K. Prentice, whose passion for foundations of physics inspired those of us privileged to have known him. For any infelicities that remain, I am solely responsible.

6.0 References

[1] von Neumann J. *Mathematical Foundations of Quantum Mechanics*. 1936. Translated by R. T. Beyer. Princeton. 1983.
[2] McCune WW. *prover9 and mace4.* URL http://www.cs.unm.edu/~mccune/prover9/. 2009.
[3] Horner JK. MHT *prover9* scripts. 2011. Available from the author on request.
[4] Dalla Chiara ML and Giuntini R. *Quantum Logics*. URL http://xxx.lanl.gov/abs/quant-ph/0101028. 2004.
[5] Megill ND and Pavičić M. Orthomodular lattices and quantum algebra. *International Journal of Theoretical Physics* 40 (2001), pp. 1387-1410.
[6] Akhiezer NI and Glazman IM. *Theory of Linear Operators in Hilbert Space. Volume I.* Translated by M. Nestell. Frederick Ungar. 1961.
[7] Holland, Jr. SS Orthomodularity in infinite dimensions: a theorem of M. Solèr. *Bulletin of the American Mathematical Society* 32 (1995), pp. 205-234.
[8] Marsden EL and Herman LM. *A condition for distribution in orthomodular lattices.* Kansas State University Technical Report #40. 1974.
[9] Knuth DE and Bendix PB. Simple word problems in universal algebras. In J. Leech, ed. *Computational Problems in Abstract Algebra.* Pergamon Press. 1970. pp. 263-297.
[10] Chang CC and Keisler HJ. *Model Theory.* North-Holland. 1990. pp. 38-39.
[11] Birkhoff G. *Lattice Theory.* Third Edition. American Mathematical Society. 1967.
[12] Church A. *Introduction to Mathematical Logic. Volume I.* Princeton. 1956.

[13] Jauch J. *Foundations of Quantum Mechanics.* Addison-Wesley. 1968.
[14] Megill ND. *Metamath.* URL http://us.metamath.org/qlegif/mmql.html#unify. 2004.
[15] Horner JK. An automated deduction system for orthomodular lattice theory. *Proceedings of the 2005 International Conference on Artificial Intelligence.* CSREA Press. 2005. pp. 260-265.
[16] Horner JK. An automated equational logic deduction of join elimination in orthomodular lattice theory. *Proceedings of the 2007 International Conference on Artificial Intelligence.* CSREA Press. 2007. pp. 481-488.
[17] Messiah A. *Quantum Mechanics*. Dover. 1958.
[18] Horner JK. Using automated theorem-provers to aid the design of efficient compilers for quantum computing. *Los Alamos National Laboratory Quantum Institute Workshop.* December 9–10, 2002. URL http://www.lanl.gov/science/centers/quantum/qls_pdfs/horner.pdf.
[19] Birkhoff G and von Neumann J. The logic of quantum mechanics. *Annals of Mathematics* 37 (1936), 823-243.
[20] Nielsen MA and Chuang L . *Quantum Computation and Quantum Information*. Cambridge. 2000.
[21] Pavičić M and Megill N. Quantum and classical implicational algebras with primitive implication. *International Journal of Theoretical Physics* 37 (1998), 2091-2098. ftp://m3k.grad.hr/pavicic/quantum-logic/1998-int-j-theor-phys-2.ps.gz.
[22] Horner JK. An automated deduction of the relative strength of orthomodular and weakly orthomodular lattice theory. *Proceedings of the 2009 International Conference on Artificial Intelligence*. CSREA Press. 2009. pp. 525-530.

Quantum-Intersection Equivalents of the Orthomodularity Law in Quantum Logic: Part 3

Jack K. Horner
P. O. Box 266
Los Alamos, New Mexico 87544 USA
jhorner@cybermesa.com

FCS 2014

Abstract

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of quantum-mechanical behaviors and operations. In much the same way that the structure of conventional propositional (Boolean) logic (BL) is the logic of the description of the behavior of classical physical systems and is isomorphic to a Boolean algebra (BA), so also the algebra, C(H), of closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space is a logic of the descriptions of the behavior of quantum mechanical systems and is a model of an ortholattice (OL). An OL can thus be thought of as a kind of "quantum logic" (QL). C(H) is also a model of an orthomodular lattice, which is an OL conjoined with the orthomodularity axiom (OMA). The rationalization of the OMA as a claim proper to physics has proven problematic, motivating the question of whether the OMA and its equivalents are required in an adequate characterization of QL. Here I provide an automated deduction of two quantum-intersection-based equivalents of the OMA. The proofs may be novel.

Keywords: automated deduction, quantum computing, orthomodular lattice, Hilbert space

1.0 Introduction

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of the description of quantum-mechanical behaviors ([1], [17], [18], [20]). In much the same way that conventional propositional (Boolean) logic (BL,[12]) is the logical structure of description of the behavior of classical physical systems (e.g. "the measurements of the position and momentum of particle P are commutative", i.e., can be measured in either other, yielding the same results) and is isomorphic to a Boolean lattice ([10], [11], [19]), so also the algebra, *C(H)*, of the closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space *H* ([1], [4], [6], [9], [13]) is *a* logic of the descriptions of the behavior of quantum mechanical systems (e.g., "the measurements of the position and momentum of particle P are *not* commutative") and is a model ([10]) of an ortholattice (OL; [4]). An OL can thus be thought of as a kind of "quantum logic" (QL; [19]). *C(H)* is also a model of (i.e., isomorphic to a set of sentences which hold in) an orthomodular lattice (OML; [4], [7]), which is an OL conjoined with the orthomodularity axiom (OMA; see Figure 1). The rationalization of the OMA as a claim proper to physics has proven problematic ([13], Section 5-6), motivating the question of whether the OMA is required in an adequate characterization of QL. Thus formulated, the question suggests that the OMA and its equivalents are specific to an OML, and that as a consequence, banning the OMA from QL yields a "truer" quantum logic.

Lattice axioms

```
      x = c(c(x))                       (AxLat1)
      x v y = y v x                     (AxLat2)
      (x v y) v z = x v (y v z)          (AxLat3)
      (x ^ y) ^ z = x ^ (y ^ z)          (AxLat4)
      x v (x ^ y) = x                   (AxLat5)
      x ^ (x v y) = x                   (AxLat6)
```

Ortholattice axioms

```
      c(x) ^ x = 0                      (AxOL1)
      c(x) v x = 1                      (AxOL2)
      x ^ y = c(c(x) v c(y))            (AxOL3)
```

Orthomodularity axiom

```
      y v (c(y) ^ (x v y)) = x v y      (OMA)
```

Definitions of implications and partial order

```
   i1(x,y) = c(x) v (x ^ y).
   i2(x,y) = i1(c(y), c(x).
   i3(x,y) = (c(x) ^ y) v (c(x) ^ c(y)) v i1(x,y).
   i4(x,y) = i3(c(y), c(x)).
   i5(x,y) = (x ^ y) v (c(x) ^ y) v (c(x) ^ c(y)).
   le(x,y) = (x = (x ^ y)).
```

```
Definitions of indexed intersection
   int1(x,y) = c(i1(x,c(y)))
   int2(x,y) = c(i2(x,c(y)))
   int3(x,y) = c(i3(x,c(y)))
   int4(x,y) = c(i4(x,c(y)))
   int5(x,y) = c(i5(x,c(y)))
```

```
where
   x, y are variables ranging over lattice nodes
   ^ is lattice meet
   v is lattice join
   c(x) is the orthocomplement of x
   i1(x,y) means x →1 y (Sasaki implication)
   i2(x,y) means x →2 y (Dishkant implication)
   i3(x,y) means x →3 y (Kalmbach implication)
   i4(x,y) means x →4 y (non-tollens implication)
   i5(x,y) means x →5 y (relevance implication)
   le(x,y) means x ≤ y
   <-> means if and only if
   =  is equivalence ([12])
   1 is the maximum lattice element (= x v c(x))
   0 is the minimum lattice element (= c(1))
```

Figure 1. Lattice, ortholattice, orthomodularity axioms, and some definitions.

There are at least 21 nominal equivalents (in the sense that ortholattice theory, together with these "equivalents", imply the OMA, and vice versa) of the OMA in quantum logic ([5], Theorem 2.5); as nominal equivalents, they are thus of import to optimizing quantum circuit design. Among these is the Proposition shown in Figure 2:

$$x \cap_i y \quad <-> \quad x \perp y$$

where

$x \cap_i y$ means $c(x \rightarrow_i c(y))$
$x \perp y$ means $le(x, c(y))$
$i = 1,2,3,4,5$

Figure 2. Proposition 2.12 of [5]

2.0 Method

The OML axiomatizations of Megill, Pavičić, and Horner ([5], [14], [15], [16], [21], [22]) were implemented in a *prover9* ([2]) script ([3]) configured to derive Proposition 2.12 of [5]from orthomodular lattice theory, for each of i = 1,2, then executed in that framework on a Dell Inspiron 545 with an Intel Core2 Quad CPU Q8200 (clocked @ 2.33 GHz) and 8.00 GB RAM, running under the *Windows Vista Home Premium /Cygwin* operating environment.

3.0 Results

Figure 3 shows the proofs, generated by [3] on the platform described in Section 2.0, that orthomodular lattice theory implies Proposition 2.12 (for each of i = 1,2.

```
============================== PROOF =================================

% Proof 1 at 7.46 (+ 0.14) seconds.
% Length of proof is 55.
% Level of proof is 10.

1 le(x,y) <-> x = x ^ y # label("Df: less than") # label(non_clause).  [assumption].
2 perp(x,y) <-> le(x,c(y)) # label("Df: perpendicular") # label(non_clause).
[assumption].
4 int1(x,y) = 0 <-> perp(x,y) # label("Proposition 2.10int1") # label(non_clause) #
label(goal).  [goal].
7 x = c(c(x)) # label("AxL1").  [assumption].
8 c(c(x)) = x.  [copy(7),flip(a)].
9 x v y = y v x # label("AxL2").  [assumption].
10 (x v y) v z = x v (y v z) # label("AxL3").  [assumption].
12 x v (x ^ y) = x # label("AxL5").  [assumption].
13 x ^ (x v y) = x # label("AxL6").  [assumption].
14 c(x) ^ x = 0 # label("AxOL1").  [assumption].
15 c(x) v x = 1 # label("AxOL2").  [assumption].
16 x v c(x) = 1.  [copy(15),rewrite([9(2)])].
17 x ^ y = c(c(x) v c(y)) # label("AxOL3").  [assumption].
18 x v (c(x) ^ (y v x)) = y v x # label("OMA").  [assumption].
19 x v c(x v c(y v x)) = y v x.  [copy(18),rewrite([17(3),8(2)])].
21 i1(x,y) = c(x) v (x ^ y) # label("Df: i1").  [assumption].
22 i1(x,y) = c(x) v c(c(x) v c(y)).  [copy(21),rewrite([17(3)])].
29 i5(x,y) = ((x ^ y) v (c(x) ^ y)) v (c(x) ^ c(y)) # label("Df: i5").  [assumption].
30 i5(x,y) = c(x v y) v (c(x v c(y)) v c(c(x) v c(y))).
[copy(29),rewrite([17(2),17(7),8(7),9(9),17(12),8(11),8(11),9(12)])].
41 int1(x,y) = c(i1(x,c(y))) # label("Df: int1").  [assumption].
42 int1(x,y) = c(c(x) v c(c(x) v y)).  [copy(41),rewrite([22(3),8(5)])].
51 -le(x,y) | x ^ y = x # label("Df: less than").  [clausify(1)].
52 -le(x,y) | c(c(x) v c(y)) = x.  [copy(51),rewrite([17(2)])].
53 le(x,y) | x ^ y != x # label("Df: less than").  [clausify(1)].
54 le(x,y) | c(c(x) v c(y)) != x.  [copy(53),rewrite([17(2)])].
55 -perp(x,y) | le(x,c(y)) # label("Df: perpendicular").  [clausify(2)].
56 perp(x,y) | -le(x,c(y)) # label("Df: perpendicular").  [clausify(2)].
```

```
57 int1(c1,c2) = 0 | perp(c1,c2) # label("Proposition 2.10int1").  [deny(4)].
58 c(c(c1) v c(c2 v c(c1))) = 0 | perp(c1,c2).  [copy(57),rewrite([42(3),9(6)])].
59 int1(c1,c2) != 0 | -perp(c1,c2) # label("Proposition 2.10int1").  [deny(4)].
60 c(c(c1) v c(c2 v c(c1))) != 0 | -perp(c1,c2).  [copy(59),rewrite([42(3),9(6)])].
61 c(1) = 0.  [back_rewrite(14),rewrite([17(2),8(2),16(2)])].
62 c(c(x) v c(x v y)) = x.  [back_rewrite(13),rewrite([17(2)])].
63 x v c(c(x) v c(y)) = x.  [back_rewrite(12),rewrite([17(1)])].
65 x v (y v z) = y v (x v z).  [para(9(a,1),10(a,1,1)),rewrite([10(2)])].
80 le(x,c(y)) | c(c(x) v y) != x.  [para(8(a,1),54(b,1,1,2))].
85 c(c(c1) v c(c2 v c(c1))) = 0 | le(c1,c(c2)).  [resolve(58,b,55,a)].
92 c(x) v c(x v y) = c(x).  [para(62(a,1),8(a,1,1)),flip(a)].
96 c(0 v c(x)) = x.  [para(16(a,1),62(a,1,1,2,1)),rewrite([61(3),9(3)])].
97 c(x v y) v c(x v c(x v y)) = c(x).
[para(62(a,1),19(a,1,2,1,2)),rewrite([9(5),92(11)])].
98 1 v x = 1.  [para(61(a,1),62(a,1,1,1)),rewrite([96(6)])].
107 x v 0 = x.  [para(16(a,1),63(a,1,2,1)),rewrite([61(2)])].
109 x v x = x.  [para(61(a,1),63(a,1,2,1,2)),rewrite([9(3),96(4)])].
110 x v (y v c(x)) = y v 1.  [para(16(a,1),65(a,1,2)),flip(a)].
121 x v 1 = 1.  [para(98(a,1),9(a,1)),flip(a)].
122 x v (y v c(x)) = 1.  [back_rewrite(110),rewrite([121(5)])].
123 0 v x = x.  [para(107(a,1),9(a,1)),flip(a)].
129 x v (y v x) = y v x.  [para(109(a,1),10(a,2,2)),rewrite([9(2)])].
315 le(x,c(y)) | c(y v c(x)) != x.  [para(9(a,1),80(b,1,1))].
407 c(c(c1) v c(c2 v c(c1))) = 0 | c(c2 v c(c1)) = c1.
[resolve(85,b,52,a),rewrite([8(16),9(15)])].
454 c(x v y) v c(y v c(x v y)) = c(y).  [para(19(a,1),97(a,1,1,1)),rewrite([19(7)])].
19987 c(c2 v c(c1)) = c1.
[para(407(a,1),30(a,2,1)),rewrite([30(15),8(24),129(23),8(24),8(28),122(27),61(23),9(23),
123(23),9(22),454(22),8(10),8(17),129(16),8(17),8(21),122(20),61(16),9(16),123(16),123(15
)]),flip(b),merge(b)].
19988 -perp(c1,c2).  [back_rewrite(60),rewrite([19987(7),9(4),16(4),61(2)]),xx(a)].
19989 -le(c1,c(c2)).  [ur(56,a,19988,a)].
19990 $F.  [ur(315,a,19989,a),rewrite([19987(5)]),xx(a)].

============================== end of proof ==========================



============================== PROOF =================================

% Proof 1 at 33.74 (+ 0.87) seconds: "OMA".
% Length of proof is 50.
% Level of proof is 12.

1 le(x,y) <-> x = x ^ y # label("Df: less than") # label(non_clause).  [assumption].
2 perp(x,y) <-> le(x,c(y)) # label("Df: perpendicular") # label(non_clause).
[assumption].
4 int2(x,y) = 0 <-> perp(x,y) # label("Hypothesis for Proposition 2.10int2") #
label(non_clause).  [assumption].
5 y v (c(y) ^ (x v y)) = x v y # label("OMA") # label(non_clause) # label(goal).  [goal].
7 -perp(x,y) | le(x,c(y)) # label("Df: perpendicular").  [clausify(2)].
8 int2(x,y) != 0 | perp(x,y) # label("Hypothesis for Proposition 2.10int2").
[clausify(4)].
12 x = c(c(x)) # label("AxL1").  [assumption].
13 c(c(x)) = x.  [copy(12),flip(a)].
14 x v y = y v x # label("AxL2").  [assumption].
15 (x v y) v z = x v (y v z) # label("AxL3").  [assumption].
17 x v (x ^ y) = x # label("AxL5").  [assumption].
18 x ^ (x v y) = x # label("AxL6").  [assumption].
19 c(x) ^ x = 0 # label("AxOL1").  [assumption].
20 c(x) v x = 1 # label("AxOL2").  [assumption].
21 x v c(x) = 1.  [copy(20),rewrite([14(2)])].
22 x ^ y = c(c(x) v c(y)) # label("AxOL3").  [assumption].
26 i2(x,y) = c(c(y)) v (c(y) ^ c(x)) # label("Df: i2").  [assumption].
27 i2(x,y) = y v c(y v x).  [copy(26),rewrite([13(3),22(4),13(3),13(3)])].
46 int2(x,y) = c(i2(x,c(y))) # label("Df: int2").  [assumption].
47 int2(x,y) = c(c(y) v c(c(y) v x)).  [copy(46),rewrite([27(3)])].
54 -le(x,y) | x ^ y = x # label("Df: less than").  [clausify(1)].
55 -le(x,y) | c(c(x) v c(y)) = x.  [copy(54),rewrite([22(2)])].
58 c1 v (c(c1) ^ (c2 v c1)) != c2 v c1 # label("OMA") # answer("OMA").  [deny(5)].
```

```
59 c1 v c(c1 v c(c1 v c2)) != c1 v c2 # answer("OMA").
[copy(58),rewrite([14(6),22(7),13(4),14(12)])].
60 int2(x,y) != 0 | le(x,c(y)).  [resolve(8,b,7,a)].
61 c(c(x) v c(c(x) v y)) != 0 | le(y,c(x)).  [copy(60),rewrite([47(1)])].
64 c(1) = 0.  [back_rewrite(19),rewrite([22(2),13(2),21(2)])].
65 c(c(x) v c(x v y)) = x.  [back_rewrite(18),rewrite([22(2)])].
66 x v c(c(x) v c(y)) = x.  [back_rewrite(17),rewrite([22(1)])].
68 x v (y v z) = y v (x v z).  [para(14(a,1),15(a,1,1)),rewrite([15(2)])].
75 x v (y v c(x v y)) = 1.  [para(21(a,1),15(a,1)),flip(a)].
80 c(x v c(x v y)) != 0 | le(y,x).  [para(13(a,1),61(a,1,1,1)),rewrite([13(2),13(8)])].
93 c(0 v c(x)) = x.  [para(21(a,1),65(a,1,1,2,1)),rewrite([64(3),14(3)])].
99 x v c(c(x) v y) = x.  [para(13(a,1),66(a,1,2,1,2))].
104 x v x = x.  [para(64(a,1),66(a,1,2,1,2)),rewrite([14(3),93(4)])].
116 x v (x v y) = x v y.  [para(104(a,1),15(a,1,1)),flip(a)].
135 x v c(y v c(x)) = x.  [para(14(a,1),99(a,1,2,1))].
137 x v (y v c(c(x v y) v z)) = x v y.  [para(99(a,1),15(a,1)),flip(a)].
150 x v (y v c(y v x)) = 1.  [para(14(a,1),75(a,1,2,2,1))].
196 c(x) v c(y v x) = c(x).  [para(13(a,1),135(a,1,2,1,2))].
198 x v (y v c(z v c(x v y))) = x v y.  [para(135(a,1),15(a,1)),flip(a)].
217 c(x v c(y v x)) != 0 | le(y,x).  [para(14(a,1),80(a,1,1,2,1))].
322 c(x v y) v c(x v (z v y)) = c(x v y).  [para(68(a,1),196(a,1,2,1))].
1123 c(x v (c(y v x) v c(c(y v x) v z))) != 0 | le(y,x v c(c(y v x) v z)).
[para(137(a,1),217(a,1,1,2,1)),rewrite([14(8),68(8)])].
3899 c(x v y) v c(x v c(z v c(x v y))) = c(x v c(z v c(x v y))).
[para(198(a,1),322(a,1,2,1)),rewrite([14(9)])].
23785 le(x,y v c(y v c(x v y))).
[para(150(a,1),1123(a,1,1)),rewrite([64(2),14(6)]),xx(a)].
23791 le(x,y v c(y v c(y v x))).  [para(14(a,1),23785(a,2,2,1,2,1))].
23802 le(x v y,x v c(x v c(x v y))).  [para(116(a,1),23791(a,2,2,1,2,1))].
23810 x v c(x v c(x v y)) = x v y.  [hyper(55,a,23802,a),rewrite([3899(9),13(7)])].
23811 $F # answer("OMA").  [resolve(23810,a,59,a)].

============================== end of proof ==========================
```

Figure 3. Summary of a *prover9* ([2]) proof of Proposition 2.12, for each of i = 1,2 from orthomodular lattice theory. The proofs assume the default inference rules of *prover9*. The general form of a line in this proof is "*line_number conclusion [derivation]*", where *line_number* is a unique identifier of a line in the proof, and *conclusion* is the result of applying the *prover9* inference rules (such as *para*modulation, *copy*ing, and *rewrit*ing), noted in square brackets (denoting the *derivation*), to the lines cited in those brackets. Note that some of "logical" proof lines in the above have been transformed to two text lines, with the *derivation* appearing on a text line following a text line containing the first part of that logical line. The detailed syntax and semantics of these notations can be found in [2]. All *prover9* proofs are by default proofs by contradiction.

The total time to produce the proofs in Figure 3 on the platform described in Section 2.0 was approximately 40 seconds.

4.0 Discussion

The results of Section 3.0 motivate several observations:

1. Both proofs in Figure 3 use L1, L2, L3, L5, L6, OL1, OL2, and OL3.

2. The proofs in Section 3.0 may be novel.

3. Companion papers provide proofs for i = 3,4,5, and for the converse propositions.

4. Proposition 2.12 can be regarded as a definition of quantum intersection; thus, this paper together with the papers mentioned in (3), constitute a proof that the definition of quantum intersection is equivalent to the OMA in orthomodular

quantum logic. . Companion papers derive equivalences for the OMA with definitions of quantum-identity and quantum-union. Collectively, these papers provide a theory of equivalence of the OMA with the quantum connectives. In light of these equivalences, QL without the OMA would hardly qualify as a logic.

5.0 Acknowledgements

This work benefited from discussions with Tom Oberdan, Frank Pecchioni, Tony Pawlicki, and the late John K. Prentice, whose passion for foundations of physics inspired those of us privileged to have known him. For any infelicities that remain, I am solely responsible.

6.0 References

[1] von Neumann J. *Mathematical Foundations of Quantum Mechanics*. 1936. Translated by R. T. Beyer. Princeton. 1983.

[2] McCune WW. *prover9 and mace4.* URL http://www.cs.unm.edu/~mccune/prover9/. 2009.

[3] Horner JK. MHT *prover9* scripts. 2011. Available from the author on request.

[4] Dalla Chiara ML and Giuntini R. *Quantum Logics*. URL http://xxx.lanl.gov/abs/quant-ph/0101028. 2004.

[5] Megill ND and Pavičić M. Orthomodular lattices and quantum algebra. *International Journal of Theoretical Physics* 40 (2001), pp. 1387-1410.

[6] Akhiezer NI and Glazman IM. *Theory of Linear Operators in Hilbert Space. Volume I.* Translated by M. Nestell. Frederick Ungar. 1961.

[7] Holland, Jr. SS Orthomodularity in infinite dimensions: a theorem of M. Solèr. *Bulletin of the American Mathematical Society* 32 (1995), pp. 205-234.

[8] Marsden EL and Herman LM. *A condition for distribution in orthomodular lattices.* Kansas State University Technical Report #40. 1974.

[9] Knuth DE and Bendix PB. Simple word problems in universal algebras. In J. Leech, ed. *Computational Problems in Abstract Algebra.* Pergamon Press. 1970. pp. 263-297.

[10] Chang CC and Keisler HJ. *Model Theory*. North-Holland. 1990. pp. 38-39.

[11] Birkhoff G. *Lattice Theory.* Third Edition. American Mathematical Society. 1967.

[12] Church A. *Introduction to Mathematical Logic. Volume I.* Princeton. 1956.

[13] Jauch J. *Foundations of Quantum Mechanics.* Addison-Wesley. 1968.

[14] Megill ND. *Metamath.* URL http://us.metamath.org/qlegif/mmql.html#unify. 2004.

[15] Horner JK. An automated deduction system for orthomodular lattice theory. *Proceedings of the 2005 International Conference on Artificial Intelligence.* CSREA Press. 2005. pp. 260-265.

[16] Horner JK. An automated equational logic deduction of join elimination in orthomodular lattice theory. *Proceedings of the 2007 International Conference on Artificial Intelligence.* CSREA Press. 2007. pp. 481-488.

[17] Messiah A. *Quantum Mechanics*. Dover. 1958.
[18] Horner JK. Using automated theorem-provers to aid the design of efficient compilers for quantum computing. *Los Alamos National Laboratory Quantum Institute Workshop*. December 9–10, 2002. URL http://www.lanl.gov/science/centers/quantum/qls_pdfs/horner.pdf.

[19] Birkhoff G and von Neumann J. The logic of quantum mechanics. *Annals of Mathematics* 37 (1936), 823-243.

[20] Nielsen MA and Chuang L . *Quantum Computation and Quantum Information*. Cambridge. 2000.

[21] Pavičić M and Megill N. Quantum and classical implicational algebras with primitive implication. *International Journal of Theoretical Physics* 37 (1998), 2091-2098. ftp://m3k.grad.hr/pavicic/quantum-logic/1998-int-j-theor-phys-2.ps.gz.

[22] Horner JK. An automated deduction of the relative strength of orthomodular and weakly orthomodular lattice theory. *Proceedings of the 2009 International Conference on Artificial Intelligence*. CSREA Press. 2009. pp. 525-530.

Quantum-Intersection Equivalents of the Orthomodularity Law in Quantum Logic: Part 4

Jack K. Horner
P. O. Box 266
Los Alamos, New Mexico 87544 USA
jhorner@cybermesa.com

FCS 2014

Abstract

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of quantum-mechanical behaviors and operations. In much the same way that the structure of conventional propositional (Boolean) logic (BL) is the logic of the description of the behavior of classical physical systems and is isomorphic to a Boolean algebra (BA), so also the algebra, C(H), of closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space is a logic of the descriptions of the behavior of quantum mechanical systems and is a model of an ortholattice (OL). An OL can thus be thought of as a kind of "quantum logic" (QL). C(H) is also a model of an orthomodular lattice, which is an OL conjoined with the orthomodularity axiom (OMA). The rationalization of the OMA as a claim proper to physics has proven problematic, motivating the question of whether the OMA and its equivalents are required in an adequate characterization of QL. Here I provide an automated deduction of two quantum-intersection-based equivalents of the OMA. The proofs may be novel.

Keywords: automated deduction, quantum computing, orthomodular lattice, Hilbert space

1.0 Introduction

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of the description of quantum-mechanical behaviors ([1], [17], [18], [20]). In much the same way that conventional propositional (Boolean) logic (BL,[12]) is the logical structure of description of the behavior of classical physical systems (e.g. "the measurements of the position and momentum of particle P are commutative", i.e., can be measured in either other, yielding the same results) and is isomorphic to a Boolean lattice ([10], [11], [19]), so also the algebra, *C(H)*, of the closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space *H* ([1], [4], [6], [9], [13]) is *a* logic of the descriptions of the behavior of quantum mechanical systems (e.g., "the measurements of the position and momentum of particle P are *not* commutative") and is a model ([10]) of an ortholattice (OL; [4]). An OL can thus be thought of as a kind of "quantum logic" (QL; [19]). *C(H)* is also a model of (i.e., isomorphic to a set of sentences which hold in) an orthomodular lattice (OML; [4], [7]), which is an OL conjoined with the orthomodularity axiom (OMA; see Figure 1). The rationalization of the OMA as a claim proper to physics has proven problematic ([13], Section 5-6), motivating the question of whether the OMA is required in an adequate characterization of QL. Thus formulated, the question suggests that the OMA and its equivalents are specific to an OML, and that as a consequence, banning the OMA from QL yields a "truer" quantum logic.

Lattice axioms

```
      x = c(c(x))                       (AxLat1)
      x v y = y v x                     (AxLat2)
      (x v y) v z = x v (y v z)         (AxLat3)
      (x ^ y) ^ z = x ^ (y ^ z)         (AxLat4)
      x v (x ^ y) = x                   (AxLat5)
      x ^ (x v y) = x                   (AxLat6)
```

Ortholattice axioms

```
      c(x) ^ x = 0                      (AxOL1)
      c(x) v x = 1                      (AxOL2)
      x ^ y = c(c(x) v c(y))            (AxOL3)
```

Orthomodularity axiom

```
      y v (c(y) ^ (x v y)) = x v y      (OMA)
```

Definitions of implications and partial order

```
   i1(x,y) = c(x) v (x ^ y).
   i2(x,y) = i1(c(y), c(x).
   i3(x,y) = (c(x) ^ y) v (c(x) ^ c(y)) v i1(x,y).
   i4(x,y) = i3(c(y), c(x)).
   i5(x,y) = (x ^ y) v (c(x) ^ y) v (c(x) ^ c(y)).
   le(x,y) = (x = (x ^ y)).
```

```
Definitions of indexed intersection
   int1(x,y) = c(i1(x,c(y)))
   int2(x,y) = c(i2(x,c(y)))
   int3(x,y) = c(i3(x,c(y)))
   int4(x,y) = c(i4(x,c(y)))
   int5(x,y) = c(i5(x,c(y)))
```

```
where
   x, y are variables ranging over lattice nodes
   ^ is lattice meet
   v is lattice join
   c(x) is the orthocomplement of x
   i1(x,y) means x →1 y (Sasaki implication)
   i2(x,y) means x →2 y (Dishkant implication)
   i3(x,y) means x →3 y (Kalmbach implication)
   i4(x,y) means x →4 y (non-tollens implication)
   i5(x,y) means x →5 y (relevance implication)
   le(x,y) means x ≤ y
   <-> means if and only if
   =  is equivalence ([12])
   1 is the maximum lattice element (= x v c(x))
   0 is the minimum lattice element (= c(1))
```

Figure 1. Lattice, ortholattice, orthomodularity axioms, and some definitions.

There are at least 21 nominal equivalents (in the sense that ortholattice theory, together with these "equivalents", imply the OMA, and vice versa) of the OMA in quantum logic ([5], Theorem 2.5); as nominal equivalents, they are thus of import to optimizing quantum circuit design. Among these is the Proposition shown in Figure 2:

$$x \cap_i y \quad <-> \quad x \perp y$$

where

$x \cap_i y$ means $c(x \rightarrow_i c(y))$
$x \perp y$ means le(x,c(y))
i = 1,2,3,4,5

Figure 2. Proposition 2.12 of [5]

2.0 Method

The OML axiomatizations of Megill, Pavičić, and Horner ([5], [14], [15], [16], [21], [22]) were implemented in a *prover9* ([2]) script ([3]) configured to derive Proposition 2.12 of [5] from orthomoldular lattice theory, for each of i = 3,4, then executed in that framework on a Dell Inspiron 545 with an Intel Core2 Quad CPU Q8200 (clocked @ 2.33 GHz) and 8.00 GB RAM, running under the *Windows Vista Home Premium /Cygwin* operating environment.

3.0 Results

Figure 3 shows the proofs, generated by [3] on the platform described in Section 2.0, that orthomodular lattice theory implies Proposition 2.12 (for each of i = 3,4.

```
============================== PROOF =================================

% Proof 1 at 1037.25 (+ 19.27) seconds.
% Length of proof is 61.
% Level of proof is 12.

1 le(x,y) <-> x = x ^ y # label("Df: less than") # label(non_clause).  [assumption].
2 perp(x,y) <-> le(x,c(y)) # label("Df: perpendicular") # label(non_clause).
[assumption].
4 int3(x,y) = 0 <-> perp(x,y) # label("Proposition 2.10int3") # label(non_clause) #
label(goal).  [goal].
7 x = c(c(x)) # label("AxL1").  [assumption].
8 c(c(x)) = x.  [copy(7),flip(a)].
9 x v y = y v x # label("AxL2").  [assumption].
10 (x v y) v z = x v (y v z) # label("AxL3").  [assumption].
12 x v (x ^ y) = x # label("AxL5").  [assumption].
13 x ^ (x v y) = x # label("AxL6").  [assumption].
14 c(x) ^ x = 0 # label("AxOL1").  [assumption].
15 c(x) v x = 1 # label("AxOL2").  [assumption].
16 x v c(x) = 1.  [copy(15),rewrite([9(2)])].
17 x ^ y = c(c(x) v c(y)) # label("AxOL3").  [assumption].
18 x v (c(x) ^ (y v x)) = y v x # label("OMA").  [assumption].
19 x v c(x v c(y v x)) = y v x.  [copy(18),rewrite([17(3),8(2)])].
25 i3(x,y) = ((c(x) ^ y) v (c(x) ^ c(y))) v (c(x) v (x ^ y)) # label("Df: i3").
[assumption].
26 i3(x,y) = c(x v y) v (c(x v c(y)) v (c(x) v c(c(x) v c(y)))).
[copy(25),rewrite([17(3),8(3),17(7),8(6),8(6),9(7),17(9),10(14)])].
29 i5(x,y) = ((x ^ y) v (c(x) ^ y)) v (c(x) ^ c(y)) # label("Df: i5").  [assumption].
30 i5(x,y) = c(x v y) v (c(x v c(y)) v c(c(x) v c(y))).
[copy(29),rewrite([17(2),17(7),8(7),9(9),17(12),8(11),8(11),9(12)])].
45 int3(x,y) = c(i3(x,c(y))) # label("Df: int3").  [assumption].
46 int3(x,y) = c(c(x v c(y)) v (c(x v y) v (c(x) v c(c(x) v y)))).
[copy(45),rewrite([26(3),8(6),8(10)])].
51 -le(x,y) | x ^ y = x # label("Df: less than").  [clausify(1)].
52 -le(x,y) | c(c(x) v c(y)) = x.  [copy(51),rewrite([17(2)])].
53 le(x,y) | x ^ y != x # label("Df: less than").  [clausify(1)].
```

```
54 le(x,y) | c(c(x) v c(y)) != x.  [copy(53),rewrite([17(2)])].
55 -perp(x,y) | le(x,c(y)) # label("Df: perpendicular").  [clausify(2)].
56 perp(x,y) | -le(x,c(y)) # label("Df: perpendicular").  [clausify(2)].
57 int3(c1,c2) = 0 | perp(c1,c2) # label("Proposition 2.10int3").  [deny(4)].
58 c(c(c1 v c2) v (c(c1) v (c(c2 v c(c1)) v c(c1 v c(c2))))) = 0 | perp(c1,c2).
[copy(57),rewrite([46(3),9(15),9(19),10(19),10(18)])].
59 int3(c1,c2) != 0 | -perp(c1,c2) # label("Proposition 2.10int3").  [deny(4)].
60 c(c(c1 v c2) v (c(c1) v (c(c2 v c(c1)) v c(c1 v c(c2))))) != 0 | -perp(c1,c2).
[copy(59),rewrite([46(3),9(15),9(19),10(19),10(18)])].
61 c(1) = 0.  [back_rewrite(14),rewrite([17(2),8(2),16(2)])].
62 c(c(x) v c(x v y)) = x.  [back_rewrite(13),rewrite([17(2)])].
63 x v c(c(x) v c(y)) = x.  [back_rewrite(12),rewrite([17(1)])].
65 x v (y v z) = y v (x v z).  [para(9(a,1),10(a,1,1)),rewrite([10(2)])].
66 c(c(c1) v (c(c1 v c2) v (c(c2 v c(c1)) v c(c1 v c(c2))))) != 0 | -perp(c1,c2).
[back_rewrite(60),rewrite([65(19)])].
67 c(c(c1) v (c(c1 v c2) v (c(c2 v c(c1)) v c(c1 v c(c2))))) = 0 | perp(c1,c2).
[back_rewrite(58),rewrite([65(19)])].
82 le(x,c(y)) | c(c(x) v y) != x.  [para(8(a,1),54(b,1,1,2))].
93 c(x) v c(x v y) = c(x).  [para(62(a,1),8(a,1,1)),flip(a)].
97 c(0 v c(x)) = x.  [para(16(a,1),62(a,1,1,2,1)),rewrite([61(3),9(3)])].
98 c(x v y) v c(x v c(x v y)) = c(x).
[para(62(a,1),19(a,1,2,1,2)),rewrite([9(5),93(11)])].
99 1 v x = 1.  [para(61(a,1),62(a,1,1,1)),rewrite([97(6)])].
108 x v 0 = x.  [para(16(a,1),63(a,1,2,1)),rewrite([61(2)])].
110 x v x = x.  [para(61(a,1),63(a,1,2,1,2)),rewrite([9(3),97(4)])].
111 x v (y v c(x)) = y v 1.  [para(16(a,1),65(a,1,2)),flip(a)].
118 c(c(c1) v (c(c1 v c2) v (c(c2 v c(c1)) v c(c1 v c(c2))))) = 0 | le(c1,c(c2)).
[resolve(67,b,55,a)].
123 x v 1 = 1.  [para(99(a,1),9(a,1)),flip(a)].
124 x v (y v c(x)) = 1.  [back_rewrite(111),rewrite([123(5)])].
125 0 v x = x.  [para(108(a,1),9(a,1)),flip(a)].
131 x v (y v x) = y v x.  [para(110(a,1),10(a,2,2)),rewrite([9(2)])].
344 le(x,c(y)) | c(y v c(x)) != x.  [para(9(a,1),82(b,1,1))].
422 c(x) v (c(x v y) v z) = c(x) v z.  [para(93(a,1),10(a,1,1)),flip(a)].
424 c(x) v (y v c(x v z)) = y v c(x).  [para(93(a,1),65(a,1,2)),flip(a)].
434 c(c(c1) v c(c2 v c(c1))) = 0 | le(c1,c(c2)).
[back_rewrite(118),rewrite([422(19),424(14),9(8)])].
439 c(c(c1) v c(c2 v c(c1))) != 0 | -perp(c1,c2).
[back_rewrite(66),rewrite([422(19),424(14),9(8)])].
459 c(x v y) v c(y v c(x v y)) = c(y).  [para(19(a,1),98(a,1,1,1)),rewrite([19(7)])].
21423 c(c(c1) v c(c2 v c(c1))) = 0 | c(c2 v c(c1)) = c1.
[resolve(434,b,52,a),rewrite([8(16),9(15)])].
59740 c(c2 v c(c1)) = c1.
[para(21423(a,1),30(a,2,1)),rewrite([30(15),8(24),131(23),8(24),8(28),124(27),61(23),9(23
),125(23),9(22),459(22),8(10),8(17),131(16),8(17),8(21),124(20),61(16),9(16),125(16),125(
15)]),flip(b),merge(b)].
59741 -perp(c1,c2).  [back_rewrite(439),rewrite([59740(7),9(4),16(4),61(2)]),xx(a)].
59742 -le(c1,c(c2)).  [ur(56,a,59741,a)].
59743 $F.  [ur(344,a,59742,a),rewrite([59740(5)]),xx(a)].

============================== end of proof ==========================



============================== PROOF =================================

% Proof 1 at 30.14 (+ 0.72) seconds.
% Length of proof is 50.
% Level of proof is 11.

1 le(x,y) <-> x = x ^ y # label("Df: less than") # label(non_clause).  [assumption].
2 perp(x,y) <-> le(x,c(y)) # label("Df: perpendicular") # label(non_clause).
[assumption].
4 int4(x,y) = 0 <-> perp(x,y) # label("Proposition 2.10int4") # label(non_clause) #
label(goal).  [goal].
7 x = c(c(x)) # label("AxL1").  [assumption].
8 c(c(x)) = x.  [copy(7),flip(a)].
9 x v y = y v x # label("AxL2").  [assumption].
10 (x v y) v z = x v (y v z) # label("AxL3").  [assumption].
12 x v (x ^ y) = x # label("AxL5").  [assumption].
13 x ^ (x v y) = x # label("AxL6").  [assumption].
```

```
14 c(x) ^ x = 0 # label("AxOL1").  [assumption].
15 c(x) v x = 1 # label("AxOL2").  [assumption].
16 x v c(x) = 1.  [copy(15),rewrite([9(2)])].
17 x ^ y = c(c(x) v c(y)) # label("AxOL3").  [assumption].
18 x v (c(x) ^ (y v x)) = y v x # label("OMA").  [assumption].
19 x v c(x v c(y v x)) = y v x.  [copy(18),rewrite([17(3),8(2)])].
27 i4(x,y) = ((c(c(y)) ^ c(x)) v (c(c(y)) ^ c(c(x)))) v (c(c(y)) v (c(y) ^ c(x))) #
label("Df: i4").  [assumption].
28 i4(x,y) = y v (c(y v x) v (c(c(y) v x) v c(c(y) v c(x)))).
[copy(27),rewrite([8(3),17(3),8(4),8(6),8(6),17(5),8(11),17(12),8(11),8(11),9(13),10(13)]
)].
47 int4(x,y) = c(i4(x,c(y))) # label("Df: int4").  [assumption].
48 int4(x,y) = c(c(y) v (c(c(y) v x) v (c(y v x) v c(y v c(x))))).
[copy(47),rewrite([28(3),8(7),8(9)])].
51 -le(x,y) | x ^ y = x # label("Df: less than").  [clausify(1)].
52 -le(x,y) | c(c(x) v c(y)) = x.  [copy(51),rewrite([17(2)])].
53 le(x,y) | x ^ y != x # label("Df: less than").  [clausify(1)].
54 le(x,y) | c(c(x) v c(y)) != x.  [copy(53),rewrite([17(2)])].
55 -perp(x,y) | le(x,c(y)) # label("Df: perpendicular").  [clausify(2)].
56 perp(x,y) | -le(x,c(y)) # label("Df: perpendicular").  [clausify(2)].
57 int4(c1,c2) = 0 | perp(c1,c2) # label("Proposition 2.10int4").  [deny(4)].
58 c(c(c2) v (c(c1 v c2) v (c(c2 v c(c1)) v c(c1 v c(c2))))) = 0 | perp(c1,c2).
[copy(57),rewrite([48(3),9(6),9(10),9(18),10(18)])].
59 int4(c1,c2) != 0 | -perp(c1,c2) # label("Proposition 2.10int4").  [deny(4)].
60 c(c(c2) v (c(c1 v c2) v (c(c2 v c(c1)) v c(c1 v c(c2))))) != 0 | -perp(c1,c2).
[copy(59),rewrite([48(3),9(6),9(10),9(18),10(18)])].
61 c(1) = 0.  [back_rewrite(14),rewrite([17(2),8(2),16(2)])].
62 c(c(x) v c(x v y)) = x.  [back_rewrite(13),rewrite([17(2)])].
63 x v c(c(x) v c(y)) = x.  [back_rewrite(12),rewrite([17(1)])].
84 perp(x,c(y)) | -le(x,y).  [para(8(a,1),56(b,2))].
85 c(c(c2) v (c(c1 v c2) v (c(c2 v c(c1)) v c(c1 v c(c2))))) = 0 | le(c1,c(c2)).
[resolve(58,b,55,a)].
91 le(x,x v y).  [resolve(62,a,54,b)].
92 c(x) v c(x v y) = c(x).  [para(62(a,1),8(a,1,1)),flip(a)].
107 x v 0 = x.  [para(16(a,1),63(a,1,2,1)),rewrite([61(2)])].
108 x v c(y v c(x)) = x.  [para(19(a,1),63(a,1,2,1))].
123 0 v x = x.  [para(107(a,1),9(a,1)),flip(a)].
134 perp(x,c(x v y)).  [resolve(84,b,91,a)].
252 c(x) v c(y v x) = c(x).  [para(8(a,1),108(a,1,2,1,2))].
407 c(c(c2) v (c(c1 v c2) v (c(c2 v c(c1)) v c(c1 v c(c2))))) = 0 | c(c2 v c(c1)) = c1.
[resolve(85,b,52,a),rewrite([8(27),9(26)])].
420 c(x) v (c(x v y) v z) = c(x) v z.  [para(92(a,1),10(a,1,1)),flip(a)].
676 c(x) v (c(y v x) v z) = c(x) v z.  [para(252(a,1),10(a,1,1)),flip(a)].
693 c(c(c2) v c(c1 v c(c2))) = 0 | c(c2 v c(c1)) = c1.
[back_rewrite(407),rewrite([676(19),420(14)])].
695 c(c(c2) v c(c1 v c(c2))) != 0 | -perp(c1,c2).
[back_rewrite(60),rewrite([676(19),420(14)])].
37940 c(c2 v c(c1)) = c1 | c1 v c(c2) = c(c2).
[para(693(a,1),19(a,1,2)),rewrite([9(11),123(11)]),flip(b)].
38009 c1 v c(c2) = c(c2).  [para(37940(a,1),92(a,1,2)),rewrite([9(11)]),merge(b)].
38010 -perp(c1,c2).  [back_rewrite(695),rewrite([38009(6),8(5),9(4),16(4),61(2)]),xx(a)].
38014 $F.  [para(38009(a,1),134(a,2,1)),rewrite([8(4)]),unit_del(a,38010)].

============================== end of proof ==========================
```

Figure 3. Summary of a *prover9* ([2]) proof of Proposition 2.12, for each of i = 3,4 from orthomodular lattice theory. The proofs assume the default inference rules of *prover9*. The general form of a line in this proof is "*line_number conclusion [derivation]*", where *line_number* is a unique identifier of a line in the proof, and *conclusion* is the result of applying the *prover9* inference rules (such as *para*modulation, *copy*ing, and *rewrit*ing), noted in square brackets (denoting the *derivation*), to the lines cited in those brackets. Note that some of "logical" proof lines in the above have been transformed to two text lines, with the *derivation* appearing on a text line following a text line containing the first part of that logical line. The detailed syntax and semantics of these notations can be found in [2]. All *prover9* proofs are by default proofs by contradiction.

The total time to produce the proofs in Figure 3 on the platform described in Section 2.0 was approximately 1070 seconds.

4.0 Discussion

The results of Section 3.0 motivate several observations:

1. Both proofs in Figure 3 use L1, L2, L3, L5, L6, OL1, OL2, and OL3.

2. The proofs in Section 3.0 may be novel.

3. Companion papers provide proofs for i = 5 and for the converse proposition.

4. Proposition 2.12 can be regarded as a definition of quantum intersection; thus, this paper together with the papers mentioned in (3), constitute a proof that the definition of quantum intersection is equivalent to the OMA in orthomodular quantum logic. Companion papers derive equivalences for the OMA with definitions of quantum-identity and quantum-union. Collectively, these papers provide a theory of equivalence of the OMA with the quantum connectives. In light of these equivalences, QL without the OMA would hardly qualify as a logic.

5.0 Acknowledgements

This work benefited from discussions with Tom Oberdan, Frank Pecchioni, Tony Pawlicki, and the late John K. Prentice, whose passion for foundations of physics inspired those of us privileged to have known him. For any infelicities that remain, I am solely responsible.

6.0 References

[1] von Neumann J. *Mathematical Foundations of Quantum Mechanics*. 1936. Translated by R. T. Beyer. Princeton. 1983.

[2] McCune WW. *prover9 and mace4.* URL http://www.cs.unm.edu/~mccune/prover9/. 2009.

[3] Horner JK. MHT *prover9* scripts. 2011. Available from the author on request.

[4] Dalla Chiara ML and Giuntini R. *Quantum Logics*. URL http://xxx.lanl.gov/abs/quant-ph/0101028. 2004.

[5] Megill ND and Pavičić M. Orthomodular lattices and quantum algebra. *International Journal of Theoretical Physics* 40 (2001), pp. 1387-1410.

[6] Akhiezer NI and Glazman IM. *Theory of Linear Operators in Hilbert Space. Volume I.* Translated by M. Nestell. Frederick Ungar. 1961.

[7] Holland, Jr. SS Orthomodularity in infinite dimensions: a theorem of M. Solèr. *Bulletin of the American Mathematical Society* 32 (1995), pp. 205-234.

[8] Marsden EL and Herman LM. *A condition for distribution in orthomodular lattices.* Kansas State University Technical Report #40. 1974.

[9] Knuth DE and Bendix PB. Simple word problems in universal algebras. In J. Leech, ed. *Computational Problems in Abstract Algebra.* Pergamon Press. 1970. pp. 263-297.

[10] Chang CC and Keisler HJ. *Model Theory*. North-Holland. 1990. pp. 38-39.

[11] Birkhoff G. *Lattice Theory*. Third Edition. American Mathematical Society. 1967.

[12] Church A. *Introduction to Mathematical Logic. Volume I.* Princeton. 1956.

[13] Jauch J. *Foundations of Quantum Mechanics.* Addison-Wesley. 1968.

[14] Megill ND. *Metamath.* URL http://us.metamath.org/qlegif/mmql.html#unify. 2004.

[15] Horner JK. An automated deduction system for orthomodular lattice theory. *Proceedings of the 2005 International Conference on Artificial Intelligence.* CSREA Press. 2005. pp. 260-265.

[16] Horner JK. An automated equational logic deduction of join elimination in orthomodular lattice theory. *Proceedings of the 2007 International Conference on Artificial Intelligence.* CSREA Press. 2007. pp. 481-488.

[17] Messiah A. *Quantum Mechanics*. Dover. 1958.

[18] Horner JK. Using automated theorem-provers to aid the design of efficient compilers for quantum computing. *Los Alamos National Laboratory Quantum Institute Workshop.* December 9–10, 2002. URL http://www.lanl.gov/science/centers/quantum/qls_pdfs/horner.pdf.

[19] Birkhoff G and von Neumann J. The logic of quantum mechanics. *Annals of Mathematics* 37 (1936), 823-243.

[20] Nielsen MA and Chuang L . *Quantum Computation and Quantum Information*. Cambridge. 2000.

[21] Pavičić M and Megill N. Quantum and classical implicational algebras with primitive implication. *International Journal of Theoretical Physics* 37 (1998), 2091-2098. ftp://m3k.grad.hr/pavicic/quantum-logic/1998-int-j-theor-phys-2.ps.gz.

[22] Horner JK. An automated deduction of the relative strength of orthomodular and weakly orthomodular lattice theory. *Proceedings of the 2009 International Conference on Artificial Intelligence*. CSREA Press. 2009. pp. 525-530.

Quantum-Intersection Equivalents of the Orthomodularity Law in Quantum Logic: Part 5

Jack K. Horner
P. O. Box 266
Los Alamos, New Mexico 87544 USA
jhorner@cybermesa.com

FCS 2014

Abstract

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of quantum-mechanical behaviors and operations. In much the same way that the structure of conventional propositional (Boolean) logic (BL) is the logic of the description of the behavior of classical physical systems and is isomorphic to a Boolean algebra (BA), so also the algebra, C(H), of closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space is a logic of the descriptions of the behavior of quantum mechanical systems and is a model of an ortholattice (OL). An OL can thus be thought of as a kind of "quantum logic" (QL). C(H) is also a model of an orthomodular lattice, which is an OL conjoined with the orthomodularity axiom (OMA). The rationalization of the OMA as a claim proper to physics has proven problematic, motivating the question of whether the OMA and its equivalents are required in an adequate characterization of QL. Here I provide an automated deduction of a quantum-intersection-based equivalent of the OMA. The proof may be novel.

Keywords: automated deduction, quantum computing, orthomodular lattice, Hilbert space

1.0 Introduction

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of the description of quantum-mechanical behaviors ([1], [17], [18], [20]). In much the same way that conventional propositional (Boolean) logic (BL,[12]) is the logical structure of description of the behavior of classical physical systems (e.g. "the measurements of the position and momentum of particle P are commutative", i.e., can be measured in either other, yielding the same results) and is isomorphic to a Boolean lattice ([10], [11], [19]), so also the algebra, *C(H)*, of the closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space *H* ([1], [4], [6], [9], [13]) is *a* logic of the descriptions of the behavior of quantum mechanical systems (e.g., "the measurements of the position and momentum of particle P are *not* commutative") and is a model ([10]) of an ortholattice (OL; [4]). An OL can thus be thought of as a kind of "quantum logic" (QL; [19]). *C(H)* is also a model of (i.e., isomorphic to a set of sentences which hold in) an orthomodular lattice (OML; [4], [7]), which is an OL conjoined with the orthomodularity axiom (OMA; see Figure 1). The rationalization of the OMA as a claim proper to physics has proven problematic ([13], Section 5-6), motivating the question of whether the OMA is required in an adequate characterization of QL. Thus formulated, the question suggests that the OMA and its equivalents are specific to an OML, and that as a consequence, banning the OMA from QL yields a "truer" quantum logic.

Lattice axioms

```
      x = c(c(x))                       (AxLat1)
      x v y = y v x                     (AxLat2)
      (x v y) v z = x v (y v z)          (AxLat3)
      (x ^ y) ^ z = x ^ (y ^ z)          (AxLat4)
      x v (x ^ y) = x                   (AxLat5)
      x ^ (x v y) = x                   (AxLat6)
```

Ortholattice axioms

```
      c(x) ^ x = 0                      (AxOL1)
      c(x) v x = 1                      (AxOL2)
      x ^ y = c(c(x) v c(y))            (AxOL3)
```

Orthomodularity axiom

```
      y v (c(y) ^ (x v y)) = x v y      (OMA)
```

Definitions of implications and partial order

```
   i1(x,y) = c(x) v (x ^ y).
   i2(x,y) = i1(c(y), c(x).
   i3(x,y) = (c(x) ^ y) v (c(x) ^ c(y)) v i1(x,y).
   i4(x,y) = i3(c(y), c(x)).
   i5(x,y) = (x ^ y) v (c(x) ^ y) v (c(x) ^ c(y)).
   le(x,y) = (x = (x ^ y)).
```

```
Definitions of indexed intersection
   int1(x,y) = c(i1(x,c(y)))
   int2(x,y) = c(i2(x,c(y)))
   int3(x,y) = c(i3(x,c(y)))
   int4(x,y) = c(i4(x,c(y)))
   int5(x,y) = c(i5(x,c(y)))
```

```
where
   x, y are variables ranging over lattice nodes
   ^ is lattice meet
   v is lattice join
   c(x) is the orthocomplement of x
   i1(x,y) means x →1 y (Sasaki implication)
   i2(x,y) means x →2 y (Dishkant implication)
   i3(x,y) means x →3 y (Kalmbach implication)
   i4(x,y) means x →4 y (non-tollens implication)
   i5(x,y) means x →5 y (relevance implication)
   le(x,y) means x ≤ y
   <-> means if and only if
   =  is equivalence ([12])
   1 is the maximum lattice element (= x v c(x))
   0 is the minimum lattice element (= c(1))
```

Figure 1. Lattice, ortholattice, orthomodularity axioms, and some definitions.

There are at least 21 nominal equivalents (in the sense that ortholattice theory, together with these "equivalents", imply the OMA, and vice versa) of the OMA in quantum logic ([5], Theorem 2.5); as nominal equivalents, they are thus of import to optimizing quantum circuit design. Among these is the Proposition shown in Figure 2:

$$x \cap_i y \quad <-> \quad x \perp y$$

where

$x \cap_i y$ means $c(x \rightarrow_i c(y))$
$x \perp y$ means le(x,c(y))
i = 1,2,3,4,5

Figure 2. Proposition 2.12 of [5]

2.0 Method

The OML axiomatizations of Megill, Pavičić, and Horner ([5], [14], [15], [16], [21], [22]) were implemented in a *prover9* ([2]) script ([3]) configured to derive Proposition 2.12 of [5] from orthomoldular lattice theory, for i = 5 then executed in that framework on a Dell Inspiron 545 with an Intel Core2 Quad CPU Q8200 (clocked @ 2.33 GHz) and 8.00 GB RAM, running under the *Windows Vista Home Premium /Cygwin* operating environment.

3.0 Results

Figure 3 shows the proofs, generated by [3] on the platform described in Section 2.0, that orthomodular lattice theory implies Proposition 2.12 of [5] (i = 5).

```
============== PROOF =================================

% Proof 1 at 520.22 (+ 10.59) seconds.
% Length of proof is 66.
% Level of proof is 12.

1 le(x,y) <-> x = x ^ y # label("Df: less than") #
label(non_clause).  [assumption].
2 perp(x,y) <-> le(x,c(y)) # label("Df: perpendicular") #
label(non_clause).  [assumption].
4 int5(x,y) = 0 <-> perp(x,y) # label("Proposition 2.10int5") #
label(non_clause) # label(goal).  [goal].
7 x = c(c(x)) # label("AxL1").  [assumption].
8 c(c(x)) = x.  [copy(7),flip(a)].
9 x v y = y v x # label("AxL2").  [assumption].
10 (x v y) v z = x v (y v z) # label("AxL3").  [assumption].
12 x v (x ^ y) = x # label("AxL5").  [assumption].
13 x ^ (x v y) = x # label("AxL6").  [assumption].
14 c(x) ^ x = 0 # label("AxOL1").  [assumption].
15 c(x) v x = 1 # label("AxOL2").  [assumption].
16 x v c(x) = 1.  [copy(15),rewrite([9(2)])].
17 x ^ y = c(c(x) v c(y)) # label("AxOL3").  [assumption].
18 x v (c(x) ^ (y v x)) = y v x # label("OMA").  [assumption].
19 x v c(x v c(y v x)) = y v x.  [copy(18),rewrite([17(3),8(2)])].
29 i5(x,y) = ((x ^ y) v (c(x) ^ y)) v (c(x) ^ c(y)) # label("Df:
i5").  [assumption].
```

```
30 i5(x,y) = c(x v y) v (c(x v c(y)) v c(c(x) v c(y))).
[copy(29),rewrite([17(2),17(7),8(7),9(9),17(12),8(11),8(11),9(12)])]
.
49 int5(x,y) = c(i5(x,c(y))) # label("Df: int5").  [assumption].
50 int5(x,y) = c(c(x v c(y)) v (c(x v y) v c(c(x) v y))).
[copy(49),rewrite([30(3),8(6),8(9)])].
51 -le(x,y) | x ^ y = x # label("Df: less than").  [clausify(1)].
52 -le(x,y) | c(c(x) v c(y)) = x.  [copy(51),rewrite([17(2)])].
53 le(x,y) | x ^ y != x # label("Df: less than").  [clausify(1)].
54 le(x,y) | c(c(x) v c(y)) != x.  [copy(53),rewrite([17(2)])].
55 -perp(x,y) | le(x,c(y)) # label("Df: perpendicular").
[clausify(2)].
56 perp(x,y) | -le(x,c(y)) # label("Df: perpendicular").
[clausify(2)].
57 int5(c1,c2) = 0 | perp(c1,c2) # label("Proposition 2.10int5").
[deny(4)].
58 c(c(c1 v c2) v (c(c2 v c(c1)) v c(c1 v c(c2)))) = 0 |
perp(c1,c2).  [copy(57),rewrite([50(3),9(13),9(16),10(16)])].
59 int5(c1,c2) != 0 | -perp(c1,c2) # label("Proposition 2.10int5").
[deny(4)].
60 c(c(c1 v c2) v (c(c2 v c(c1)) v c(c1 v c(c2)))) != 0 | -
perp(c1,c2).  [copy(59),rewrite([50(3),9(13),9(16),10(16)])].
61 c(1) = 0.  [back_rewrite(14),rewrite([17(2),8(2),16(2)])].
62 c(c(x) v c(x v y)) = x.  [back_rewrite(13),rewrite([17(2)])].
63 x v c(c(x) v c(y)) = x.  [back_rewrite(12),rewrite([17(1)])].
65 x v (y v z) = y v (x v z).
[para(9(a,1),10(a,1,1)),rewrite([10(2)])].
72 x v (y v c(x v y)) = 1.  [para(16(a,1),10(a,1)),flip(a)].
73 x v c(x v c(x v y)) = y v x.  [para(9(a,1),19(a,1,2,1,2,1))].
85 c(c(c1 v c2) v (c(c2 v c(c1)) v c(c1 v c(c2)))) = 0 |
le(c1,c(c2)).  [resolve(58,b,55,a)].
91 le(x,x v y).  [resolve(62,a,54,b)].
92 c(x) v c(x v y) = c(x).  [para(62(a,1),8(a,1,1)),flip(a)].
96 c(0 v c(x)) = x.
[para(16(a,1),62(a,1,1,2,1)),rewrite([61(3),9(3)])].
97 c(x v y) v c(x v c(x v y)) = c(x).
[para(62(a,1),19(a,1,2,1,2)),rewrite([9(5),92(11)])].
98 1 v x = 1.  [para(61(a,1),62(a,1,1,1)),rewrite([96(6)])].
103 x v c(c(x) v y) = x.  [para(8(a,1),63(a,1,2,1,2))].
107 x v 0 = x.  [para(16(a,1),63(a,1,2,1)),rewrite([61(2)])].
109 x v x = x.  [para(61(a,1),63(a,1,2,1,2)),rewrite([9(3),96(4)])].
110 x v (y v c(x)) = y v 1.  [para(16(a,1),65(a,1,2)),flip(a)].
118 le(x,y v x).  [para(9(a,1),91(a,2))].
121 x v 1 = 1.  [para(98(a,1),9(a,1)),flip(a)].
122 x v (y v c(x)) = 1.  [back_rewrite(110),rewrite([121(5)])].
123 0 v x = x.  [para(107(a,1),9(a,1)),flip(a)].
129 x v (y v x) = y v x.
[para(109(a,1),10(a,2,2)),rewrite([9(2)])].
160 x v (y v c(x v (y v c(x v (y v z))))) = z v (x v y).
[para(73(a,1),10(a,1)),rewrite([10(5),10(7)]),flip(a)].
178 le(c(c(x) v y),x).  [para(103(a,1),118(a,2))].
183 perp(c(x v y),x).  [resolve(178,a,56,b),rewrite([8(2)])].
407 c(c(c1 v c2) v (c(c2 v c(c1)) v c(c1 v c(c2)))) = 0 | c(c2 v
c(c1)) = c1.  [resolve(85,b,52,a),rewrite([8(24),9(23)])].
```

```
422 c(x) v (y v c(x v z)) = y v c(x).
[para(92(a,1),65(a,1,2)),flip(a)].
454 c(x v y) v c(y v c(x v y)) = c(y).
[para(19(a,1),97(a,1,1,1)),rewrite([19(7)])].
1122 x v (c(y v x) v (z v y)) = 1.
[para(72(a,1),160(a,1,2,2,1,2,2,1,2)),rewrite([121(2),61(2),107(2),7
2(4),10(6)]),flip(a)].
7094 x v (c(y) v c(c(y v z) v x)) = 1.
[para(92(a,1),1122(a,1,2,2)),rewrite([9(6)])].
31295 c(c2 v c(c1)) = c1 | c(c1) v c(c2 v c(c1)) = 1.
[para(407(a,1),7094(a,1,2,2)),rewrite([9(22),123(22),9(21),422(21),9
(15)])].
91125 c(c2 v c(c1)) = c1.
[para(31295(b,1),30(a,2,1,1)),rewrite([30(15),8(24),129(23),8(24),8(
28),122(27),61(23),9(23),123(23),9(22),454(22),8(10),61(10),8(17),12
9(16),8(17),8(21),122(20),61(16),9(16),123(16),123(15)]),flip(b),mer
ge(b)].
91126 c(c1 v (c(c1 v c2) v c(c1 v c(c2)))) != 0 | -perp(c1,c2).
[back_rewrite(60),rewrite([91125(9),65(12)])].
91131 c2 v c(c1) = c(c1).  [para(91125(a,1),8(a,1,1)),flip(a)].
91133 c2 v c(c1 v c2) = c(c1).
[para(91125(a,1),73(a,1,2,1,2)),rewrite([9(4),9(10),91131(10)])].
91135 perp(c1,c2).  [para(91125(a,1),183(a,1))].
91141 c1 v c(c2) = c(c2).
[para(91125(a,1),92(a,1,2)),rewrite([9(4)])].
91935 $F.
[back_unit_del(91126),rewrite([91141(9),8(8),9(7),91133(7),16(4),61(
2)]),xx(a),unit_del(a,91135)].

============== end of proof ==========================
```

Figure 3. Summary of a *prover9* ([2]) proof of Proposition 2.12, for i = 5 from orthomodular lattice theory. The proofs assume the default inference rules of *prover9*. The general form of a line in this proof is "*line_number conclusion [derivation]*", where *line_number* is a unique identifier of a line in the proof, and *conclusion* is the result of applying the *prover9* inference rules (such as *para*modulation, *copy*ing, and *rewrit*ing), noted in square brackets (denoting the *derivation*), to the lines cited in those brackets. Note that some of "logical" proof lines in the above have been transformed to two text lines, with the *derivation* appearing on a text line following a text line containing the first part of that logical line. The detailed syntax and semantics of these notations can be found in [2]. All *prover9* proofs are by default proofs by contradiction.

The total time to produce the proofs in Figure 3 on the platform described in Section 2.0 was approximately 1070 seconds.

4.0 Discussion

The results of Section 3.0 motivate several observations:

1. The proof in Figure 3 uses L1, L2, L3, L5, L6, OL1, OL2, and OL3.

2. The proofs in Section 3.0 may be novel.

3. Companion papers provide proofs for i = 1,2,3,4, and for the converse.

4. Proposition 2.12 can be regarded as a definition of quantum intersection; thus, this paper together with the papers mentioned in (3), constitute a proof that the definition of quantum intersection is equivalent to the OMA in orthomodular quantum logic. Companion papers derive equivalences for the OMA with definitions of quantum-identity and quantum-union. Collectively, these papers provide a theory of equivalence of the OMA with the quantum connectives. In light of these equivalences, QL without the OMA would hardly qualify as a logic.

5.0 Acknowledgements

This work benefited from discussions with Tom Oberdan, Frank Pecchioni, Tony Pawlicki, and the late John K. Prentice, whose passion for foundations of physics inspired those of us privileged to have known him. For any infelicities that remain, I am solely responsible.

6.0 References

[1] von Neumann J. *Mathematical Foundations of Quantum Mechanics*. 1936. Translated by R. T. Beyer. Princeton. 1983.

[2] McCune WW. *prover9 and mace4.* URL http://www.cs.unm.edu/~mccune/prover9/. 2009.

[3] Horner JK. MHT *prover9* scripts. 2011. Available from the author on request.

[4] Dalla Chiara ML and Giuntini R. *Quantum Logics*. URL http://xxx.lanl.gov/abs/quant-ph/0101028. 2004.

[5] Megill ND and Pavičić M. Orthomodular lattices and quantum algebra. *International Journal of Theoretical Physics* 40 (2001), pp. 1387-1410.

[6] Akhiezer NI and Glazman IM. *Theory of Linear Operators in Hilbert Space. Volume I.* Translated by M. Nestell. Frederick Ungar. 1961.

[7] Holland, Jr. SS Orthomodularity in infinite dimensions: a theorem of M. Solèr. *Bulletin of the American Mathematical Society* 32 (1995), pp. 205-234.

[8] Marsden EL and Herman LM. *A condition for distribution in orthomodular lattices.* Kansas State University Technical Report #40. 1974.

[9] Knuth DE and Bendix PB. Simple word problems in universal algebras. In J. Leech, ed. *Computational Problems in Abstract Algebra.* Pergamon Press. 1970. pp. 263-297.

[10] Chang CC and Keisler HJ. *Model Theory*. North-Holland. 1990. pp. 38-39.

[11] Birkhoff G. *Lattice Theory.* Third Edition. American Mathematical Society. 1967.

[12] Church A. *Introduction to Mathematical Logic. Volume I.* Princeton. 1956.

[13] Jauch J. *Foundations of Quantum Mechanics.* Addison-Wesley. 1968.

[14] Megill ND. *Metamath.* URL http://us.metamath.org/qlegif/mmql.html#unify. 2004.

[15] Horner JK. An automated deduction system for orthomodular lattice theory. *Proceedings of the 2005 International Conference on Artificial Intelligence.* CSREA Press. 2005. pp. 260-265.

[16] Horner JK. An automated equational logic deduction of join elimination in orthomodular lattice theory. *Proceedings of the 2007 International Conference on Artificial Intelligence.* CSREA Press. 2007. pp. 481-488.

[17] Messiah A. *Quantum Mechanics*. Dover. 1958.

[18] Horner JK. Using automated theorem-provers to aid the design of efficient compilers for quantum computing. *Los Alamos National Laboratory Quantum Institute Workshop*. December 9–10, 2002. URL http://www.lanl.gov/science/centers/quantum/qls_pdfs/horner.pdf.

[19] Birkhoff G and von Neumann J. The logic of quantum mechanics. *Annals of Mathematics* 37 (1936), 823-243.

[20] Nielsen MA and Chuang L . *Quantum Computation and Quantum Information*. Cambridge. 2000.

[21] Pavičić M and Megill N. Quantum and classical implicational algebras with primitive implication. *International Journal of Theoretical Physics* 37 (1998), 2091-2098. ftp://m3k.grad.hr/pavicic/quantum-logic/1998-int-j-theor-phys-2.ps.gz.

[22] Horner JK. An automated deduction of the relative strength of orthomodular and weakly orthomodular lattice theory. *Proceedings of the 2009 International Conference on Artificial Intelligence*. CSREA Press. 2009. pp. 525-530.

Quantum-Identity Equivalents of the Orthomodularity Law in Quantum Logic: Part 1

Jack K. Horner
P. O. Box 266
Los Alamos, New Mexico 87544 USA

FCS 2014

Abstract

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of quantum-mechanical behaviors and operations. In much the same way that the structure of conventional propositional (Boolean) logic (BL) is the logic of the description of the behavior of classical physical systems and is isomorphic to a Boolean algebra (BA), so also the algebra, C(H), of closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space is a logic of the descriptions of the behavior of quantum mechanical systems and is a model of an ortholattice (OL). An OL can thus be thought of as a kind of "quantum logic" (QL). C(H) is also a model of an orthomodular lattice, which is an OL conjoined with the orthomodularity axiom (OMA). The rationalization of the OMA as a claim proper to physics has proven problematic, motivating the question of whether the OMA and its equivalents are required in an adequate characterization of QL. Here I provide an automated deduction of the OMA from three quantum-identity-based equivalents of the OMA. The proofs may be novel.

Keywords: automated deduction, quantum computing, orthomodular lattice, Hilbert space

1.0 Introduction

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of the description of quantum-mechanical behaviors ([1], [17], [18], [20]). In much the same way that conventional propositional (Boolean) logic (BL,[12]) is the logical structure of description of the behavior of classical physical systems (e.g. "the measurements of the position and momentum of particle P are commutative", i.e., can be measured in either other, yielding the same results) and is isomorphic to a Boolean lattice ([10], [11], [19]), so also the algebra, *C(H)*, of the closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space *H* ([1], [4], [6], [9], [13]) is *a* logic of the descriptions of the behavior of quantum mechanical systems (e.g., "the measurements of the position and momentum of particle P are *not* commutative") and is a model ([10]) of an ortholattice (OL; [4]). An OL can thus be thought of as a kind of "quantum logic" (QL; [19]). *C(H)* is also a model of (i.e., isomorphic to a set of sentences which hold in) an orthomodular lattice (OML; [4], [7]), which is an OL conjoined with the orthomodularity axiom (OMA; see Figure 1). The rationalization of the OMA as a claim proper to physics has proven problematic ([13], Section 5-6), motivating the question of whether the OMA is required in an adequate characterization of QL. Thus formulated, the question suggests that the OMA and its equivalents are specific to an OML, and that as a consequence, banning the OMA from QL yields a "truer" quantum logic.

Lattice axioms

```
      x = c(c(x))                       (AxLat1)
      x v y = y v x                     (AxLat2)
      (x v y) v z = x v (y v z)          (AxLat3)
      (x ^ y) ^ z = x ^ (y ^ z)          (AxLat4)
      x v (x ^ y) = x                   (AxLat5)
      x ^ (x v y) = x                   (AxLat6)
```

Ortholattice axioms

```
      c(x) ^ x = 0                      (AxOL1)
      c(x) v x = 1                      (AxOL2)
      x ^ y = c(c(x) v c(y))            (AxOL3)
```

Orthomodularity axiom

```
      y v (c(y) ^ (x v y)) = x v y      (OMA)
```

Definitions of implications and partial order

```
   i1(x,y) = c(x) v (x ^ y).
   i2(x,y) = i1(c(y), c(x).
   i3(x,y) = (c(x) ^ y) v (c(x) ^ c(y)) v i1(x,y).
   i4(x,y) = i3(c(y), c(x)).
   i5(x,y) = (x ^ y) v (c(x) ^ y) v (c(x) ^ c(y)).
   le(x,y) = (x = (x ^ y)).
```

Definitions of indexed identities

```
    id1(x,y) = i1(x,y) ^ i0(y, x)
    id2(x,y) = i2(x,y) ^ i0(y, x)
    id3(x,y) = i3(x,y) ^ i0(y, x)
    id4(x,y) = i4(x,y) ^ i0(y, x)
    id5(x,y) = i5(x,y) ^ i0(y, x)
```

where
   x, y are variables ranging over lattice nodes
   ^ is lattice meet
   v is lattice join
   c(x) is the orthocomplement of x
   i1(x,y) means $x \rightarrow_1 y$ (Sasaki implication)
   i2(x,y) means $x \rightarrow_2 y$ (Dishkant implication)
   i3(x,y) means $x \rightarrow_3 y$ (Kalmbach implication)
   i4(x,y) means $x \rightarrow_4 y$ (non-tollens implication)
   i5(x,y) means $x \rightarrow_5 y$ (relevance implication)
   le(x,y) means $x \leq y$
   <-> means if and only if
   = is equivalence ([12])
   1 is the maximum lattice element (= x v c(x))
   0 is the minimum lattice element (= c(1))

Figure 1. Lattice, ortholattice, orthomodularity axioms, and some definitions.

There are at least 21 nominal equivalents (in the sense that ortholattice theory, together with these "equivalents", imply the OMA, and vice versa) of the OMA in quantum logic ([5], Theorem 2.5); as nominal equivalents, they are thus of import to optimizing quantum circuit design. Among these is the Proposition shown in Figure 2:

$$x \equiv_i y \quad <-> \quad x = y$$

where

$$x \equiv_i y \text{ means } (x \rightarrow_i y) \wedge (x \rightarrow_0 y)$$

$x \rightarrow_0 y$ means $c(x) \vee y$
$i = 1,2,3,4,5$

Figure 2. Proposition 2.13 of [5]

2.0 Method

The OML axiomatizations of Megill, Pavičić, and Horner ([5], [14], [15], [16], [21], [22]) were implemented in a *prover9* ([2]) script ([3]) configured to derive the OMA from Proposition 2.13 of [5], for each of i = 1,2,3 together with ortholattice theory (orthomodular lattice theory, without the OMA), then executed in that framework on a Dell Inspiron 545 with an Intel Core2 Quad CPU Q8200 (clocked @ 2.33 GHz) and 8.00 GB RAM, running under the *Windows Vista Home Premium /Cygwin* operating environment.

3.0 Results

Figure 3 shows the proofs, generated by [3] on the platform described in Section 2.0, that Proposition 2.13 (for each of i = 1,2,3), together with ortholattice theory, imply the OMA.

```
============================== PROOF =================================

% Proof 1 at 1.70 (+ 0.09) seconds: "OMA".
% Length of proof is 49.
% Level of proof is 12.

4 id1(x,y) = 1 <-> x = y # label("Hypothesis for Proposition 2.10id1") #
label(non_clause).  [assumption].
5 y v (c(y) ^ (x v y)) = x v y # label("OMA") # label(non_clause) # label(goal).  [goal].
12 x = c(c(x)) # label("AxL1").  [assumption].
13 c(c(x)) = x.  [copy(12),flip(a)].
14 x v y = y v x # label("AxL2").  [assumption].
15 (x v y) v z = x v (y v z) # label("AxL3").  [assumption].
17 x v (x ^ y) = x # label("AxL5").  [assumption].
18 x ^ (x v y) = x # label("AxL6").  [assumption].
19 c(x) ^ x = 0 # label("AxOL1").  [assumption].
20 c(x) v x = 1 # label("AxOL2").  [assumption].
21 x v c(x) = 1.  [copy(20),rewrite([14(2)])].
22 x ^ y = c(c(x) v c(y)) # label("AxOL3").  [assumption].
23 i0(x,y) = c(x) v y # label("Df: i0").  [assumption].
24 i1(x,y) = c(x) v (x ^ y) # label("Df: i1").  [assumption].
25 i1(x,y) = c(x) v c(c(x) v c(y)).  [copy(24),rewrite([22(3)])].
54 id1(x,y) = i1(x,y) ^ i0(y,x) # label("Df: id1").  [assumption].
55 id1(x,y) = c(c(c(y) v x) v c(c(x) v c(c(x) v c(y)))).
[copy(54),rewrite([25(2),23(8),22(10),14(12)])].
64 id1(x,y) != 1 | y = x # label("Hypothesis for Proposition 2.10id1").  [clausify(4)].
65 c(c(c(x) v y) v c(c(y) v c(c(y) v c(x)))) != 1 | x = y.  [copy(64),rewrite([55(1)])].
68 c1 v (c(c1) ^ (c2 v c1)) != c2 v c1 # label("OMA") # answer("OMA").  [deny(5)].
69 c1 v c(c1 v c(c1 v c2)) != c1 v c2 # answer("OMA").
[copy(68),rewrite([14(6),22(7),13(4),14(12)])].
70 c(1) = 0.  [back_rewrite(19),rewrite([22(2),13(2),21(2)])].
71 c(c(x) v c(x v y)) = x.  [back_rewrite(18),rewrite([22(2)])].
72 x v c(c(x) v c(y)) = x.  [back_rewrite(17),rewrite([22(1)])].
74 x v (y v z) = y v (x v z).  [para(14(a,1),15(a,1,1)),rewrite([15(2)])].
82 x v (y v c(x v y)) = 1.  [para(21(a,1),15(a,1)),flip(a)].
105 c(c(c1 v (c2 v c(c1 v c(c1 v c(c1 v c2))))) v c(c(c1 v c2) v c(c(c1 v c2) v c(c1 v
c(c1 v c(c1 v c2)))))) != 1 # answer("OMA").  [ur(65,b,69,a),rewrite([14(14),15(14)])].
114 c(x) v c(x v y) = c(x).  [para(71(a,1),13(a,1,1)),flip(a)].
```

```
118 c(0 v c(x)) = x.  [para(21(a,1),71(a,1,1,2,1)),rewrite([70(3),14(3)])].
121 1 v x = 1.  [para(70(a,1),71(a,1,1,1)),rewrite([118(6)])].
133 x v c(c(x) v y) = x.  [para(13(a,1),72(a,1,2,1,2))].
137 x v 0 = x.  [para(21(a,1),72(a,1,2,1)),rewrite([70(2)])].
145 x v (y v c(x)) = y v 1.  [para(21(a,1),74(a,1,2)),flip(a)].
154 x v 1 = 1.  [para(121(a,1),14(a,1)),flip(a)].
156 x v (y v c(x)) = 1.  [back_rewrite(145),rewrite([154(5)])].
157 0 v x = x.  [para(137(a,1),14(a,1)),flip(a)].
164 c(x) v (y v x) = 1.  [para(13(a,1),156(a,1,2,2))].
171 x v c(y v c(x)) = x.  [para(14(a,1),133(a,1,2,1))].
183 c(x) v c(y v x) = c(x).  [para(13(a,1),171(a,1,2,1,2))].
184 x v (c(y v c(x)) v z) = x v z.  [para(171(a,1),15(a,1,1)),flip(a)].
185 x v (y v c(z v c(x v y))) = x v y.  [para(171(a,1),15(a,1)),flip(a)].
229 c(x) v (c(y v x) v z) = c(x) v z.  [para(183(a,1),15(a,1,1)),flip(a)].
232 c(x v y) v c(x v (z v y)) = c(x v y).  [para(74(a,1),183(a,1,2,1))].
499 x v (y v c(c(z v c(x)) v y)) = 1.
[para(184(a,1),164(a,1,2)),rewrite([14(7),15(7)])].
1916 c(x) v c(c(y v x) v c(z v x)) = 1.
[para(499(a,1),229(a,1)),rewrite([13(4)]),flip(a)].
3074 x v (y v c(x v c(z v c(x v y)))) = 1.
[para(114(a,1),1916(a,1,2,1,1,1)),rewrite([13(3),13(3),15(8)])].
3115 c(c1 v c2) v c(c(c1 v c2) v c(c1 v c(c1 v c(c1 v c2)))) != 1 # answer("OMA").
[back_rewrite(105),rewrite([3074(14),70(2),157(24),13(23)])].
3142 c(x v y) v c(x v c(z v c(x v y))) = c(x v c(z v c(x v y))).
[para(185(a,1),232(a,1,2,1)),rewrite([14(9)])].
3180 $F # answer("OMA").
[back_rewrite(3115),rewrite([3142(19),13(15),74(14),82(14)]),xx(a)].

============================== end of proof ==========================


============================== PROOF =================================

% Proof 1 at 0.11 (+ 0.01) seconds: "OMA".
% Length of proof is 36.
% Level of proof is 8.

4 id2(x,y) = 1 <-> x = y # label("Hypothesis for Proposition 2.10id2") #
label(non_clause).  [assumption].
5 y v (c(y) ^ (x v y)) = x v y # label("OMA") # label(non_clause) # label(goal).  [goal].
12 x = c(c(x)) # label("AxL1").  [assumption].
13 c(c(x)) = x.  [copy(12),flip(a)].
14 x v y = y v x # label("AxL2").  [assumption].
15 (x v y) v z = x v (y v z) # label("AxL3").  [assumption].
17 x v (x ^ y) = x # label("AxL5").  [assumption].
18 x ^ (x v y) = x # label("AxL6").  [assumption].
19 c(x) ^ x = 0 # label("AxOL1").  [assumption].
20 c(x) v x = 1 # label("AxOL2").  [assumption].
21 x v c(x) = 1.  [copy(20),rewrite([14(2)])].
22 x ^ y = c(c(x) v c(y)) # label("AxOL3").  [assumption].
23 i0(x,y) = c(x) v y # label("Df: i0").  [assumption].
26 i2(x,y) = c(c(y)) v (c(y) ^ c(x)) # label("Df: i2").  [assumption].
27 i2(x,y) = y v c(y v x).  [copy(26),rewrite([13(3),22(4),13(3),13(3)])].
56 id2(x,y) = i2(x,y) ^ i0(y,x) # label("Df: id2").  [assumption].
57 id2(x,y) = c(c(c(y) v x) v c(y v c(y v x))).
[copy(56),rewrite([27(2),23(5),22(7),14(9)])].
64 id2(x,y) != 1 | y = x # label("Hypothesis for Proposition 2.10id2").  [clausify(4)].
65 c(c(c(x) v y) v c(x v c(x v y))) != 1 | x = y.  [copy(64),rewrite([57(1)])].
68 c1 v (c(c1) ^ (c2 v c1)) != c2 v c1 # label("OMA") # answer("OMA").  [deny(5)].
69 c1 v c(c1 v c(c1 v c2)) != c1 v c2 # answer("OMA").
[copy(68),rewrite([14(6),22(7),13(4),14(12)])].
70 c(1) = 0.  [back_rewrite(19),rewrite([22(2),13(2),21(2)])].
71 c(c(x) v c(x v y)) = x.  [back_rewrite(18),rewrite([22(2)])].
72 x v c(c(x) v c(y)) = x.  [back_rewrite(17),rewrite([22(1)])].
74 x v (y v z) = y v (x v z).  [para(14(a,1),15(a,1,1)),rewrite([15(2)])].
82 x v (y v c(x v y)) = 1.  [para(21(a,1),15(a,1)),flip(a)].
106 c(0 v c(c1 v (c2 v c(c1 v (c1 v (c2 v c(c1 v c(c1 v c2))))))) != 1 # answer("OMA").
[ur(65,b,69,a(flip)),rewrite([74(14),82(14),70(2),74(17),15(16),15(19)])].
114 c(0 v c(x)) = x.  [para(21(a,1),71(a,1,1,2,1)),rewrite([70(3),14(3)])].
119 c1 v (c2 v c(c1 v (c1 v (c2 v c(c1 v c(c1 v c2)))))) != 1 # answer("OMA").
[back_rewrite(106),rewrite([114(22)])].
```

```
130 x v c(c(x) v y) = x.  [para(13(a,1),72(a,1,2,1,2))].
136 x v x = x.  [para(70(a,1),72(a,1,2,1,2)),rewrite([14(3),114(4)])].
156 x v (x v y) = x v y.  [para(136(a,1),15(a,1,1)),flip(a)].
164 c1 v (c2 v c(c1 v (c2 v c(c1 v c(c1 v c2))))) != 1 # answer("OMA").
[back_rewrite(119),rewrite([156(15)])].
181 x v c(y v c(x)) = x.  [para(14(a,1),130(a,1,2,1))].
197 x v (y v c(z v c(x v y))) = x v y.  [para(181(a,1),15(a,1)),flip(a)].
203 $F # answer("OMA").  [back_rewrite(164),rewrite([197(13),82(8)]),xx(a)].

============================== end of proof ==========================



============================== PROOF =================================

% Proof 1 at 0.08 (+ 0.03) seconds: "OMA".
% Length of proof is 37.
% Level of proof is 7.

4 id3(x,y) = 1 <-> x = y # label("Hypothesis for Proposition 2.10id3") #
label(non_clause).  [assumption].
5 y v (c(y) ^ (x v y)) = x v y # label("OMA") # label(non_clause) # label(goal).  [goal].
12 x = c(c(x)) # label("AxL1").  [assumption].
13 c(c(x)) = x.  [copy(12),flip(a)].
14 x v y = y v x # label("AxL2").  [assumption].
15 (x v y) v z = x v (y v z) # label("AxL3").  [assumption].
17 x v (x ^ y) = x # label("AxL5").  [assumption].
18 x ^ (x v y) = x # label("AxL6").  [assumption].
19 c(x) ^ x = 0 # label("AxOL1").  [assumption].
20 c(x) v x = 1 # label("AxOL2").  [assumption].
21 x v c(x) = 1.  [copy(20),rewrite([14(2)])].
22 x ^ y = c(c(x) v c(y)) # label("AxOL3").  [assumption].
23 i0(x,y) = c(x) v y # label("Df: i0").  [assumption].
28 i3(x,y) = ((c(x) ^ y) v (c(x) ^ c(y))) v (c(x) v (x ^ y)) # label("Df: i3").
[assumption].
29 i3(x,y) = c(x v y) v (c(x v c(y)) v (c(x) v c(c(x) v c(y)))).
[copy(28),rewrite([22(3),13(3),22(7),13(6),13(6),14(7),22(9),15(14)])].
58 id3(x,y) = i3(x,y) ^ i0(y,x) # label("Df: id3").  [assumption].
59 id3(x,y) = c(c(c(y) v x) v c(c(x v y) v (c(x v c(y)) v (c(x) v c(c(x) v c(y)))))).
[copy(58),rewrite([29(2),23(15),22(17),14(19)])].
64 id3(x,y) != 1 | y = x # label("Hypothesis for Proposition 2.10id3").  [clausify(4)].
65 c(c(c(x) v y) v c(c(y v x) v (c(y v c(x)) v (c(y) v c(c(y) v c(x))))))) != 1 | x = y.
[copy(64),rewrite([59(1)])].
68 c1 v (c(c1) ^ (c2 v c1)) != c2 v c1 # label("OMA") # answer("OMA").  [deny(5)].
69 c1 v c(c1 v c(c1 v c2)) != c1 v c2 # answer("OMA").
[copy(68),rewrite([14(6),22(7),13(4),14(12)])].
70 c(1) = 0.  [back_rewrite(19),rewrite([22(2),13(2),21(2)])].
71 c(c(x) v c(x v y)) = x.  [back_rewrite(18),rewrite([22(2)])].
72 x v c(c(x) v c(y)) = x.  [back_rewrite(17),rewrite([22(1)])].
74 x v (y v z) = y v (x v z).  [para(14(a,1),15(a,1,1)),rewrite([15(2)])].
76 c(c(c(x) v y) v c(c(y) v (c(y v x) v (c(y v c(x)) v c(c(y) v c(x)))))) != 1 | x = y.
[back_rewrite(65),rewrite([74(15),74(16)])].
84 x v (y v c(x v y)) = 1.  [para(21(a,1),15(a,1)),flip(a)].
91 c(0 v c(x)) = x.  [para(21(a,1),71(a,1,1,2,1)),rewrite([70(3),14(3)])].
95 x v c(c(x) v y) = x.  [para(13(a,1),72(a,1,2,1,2))].
99 x v 0 = x.  [para(21(a,1),72(a,1,2,1)),rewrite([70(2)])].
100 x v x = x.  [para(70(a,1),72(a,1,2,1,2)),rewrite([14(3),91(4)])].
104 0 v x = x.  [para(99(a,1),14(a,1)),flip(a)].
105 x v (x v y) = x v y.  [para(100(a,1),15(a,1,1)),flip(a)].
116 c(c1 v c(c1 v c(c1 v c2))) v (c(c1 v (c2 v c(c1 v c(c1 v c2)))) v c(c(c1 v c2) v c(c1
v c(c1 v c(c1 v c2))))) != 1 # answer("OMA").
[ur(76,b,69,a(flip)),rewrite([74(14),84(14),70(2),14(24),74(24),15(23),105(24),14(37),74(
37),84(37),70(25),14(39),104(41),104(43),13(42)])].
137 x v c(y v c(x)) = x.  [para(14(a,1),95(a,1,2,1))].
156 x v (y v c(z v c(x v y))) = x v y.  [para(137(a,1),15(a,1)),flip(a)].
161 $F # answer("OMA").  [back_rewrite(116),rewrite([156(21),74(32),84(32)]),xx(a)].

============================== end of proof ==========================
```

Figure 3. Summary of a *prover9* ([2]) proof of Proposition 2.11, for each of i = 1,2,3. The proofs assume the default inference rules of *prover9*. The general form of a line in this proof is "*line_number conclusion [derivation]*", where *line_number* is a unique identifier of a line in the proof, and *conclusion* is the result of applying the *prover9* inference rules (such as *para*modulation, *copy*ing, and *rewrit*ing), noted in square brackets (denoting the *derivation*), to the lines cited in those brackets. Note that some of "logical" proof lines in the above have been transformed to two text lines, with the *derivation* appearing on a text line following a text line containing the first part of that logical line. The detailed syntax and semantics of these notations can be found in [2]. All *prover9* proofs are by default proofs by contradiction.

The total time to produce the proofs in Figure 3 on the platform described in Section 2.0 was approximately 2 seconds.

4.0 Discussion

The results of Section 3.0 motivate several observations:

1. The proofs in Figure 3 use L1, L2, L3, L5, L6, OL1, OL2, and OL3.

2. The proofs in Section 3.0 may be novel.

3. Companion papers provide proofs for i = 4,5, and show that orthomodular lattice theory implies Propositions 2.13i, i = 1,2,3,4,5.

4. Proposition 2.13 can be regarded as a definition of quantum identity; thus, this paper together papers mentioned in (3), constitute a proof that the definition of quantum intersection is equivalent to the OMA in orthomodular quantum logic. Companion papers derive equivalences for the OMA with definitions of quantum-intersection and quantum-union. Collectively, these papers provide a theory of equivalence of the OMA with the quantum connectives. In light of these equivalences, QL without the OMA would hardly qualify as a logic.

5.0 Acknowledgements

This work benefited from discussions with Tom Oberdan, Frank Pecchioni, Tony Pawlicki, and the late John K. Prentice, whose passion for foundations of physics inspired those of us privileged to have known him. For any infelicities that remain, I am solely responsible.

6.0 References

[1] von Neumann J. *Mathematical Foundations of Quantum Mechanics*. 1936. Translated by R. T. Beyer. Princeton. 1983.
[2] McCune WW. *prover9 and mace4.* URL http://www.cs.unm.edu/~mccune/prover9/. 2009.
[3] Horner JK. MHT *prover9* scripts. 2011. Available from the author on request.
[4] Dalla Chiara ML and Giuntini R. *Quantum Logics*. URL http://xxx.lanl.gov/abs/quant-ph/0101028. 2004.
[5] Megill ND and Pavičić M. Orthomodular lattices and quantum algebra. *International Journal of Theoretical Physics* 40 (2001), pp. 1387-1410.
[6] Akhiezer NI and Glazman IM. *Theory of Linear Operators in Hilbert Space. Volume I.* Translated by M. Nestell. Frederick Ungar. 1961.
[7] Holland, Jr. SS Orthomodularity in infinite dimensions: a theorem of M. Solèr. *Bulletin of the American Mathematical Society* 32 (1995), pp. 205-234.
[8] Marsden EL and Herman LM. *A condition for distribution in orthomodular lattices.* Kansas State University Technical Report #40. 1974.
[9] Knuth DE and Bendix PB. Simple word problems in universal algebras. In J. Leech, ed. *Computational Problems in Abstract Algebra.* Pergamon Press. 1970. pp. 263-297.
[10] Chang CC and Keisler HJ. *Model Theory.* North-Holland. 1990. pp. 38-39.
[11] Birkhoff G. *Lattice Theory.* Third Edition. American Mathematical Society. 1967.
[12] Church A. *Introduction to Mathematical Logic. Volume I.* Princeton. 1956.
[13] Jauch J. *Foundations of Quantum Mechanics.* Addison-Wesley. 1968.
[14] Megill ND. *Metamath.* URL http://us.metamath.org/qlegif/mmql.html#unify. 2004.
[15] Horner JK. An automated deduction system for orthomodular lattice theory. *Proceedings of the 2005 International Conference on Artificial Intelligence.* CSREA Press. 2005. pp. 260-265.
[16] Horner JK. An automated equational logic deduction of join elimination in orthomodular lattice theory. *Proceedings of the 2007 International Conference on Artificial Intelligence.* CSREA Press. 2007. pp. 481-488.
[17] Messiah A. *Quantum Mechanics*. Dover. 1958.
[18] Horner JK. Using automated theorem-provers to aid the design of efficient compilers for quantum computing. *Los Alamos National Laboratory Quantum Institute Workshop.* December 9–10, 2002. URL http://www.lanl.gov/science/centers/quantum/qls_pdfs/horner.pdf.
[19] Birkhoff G and von Neumann J. The logic of quantum mechanics. *Annals of Mathematics* 37 (1936), 823-243.
[20] Nielsen MA and Chuang L . *Quantum Computation and Quantum Information*. Cambridge. 2000.
[21] Pavičić M and Megill N. Quantum and classical implicational algebras with primitive implication. *International Journal of Theoretical Physics* 37 (1998), 2091-2098. ftp://m3k.grad.hr/pavicic/quantum-logic/1998-int-j-theor-phys-2.ps.gz.
[22] Horner JK. An automated deduction of the relative strength of orthomodular and weakly orthomodular lattice theory. *Proceedings of the 2009 International Conference on Artificial Intelligence*. CSREA Press. 2009. pp. 525-530.

Quantum-Identity-Based Equivalents of the Orthomodularity Law in Quantum Logic: Part 2

Jack K. Horner
P. O. Box 266
Los Alamos, New Mexico 87544 USA

FCS 2014

Abstract

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of quantum-mechanical behaviors and operations. In much the same way that the structure of conventional propositional (Boolean) logic (BL) is the logic of the description of the behavior of classical physical systems and is isomorphic to a Boolean algebra (BA), so also the algebra, C(H), of closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space is a logic of the descriptions of the behavior of quantum mechanical systems and is a model of an ortholattice (OL). An OL can thus be thought of as a kind of "quantum logic" (QL). C(H) is also a model of an orthomodular lattice, which is an OL conjoined with the orthomodularity axiom (OMA). The rationalization of the OMA as a claim proper to physics has proven problematic, motivating the question of whether the OMA and its equivalents are required in an adequate characterization of QL. Here I provide an automated deduction of the OMA from two quantum-identity-based equivalents of the OMA. The proofs may be novel.

Keywords: automated deduction, quantum computing, orthomodular lattice, Hilbert space

1.0 Introduction

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of the description of quantum-mechanical behaviors ([1], [17], [18], [20]). In much the same way that conventional propositional (Boolean) logic (BL,[12]) is the logical structure of description of the behavior of classical physical systems (e.g. "the measurements of the position and momentum of particle P are commutative", i.e., can be measured in either other, yielding the same results) and is isomorphic to a Boolean lattice ([10], [11], [19]), so also the algebra, *C(H)*, of the closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space *H* ([1], [4], [6], [9], [13]) is *a* logic of the descriptions of the behavior of quantum mechanical systems (e.g., "the measurements of the position and momentum of particle P are *not* commutative") and is a model ([10]) of an ortholattice (OL; [4]). An OL can thus be thought of as a kind of "quantum logic" (QL; [19]). *C(H)* is also a model of (i.e., isomorphic to a set of sentences which hold in) an orthomodular lattice (OML; [4], [7]), which is an OL conjoined with the orthomodularity axiom (OMA; see Figure 1). The rationalization of the OMA as a claim proper to physics has proven problematic ([13], Section 5-6), motivating the question of whether the OMA is required in an adequate characterization of QL. Thus formulated, the question suggests that the OMA and its equivalents are specific to an OML, and that as a consequence, banning the OMA from QL yields a "truer" quantum logic

.

Lattice axioms

```
      x = c(c(x))                       (AxLat1)
      x v y = y v x                     (AxLat2)
      (x v y) v z = x v (y v z)          (AxLat3)
      (x ^ y) ^ z = x ^ (y ^ z)          (AxLat4)
      x v (x ^ y) = x                   (AxLat5)
      x ^ (x v y) = x                   (AxLat6)
```

Ortholattice axioms

```
      c(x) ^ x = 0                      (AxOL1)
      c(x) v x = 1                      (AxOL2)
      x ^ y = c(c(x) v c(y))            (AxOL3)
```

Orthomodularity axiom

```
      y v (c(y) ^ (x v y)) = x v y      (OMA)
```

Definitions of implications and partial order

```
   i1(x,y) = c(x) v (x ^ y).
   i2(x,y) = i1(c(y), c(x).
   i3(x,y) = (c(x) ^ y) v (c(x) ^ c(y)) v i1(x,y).
   i4(x,y) = i3(c(y), c(x)).
   i5(x,y) = (x ^ y) v (c(x) ^ y) v (c(x) ^ c(y)).
   le(x,y) = (x = (x ^ y)).
```

Definitions of indexed identities

```
    id1(x,y) = i1(x,y) ^ i0(y, x)
    id2(x,y) = i2(x,y) ^ i0(y, x)
    id3(x,y) = i3(x,y) ^ i0(y, x)
    id4(x,y) = i4(x,y) ^ i0(y, x)
    id5(x,y) = i5(x,y) ^ i0(y, x)
```

```
where
   x, y are variables ranging over lattice nodes
   ^ is lattice meet
   v is lattice join
   c(x) is the orthocomplement of x
   i1(x,y) means x →1 y (Sasaki implication)
   i2(x,y) means x →2 y (Dishkant implication)
   i3(x,y) means x →3 y (Kalmbach implication)
   i4(x,y) means x →4 y (non-tollens implication)
   i5(x,y) means x →5 y (relevance implication)
   le(x,y) means x ≤ y
   <-> means if and only if
   =  is equivalence ([12])
   1 is the maximum lattice element (= x v c(x))
   0 is the minimum lattice element (= c(1))
```

Figure 1. Lattice, ortholattice, orthomodularity axioms, and some definitions.

There are at least 21 nominal equivalents (in the sense that ortholattice theory, together with these "equivalents", imply the OMA, and vice versa) of the OMA in quantum logic ([5], Theorem 2.5); as nominal equivalents, they are thus of import to optimizing quantum circuit design. Among these is the Proposition shown in Figure 2:

$$x \equiv_i y \quad <-> \quad x = y$$

where

$x \equiv_i y$ means $(x \to_i y) \wedge (x \to_0 y)$
$x \to_0 y$ means $c(x) \vee y$
$i = 1,2,3,4,5$

Figure 2. Proposition 2.13 of [5]

2.0 Method

The OML axiomatizations of Megill, Pavičić, and Horner ([5], [14], [15], [16], [21], [22]) were implemented in a *prover9* ([2]) script ([3]) configured to derive OMA from Proposition 2.13 of [5], for each of i = 4,5 together with ortholattice theory (orthomodular lattice theory, without the OMA), then executed in that framework on a Dell Inspiron 545 with an Intel Core2 Quad CPU Q8200 (clocked @ 2.33 GHz) and 8.00 GB RAM, running under the *Windows Vista Home Premium /Cygwin* operating environment.

3.0 Results

Figure 3 shows the proofs, generated by [3] on the platform described in Section 2.0, that Proposition 2.13 (for each of i = 4,5), together with ortholattice theory, imply the OMA.

```
============================== PROOF =================================

% Proof 1 at 0.09 (+ 0.05) seconds: "OMA".
% Length of proof is 46.
% Level of proof is 12.

4 id4(x,y) = 1 <-> x = y # label("Hypothesis for Proposition 2.10id4") #
label(non_clause).  [assumption].
5 y v (c(y) ^ (x v y)) = x v y # label("OMA") # label(non_clause) # label(goal).  [goal].
12 x = c(c(x)) # label("AxL1").  [assumption].
13 c(c(x)) = x.  [copy(12),flip(a)].
14 x v y = y v x # label("AxL2").  [assumption].
15 (x v y) v z = x v (y v z) # label("AxL3").  [assumption].
17 x v (x ^ y) = x # label("AxL5").  [assumption].
18 x ^ (x v y) = x # label("AxL6").  [assumption].
19 c(x) ^ x = 0 # label("AxOL1").  [assumption].
20 c(x) v x = 1 # label("AxOL2").  [assumption].
21 x v c(x) = 1.  [copy(20),rewrite([14(2)])].
22 x ^ y = c(c(x) v c(y)) # label("AxOL3").  [assumption].
23 i0(x,y) = c(x) v y # label("Df: i0").  [assumption].
30 i4(x,y) = ((c(c(y)) ^ c(x)) v (c(c(y)) ^ c(c(x)))) v (c(c(y)) v (c(y) ^ c(x))) #
label("Df: i4").  [assumption].
31 i4(x,y) = y v (c(y v x) v (c(c(y) v x) v c(c(y) v c(x)))).
[copy(30),rewrite([13(3),22(3),13(4),13(6),13(6),22(5),13(11),22(12),13(11),13(11),14(13)
,15(13)])].
60 id4(x,y) = i4(x,y) ^ i0(y,x) # label("Df: id4").  [assumption].
61 id4(x,y) = c(c(c(y) v x) v c(y v (c(y v x) v (c(c(y) v x) v c(c(y) v c(x)))))).
[copy(60),rewrite([31(2),23(14),22(16),14(18)])].
64 id4(x,y) != 1 | y = x # label("Hypothesis for Proposition 2.10id4").  [clausify(4)].
65 c(c(c(x) v y) v c(x v (c(x v y) v (c(c(x) v y) v c(c(x) v c(y)))))) != 1 | x = y.
[copy(64),rewrite([61(1)])].
```

```
68 c1 v (c(c1) ^ (c2 v c1)) != c2 v c1 # label("OMA") # answer("OMA").  [deny(5)].
69 c1 v c(c1 v c(c1 v c2)) != c1 v c2 # answer("OMA").
[copy(68),rewrite([14(6),22(7),13(4),14(12)])].
70 c(1) = 0.  [back_rewrite(19),rewrite([22(2),13(2),21(2)])].
71 c(c(x) v c(x v y)) = x.  [back_rewrite(18),rewrite([22(2)])].
72 x v c(c(x) v c(y)) = x.  [back_rewrite(17),rewrite([22(1)])].
74 x v (y v z) = y v (x v z).  [para(14(a,1),15(a,1,1)),rewrite([15(2)])].
81 x v (c(x) v y) = 1 v y.  [para(21(a,1),15(a,1,1)),flip(a)].
82 x v (y v c(x v y)) = 1.  [para(21(a,1),15(a,1)),flip(a)].
103 c(0 v c(0 v (c1 v (c2 v (c(c1 v (c1 v (c2 v c(c1 v c(c1 v c2))))) v c(c(c1 v c2) v
c(c1 v c(c1 v c(c1 v c2))))))))) != 1 # answer("OMA").
[ur(65,b,69,a(flip)),rewrite([74(14),82(14),70(2),74(17),15(16),74(32),82(32),70(20),74(3
7),74(38),15(37)])].
114 c(0 v c(x)) = x.  [para(21(a,1),71(a,1,1,2,1)),rewrite([70(3),14(3)])].
118 1 v x = 1.  [para(70(a,1),71(a,1,1,1)),rewrite([114(6)])].
122 0 v (c1 v (c2 v (c(c1 v (c1 v (c2 v c(c1 v c(c1 v c2))))) v c(c(c1 v c2) v c(c1 v
c(c1 v c(c1 v c2)))))) != 1 # answer("OMA").  [back_rewrite(103),rewrite([114(41)])].
134 x v (c(x) v y) = 1.  [back_rewrite(81),rewrite([118(5)])].
137 x v c(c(x) v y) = x.  [para(13(a,1),72(a,1,2,1,2))].
141 x v 0 = x.  [para(21(a,1),72(a,1,2,1)),rewrite([70(2)])].
144 x v x = x.  [para(70(a,1),72(a,1,2,1,2)),rewrite([14(3),114(4)])].
156 0 v (0 v (c1 v (c2 v (c(c1 v (c1 v (c2 v c(c1 v c(c1 v c2))))) v c(c(c1 v c2) v c(c1
v c(c1 v c(c1 v c2)))))))) != 1 # answer("OMA").
[ur(65,b,122,a(flip)),rewrite([70(2),118(80),70(43),70(44),70(84),114(123),14(120),15(120
),15(119),15(118),15(117),118(122),70(42),14(42),114(43)])].
157 0 v (0 v (0 v (0 v (c1 v (c2 v (c(c1 v (c1 v (c2 v c(c1 v c(c1 v c2))))) v c(c(c1 v
c2) v c(c1 v c(c1 v c(c1 v c2)))))))))) != 1 # answer("OMA").
[ur(65,b,156,a),rewrite([14(42),118(42),70(2),14(81),118(81),70(42),14(83),118(83),70(43)
,70(84),14(84),114(85),74(84),74(83),144(82),114(47)])].
159 0 v (0 v (0 v (0 v (0 v (0 v (c1 v (c2 v (c(c1 v (c1 v (c2 v c(c1 v c(c1 v c2))))) v
c(c(c1 v c2) v c(c1 v c(c1 v c(c1 v c2)))))))))) != 1 # answer("OMA").
[ur(65,b,157,a),rewrite([14(46),118(46),70(2),14(89),118(89),70(46),14(91),118(91),70(47)
,70(92),14(92),114(93),74(92),74(91),144(90),114(51)])].
163 0 v x = x.  [para(141(a,1),14(a,1)),flip(a)].
164 c1 v (c2 v (c(c1 v (c1 v (c2 v c(c1 v c(c1 v c2))))) v c(c(c1 v c2) v c(c1 v c(c1 v
c(c1 v c2)))))) != 1 # answer("OMA").
[back_rewrite(159),rewrite([163(42),163(41),163(40),163(39),163(38),163(37)])].
175 x v (x v y) = x v y.  [para(144(a,1),15(a,1,1)),flip(a)].
180 c1 v (c2 v (c(c1 v (c2 v c(c1 v c(c1 v c2)))) v c(c(c1 v c2) v c(c1 v c(c1 v c(c1 v
c2)))))) != 1 # answer("OMA").  [back_rewrite(164),rewrite([175(15)])].
185 x v (y v (c(x v y) v z)) = 1.  [para(134(a,1),15(a,1)),flip(a)].
198 x v c(y v c(x)) = x.  [para(14(a,1),137(a,1,2,1))].
215 x v (y v c(z v c(x v y))) = x v y.  [para(198(a,1),15(a,1)),flip(a)].
221 $F # answer("OMA").  [back_rewrite(180),rewrite([215(13),185(25)]),xx(a)].

============================== end of proof ==========================


============================== PROOF =================================

% Proof 1 at 1.33 (+ 0.03) seconds: "OMA".
% Length of proof is 50.
% Level of proof is 9.

4 id5(x,y) = 1 <-> x = y # label("Hypothesis for Proposition 2.10id5") #
label(non_clause).  [assumption].
5 y v (c(y) ^ (x v y)) = x v y # label("OMA") # label(non_clause) # label(goal).  [goal].
12 x = c(c(x)) # label("AxL1").  [assumption].
13 c(c(x)) = x.  [copy(12),flip(a)].
14 x v y = y v x # label("AxL2").  [assumption].
15 (x v y) v z = x v (y v z) # label("AxL3").  [assumption].
17 x v (x ^ y) = x # label("AxL5").  [assumption].
18 x ^ (x v y) = x # label("AxL6").  [assumption].
19 c(x) ^ x = 0 # label("AxOL1").  [assumption].
20 c(x) v x = 1 # label("AxOL2").  [assumption].
21 x v c(x) = 1.  [copy(20),rewrite([14(2)])].
22 x ^ y = c(c(x) v c(y)) # label("AxOL3").  [assumption].
23 i0(x,y) = c(x) v y # label("Df: i0").  [assumption].
32 i5(x,y) = ((x ^ y) v (c(x) ^ y)) v (c(x) ^ c(y)) # label("Df: i5").  [assumption].
33 i5(x,y) = c(x v y) v (c(x v c(y)) v c(c(x) v c(y))).
[copy(32),rewrite([22(2),22(7),13(7),14(9),22(12),13(11),13(11),14(12)])].
```

```
62 id5(x,y) = i5(x,y) ^ i0(y,x) # label("Df: id5").  [assumption].
63 id5(x,y) = c(c(c(y) v x) v c(c(x v y) v (c(x v c(y)) v c(c(x) v c(y))))).
[copy(62),rewrite([33(2),23(13),22(15),14(17)])].
64 id5(x,y) != 1 | y = x # label("Hypothesis for Proposition 2.10id5").  [clausify(4)].
65 c(c(c(x) v y) v c(c(y v x) v (c(y v c(x)) v c(c(y) v c(x))))) != 1 | x = y.
[copy(64),rewrite([63(1)])].
68 c1 v (c(c1) ^ (c2 v c1)) != c2 v c1 # label("OMA") # answer("OMA").  [deny(5)].
69 c1 v c(c1 v c(c1 v c2)) != c1 v c2 # answer("OMA").
[copy(68),rewrite([14(6),22(7),13(4),14(12)])].
70 c(1) = 0.  [back_rewrite(19),rewrite([22(2),13(2),21(2)])].
71 c(c(x) v c(x v y)) = x.  [back_rewrite(18),rewrite([22(2)])].
72 x v c(c(x) v c(y)) = x.  [back_rewrite(17),rewrite([22(1)])].
74 x v (y v z) = y v (x v z).  [para(14(a,1),15(a,1,1)),rewrite([15(2)])].
82 x v (y v c(x v y)) = 1.  [para(21(a,1),15(a,1)),flip(a)].
105 c(c(c1 v (c2 v c(c1 v c(c1 v c(c1 v c2))))) v c(c(c1 v (c1 v (c2 v c(c1 v c(c1 v
c2))))) v (c(c1 v (c2 v c(c1 v c(c1 v c(c1 v c2))))) v c(c(c1 v c2) v c(c1 v c(c1 v c(c1
v c2)))))) != 1 # answer("OMA").
[ur(65,b,69,a),rewrite([14(14),15(14),74(28),15(27),15(43)])].
108 c(0) = 1.  [para(70(a,1),13(a,1,1))].
113 c(x) v c(x v y) = c(x).  [para(71(a,1),13(a,1,1)),flip(a)].
117 c(0 v c(x)) = x.  [para(21(a,1),71(a,1,1,2,1)),rewrite([70(3),14(3)])].
140 x v c(c(x) v y) = x.  [para(13(a,1),72(a,1,2,1,2))].
144 x v 0 = x.  [para(21(a,1),72(a,1,2,1)),rewrite([70(2)])].
147 x v x = x.  [para(70(a,1),72(a,1,2,1,2)),rewrite([14(3),117(4)])].
172 0 v x = x.  [para(144(a,1),14(a,1)),flip(a)].
190 x v (x v y) = x v y.  [para(147(a,1),15(a,1,1)),flip(a)].
192 x v (y v x) = y v x.  [para(147(a,1),15(a,2,2)),rewrite([14(2)])].
202 c(c(c1 v (c2 v c(c1 v c(c1 v c(c1 v c2))))) v c(c(c1 v (c2 v c(c1 v c(c1 v c2)))) v
(c(c1 v (c2 v c(c1 v c(c1 v c(c1 v c2))))) v c(c(c1 v c2) v c(c1 v c(c1 v c(c1 v
c2)))))) != 1 # answer("OMA").  [back_rewrite(105),rewrite([190(28)])].
217 x v c(y v c(x)) = x.  [para(14(a,1),140(a,1,2,1))].
221 x v (y v c(c(x) v z)) = y v x.  [para(140(a,1),74(a,1,2)),flip(a)].
233 c(x) v c(y v x) = c(x).  [para(13(a,1),217(a,1,2,1,2))].
235 x v (y v c(z v c(x v y))) = x v y.  [para(217(a,1),15(a,1)),flip(a)].
240 c(c(c1 v (c2 v c(c1 v c(c1 v c(c1 v c2))))) v c(c(c1 v c2) v (c(c1 v (c2 v c(c1 v
c(c1 v c(c1 v c2))))) v c(c(c1 v c2) v c(c1 v c(c1 v c(c1 v c2)))))) != 1 #
answer("OMA").  [back_rewrite(202),rewrite([235(26)])].
243 c(x v y) v c(c(y) v c(x v c(c(y) v x))) != 1 | x v c(c(y) v x) = y.
[para(82(a,1),65(a,1,1,1,1)),rewrite([70(2),14(6),221(6),14(9),82(9),70(5),14(11),172(13)
,172(14),13(13)]),flip(b)].
249 c(x) v (c(x v y) v z) = c(x) v z.  [para(113(a,1),15(a,1,1)),flip(a)].
990 c(x v y) v (c(x v (y v z)) v u) = c(x v y) v u.  [para(15(a,1),249(a,1,2,1,1))].
1023 c(c(c1 v (c2 v c(c1 v c(c1 v c(c1 v c2))))) v c(c(c1 v c2) v c(c(c1 v c2) v c(c1 v
c(c1 v c(c1 v c2)))))) != 1 # answer("OMA").  [back_rewrite(240),rewrite([990(52)])].
1824 c(x v y) v c(y v c(z v c(x v y))) = c(y v c(z v c(x v y))).
[para(235(a,1),233(a,1,2,1)),rewrite([14(9)])].
2221 x v c(x v c(y v x)) = y v x.
[para(192(a,1),243(a,1,1,1)),rewrite([14(7),1824(11),13(9),74(8),82(8),14(6)]),xx(a)].
2315 x v c(x v c(x v y)) = y v x.  [para(14(a,1),2221(a,1,2,1,2,1))].
2375 $F # answer("OMA").
[back_rewrite(1023),rewrite([2315(11),14(5),82(8),70(2),2315(18),14(12),147(14),13(10),14
(9),21(9),70(3),147(3),108(2)]),xx(a)].

============================== end of proof ==========================
```

Figure 3. Summary of a *prover9* ([2]) proof of Proposition 2.11, for each of i = 4,5. The proofs assume the default inference rules of *prover9*. The general form of a line in this proof is "*line_number conclusion [derivation]*", where *line_number* is a unique identifier of a line in the proof, and *conclusion* is the result of applying the *prover9* inference rules (such as *para*modulation, *copy*ing, and *rewrit*ing), noted in square brackets (denoting the *derivation*), to the lines cited in those brackets. Note that some of "logical" proof lines in the above have been transformed to two text lines, with the *derivation* appearing on a text line following a text line containing the first part of that logical line. The detailed syntax and semantics of these notations can be found in [2]. All *prover9* proofs are by default proofs by contradiction.

The total time to produce the proofs in Figure 3 on the platform described in Section 2.0 was approximately 1.5seconds.

4.0 Discussion

The results of Section 3.0 motivate several observations:

1. Each of the proofs in Figure 3 use L1, L2, L3, OL1, OL2, L3, L5, L6, and OL3.

2. The proofs in Section 3.0 may be novel.

3. Companion papers provide proofs for i = 1,2,3, and show that orthomodular lattice theory implies Propositions 2.13i, i = 1,2,3,4,5.

4. Proposition 2.13 can be regarded as a definition of quantum identity; thus, this paper together with papers mentioned in (3), constitute a proof that the definition of quantum intersection is equivalent to the OMA in orthomodular quantum logic. Companion papers derive equivalences for the OMA with definitions of quantum-intersection and quantum-union. Collectively, these papers provide a theory of equivalence of the OMA with the quantum connectives. In light of these equivalences, QL without the OMA would hardly qualify as a logic.

5.0 Acknowledgements

This work benefited from discussions with Tom Oberdan, Frank Pecchioni, Tony Pawlicki, and the late John K. Prentice, whose passion for foundations of physics inspired those of us privileged to have known him. For any infelicities that remain, I am solely responsible.

6.0 References

[1] von Neumann J. *Mathematical Foundations of Quantum Mechanics*. 1936. Translated by R. T. Beyer. Princeton. 1983.
[2] McCune WW. *prover9 and mace4.* URL http://www.cs.unm.edu/~mccune/prover9/. 2009.
[3] Horner JK. MHT *prover9* scripts. 2011. Available from the author on request.
[4] Dalla Chiara ML and Giuntini R. *Quantum Logics*. URL http://xxx.lanl.gov/abs/quant-ph/0101028. 2004.
[5] Megill ND and Pavičić M. Orthomodular lattices and quantum algebra. *International Journal of Theoretical Physics* 40 (2001), pp. 1387-1410.
[6] Akhiezer NI and Glazman IM. *Theory of Linear Operators in Hilbert Space. Volume I.* Translated by M. Nestell. Frederick Ungar. 1961.
[7] Holland, Jr. SS Orthomodularity in infinite dimensions: a theorem of M. Solèr. *Bulletin of the American Mathematical Society* 32 (1995), pp. 205-234.
[8] Marsden EL and Herman LM. *A condition for distribution in orthomodular lattices.* Kansas State University Technical Report #40. 1974.
[9] Knuth DE and Bendix PB. Simple word problems in universal algebras. In J. Leech, ed. *Computational Problems in Abstract Algebra.* Pergamon Press. 1970. pp. 263-297.
[10] Chang CC and Keisler HJ. *Model Theory.* North-Holland. 1990. pp. 38-39.
[11] Birkhoff G. *Lattice Theory.* Third Edition. American Mathematical Society. 1967.
[12] Church A. *Introduction to Mathematical Logic. Volume I.* Princeton. 1956.
[13] Jauch J. *Foundations of Quantum Mechanics.* Addison-Wesley. 1968.

[14] Megill ND. *Metamath.* URL http://us.metamath.org/qlegif/mmql.html#unify. 2004.
[15] Horner JK. An automated deduction system for orthomodular lattice theory. *Proceedings of the 2005 International Conference on Artificial Intelligence.* CSREA Press. 2005. pp. 260-265.
[16] Horner JK. An automated equational logic deduction of join elimination in orthomodular lattice theory. *Proceedings of the 2007 International Conference on Artificial Intelligence.* CSREA Press. 2007. pp. 481-488.
[17] Messiah A. *Quantum Mechanics*. Dover. 1958.
[18] Horner JK. Using automated theorem-provers to aid the design of efficient compilers for quantum computing. *Los Alamos National Laboratory Quantum Institute Workshop.* December 9–10, 2002. URL http://www.lanl.gov/science/centers/quantum/qls_pdfs/horner.pdf.
[19] Birkhoff G and von Neumann J. The logic of quantum mechanics. *Annals of Mathematics* 37 (1936), 823-243.
[20] Nielsen MA and Chuang L . *Quantum Computation and Quantum Information*. Cambridge. 2000.
[21] Pavičić M and Megill N. Quantum and classical implicational algebras with primitive implication. *International Journal of Theoretical Physics* 37 (1998), 2091-2098. ftp://m3k.grad.hr/pavicic/quantum-logic/1998-int-j-theor-phys-2.ps.gz.
[22] Horner JK. An automated deduction of the relative strength of orthomodular and weakly orthomodular lattice theory. *Proceedings of the 2009 International Conference on Artificial Intelligence*. CSREA Press. 2009. pp. 525-530.

Quantum-Identity Equivalents of the Orthomodularity Law in Quantum Logic: Part 3

Jack K. Horner
P. O. Box 266
Los Alamos, New Mexico 87544 USA
jhorner@cybermesa.com

FCS 2014

Abstract

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of quantum-mechanical behaviors and operations. In much the same way that the structure of conventional propositional (Boolean) logic (BL) is the logic of the description of the behavior of classical physical systems and is isomorphic to a Boolean algebra (BA), so also the algebra, C(H), of closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space is a logic of the descriptions of the behavior of quantum mechanical systems and is a model of an ortholattice (OL). An OL can thus be thought of as a kind of "quantum logic" (QL). C(H) is also a model of an orthomodular lattice, which is an OL conjoined with the orthomodularity axiom (OMA). The rationalization of the OMA as a claim proper to physics has proven problematic, motivating the question of whether the OMA and its equivalents are required in an adequate characterization of QL. Here I provide an automated deduction of two quantum-identity-based equivalents of the OMA. The proofs may be novel.

Keywords: automated deduction, quantum computing, orthomodular lattice, Hilbert space

1.0 Introduction

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of the description of quantum-mechanical behaviors ([1], [17], [18], [20]). In much the same way that conventional propositional (Boolean) logic (BL,[12]) is the logical structure of description of the behavior of classical physical systems (e.g. "the measurements of the position and momentum of particle P are commutative", i.e., can be measured in either other, yielding the same results) and is isomorphic to a Boolean lattice ([10], [11], [19]), so also the algebra, *C(H)*, of the closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space *H* ([1], [4], [6], [9], [13]) is *a* logic of the descriptions of the behavior of quantum mechanical systems (e.g., "the measurements of the position and momentum of particle P are *not* commutative") and is a model ([10]) of an ortholattice (OL; [4]). An OL can thus be thought of as a kind of "quantum logic" (QL; [19]). *C(H)* is also a model of (i.e., isomorphic to a set of sentences which hold in) an orthomodular lattice (OML; [4], [7]), which is an OL conjoined with the orthomodularity axiom (OMA; see Figure 1). The rationalization of the OMA as a claim proper to physics has proven problematic ([13], Section 5-6), motivating the question of whether the OMA is required in an adequate characterization of QL. Thus formulated, the question suggests that the OMA and its equivalents are specific to an OML, and that as a consequence, banning the OMA from QL yields a "truer" quantum logic.

Lattice axioms

```
      x = c(c(x))                       (AxLat1)
      x v y = y v x                     (AxLat2)
      (x v y) v z = x v (y v z)         (AxLat3)
      (x ^ y) ^ z = x ^ (y ^ z)         (AxLat4)
      x v (x ^ y) = x                   (AxLat5)
      x ^ (x v y) = x                   (AxLat6)
```

Ortholattice axioms

```
      c(x) ^ x = 0                      (AxOL1)
      c(x) v x = 1                      (AxOL2)
      x ^ y = c(c(x) v c(y))            (AxOL3)
```

Orthomodularity axiom

```
      y v (c(y) ^ (x v y)) = x v y      (OMA)
```

Definitions of implications and partial order

```
   i1(x,y) = c(x) v (x ^ y).
   i2(x,y) = i1(c(y), c(x).
   i3(x,y) = (c(x) ^ y) v (c(x) ^ c(y)) v i1(x,y).
   i4(x,y) = i3(c(y), c(x)).
   i5(x,y) = (x ^ y) v (c(x) ^ y) v (c(x) ^ c(y)).
   le(x,y) = (x = (x ^ y)).
```

Definitions of indexed identities

```
    id1(x,y) = i1(x,y) ^ i0(y, x)
    id2(x,y) = i2(x,y) ^ i0(y, x)
    id3(x,y) = i3(x,y) ^ i0(y, x)
    id4(x,y) = i4(x,y) ^ i0(y, x)
    id5(x,y) = i5(x,y) ^ i0(y, x)
```

where
- x, y are variables ranging over lattice nodes
- ^ is lattice meet
- v is lattice join
- c(x) is the orthocomplement of x
- i1(x,y) means $x \rightarrow_1 y$ (Sasaki implication)
- i2(x,y) means $x \rightarrow_2 y$ (Dishkant implication)
- i3(x,y) means $x \rightarrow_3 y$ (Kalmbach implication)
- i4(x,y) means $x \rightarrow_4 y$ (non-tollens implication)
- i5(x,y) means $x \rightarrow_5 y$ (relevance implication)
- le(x,y) means $x \leq y$
- <-> means if and only if
- = is equivalence ([12])
- 1 is the maximum lattice element (= x v c(x))
- 0 is the minimum lattice element (= c(1))

Figure 1. Lattice, ortholattice, orthomodularity axioms, and some definitions.

There are at least 21 nominal equivalents (in the sense that ortholattice theory, together with these "equivalents", imply the OMA, and vice versa) of the OMA in quantum logic ([5], Theorem 2.5); as nominal equivalents, they are thus of import to optimizing quantum circuit design. Among these is the Proposition shown in Figure 2:

$$x \equiv_i y \quad <-> \quad x = y$$

where

$x \equiv_i y$ means $(x \to_i y) \wedge (x \to_0 y)$
$x \to_0 y$ means $c(x) \vee y$
$i = 1,2,3,4,5$

Figure 2. Proposition 2.13 of [5]

2.0 Method

The OML axiomatizations of Megill, Pavičić, and Horner ([5], [14], [15], [16], [21], [22]) were implemented in a *prover9* ([2]) script ([3]) configured to derive Proposition 2.13 of [5], for each of i = 1,2, from orthomodular lattice theory, then executed in that framework on a Dell Inspiron 545 with an Intel Core2 Quad CPU Q8200 (clocked @ 2.33 GHz) and 8.00 GB RAM, running under the *Windows Vista Home Premium /Cygwin* operating environment.

3.0 Results

Figure 3 shows the proofs, generated by [3] on the platform described in Section 2.0, that orthomodular lattice theory implies Proposition 2.13 of [5] (for each of i = 1,2).

```
============================== PROOF =================================

% Proof 1 at 10.62 (+ 0.06) seconds.
% Length of proof is 45.
% Level of proof is 10.

4 id1(x,y) = 1 <-> x = y # label("Proposition 2.10id1") # label(non_clause) #
label(goal).  [goal].
11 x = c(c(x)) # label("AxL1").  [assumption].
12 c(c(x)) = x.  [copy(11),flip(a)].
13 x v y = y v x # label("AxL2").  [assumption].
14 (x v y) v z = x v (y v z) # label("AxL3").  [assumption].
16 x v (x ^ y) = x # label("AxL5").  [assumption].
17 x ^ (x v y) = x # label("AxL6").  [assumption].
18 c(x) ^ x = 0 # label("AxOL1").  [assumption].
19 c(x) v x = 1 # label("AxOL2").  [assumption].
20 x v c(x) = 1.  [copy(19),rewrite([13(2)])].
21 x ^ y = c(c(x) v c(y)) # label("AxOL3").  [assumption].
22 x v (c(x) ^ (y v x)) = y v x # label("OMA").  [assumption].
23 x v c(x v c(y v x)) = y v x.  [copy(22),rewrite([21(3),12(2)])].
24 i0(x,y) = c(x) v y # label("Df: i0").  [assumption].
25 i1(x,y) = c(x) v (x ^ y) # label("Df: i1").  [assumption].
26 i1(x,y) = c(x) v c(c(x) v c(y)).  [copy(25),rewrite([21(3)])].
55 id1(x,y) = i1(x,y) ^ i0(y,x) # label("Df: id1").  [assumption].
56 id1(x,y) = c(c(c(y) v x) v c(c(x) v c(c(x) v c(y)))).
[copy(55),rewrite([26(2),24(8),21(10),13(12)])].
65 id1(c1,c2) = 1 | c2 = c1 # label("Proposition 2.10id1").  [deny(4)].
66 c(c(c1 v c(c2)) v c(c(c1) v c(c(c1) v c(c2)))) = 1 | c2 = c1.
[copy(65),rewrite([56(3),13(4)])].
67 id1(c1,c2) != 1 | c2 != c1 # label("Proposition 2.10id1").  [deny(4)].
68 c(c(c1 v c(c2)) v c(c(c1) v c(c(c1) v c(c2)))) != 1 | c2 != c1.
[copy(67),rewrite([56(3),13(4)])].
69 c(1) = 0.  [back_rewrite(18),rewrite([21(2),12(2),20(2)])].
70 c(c(x) v c(x v y)) = x.  [back_rewrite(17),rewrite([21(2)])].
71 x v c(c(x) v c(y)) = x.  [back_rewrite(16),rewrite([21(1)])].
82 x v c(x v c(x v y)) = y v x.  [para(13(a,1),23(a,1,2,1,2,1))].
```

```
88 c2 = c1 | c(c1 v c(c2)) v c(c(c1) v c(c(c1) v c(c2))) = 0.
[para(66(a,1),12(a,1,1)),rewrite([69(5)]),flip(b)].
108 c(0) = 1.  [para(69(a,1),12(a,1,1))].
110 c(x) v c(x v y) = c(x).  [para(70(a,1),12(a,1,1)),flip(a)].
114 c(0 v c(x)) = x.  [para(20(a,1),70(a,1,1,2,1)),rewrite([69(3),13(3)])].
115 c(x v y) v c(x v c(x v y)) = c(x).
[para(70(a,1),23(a,1,2,1,2)),rewrite([13(5),110(11)])].
118 1 v x = 1.  [para(69(a,1),70(a,1,1,1)),rewrite([114(6)])].
143 x v 0 = x.  [para(20(a,1),71(a,1,2,1)),rewrite([69(2)])].
144 x v c(y v c(x)) = x.  [para(23(a,1),71(a,1,2,1))].
146 c2 = c1 | c1 v c(c2) = 1.  [para(66(a,1),71(a,1,2)),rewrite([13(9),118(9)]),flip(b)].
147 x v x = x.  [para(69(a,1),71(a,1,2,1,2)),rewrite([13(3),114(4)])].
154 0 v x = x.  [para(143(a,1),13(a,1)),flip(a)].
185 x v (c(y v c(x)) v z) = x v z.  [para(144(a,1),14(a,1,1)),flip(a)].
807 c2 = c1 | c2 v c(c(c1) v c(c(c1) v c(c2))) = c2.
[para(88(b,1),185(a,1,2)),rewrite([13(6),154(6)]),flip(b)].
811 x v c(y v c(y v c(x))) = x v c(y).  [para(115(a,1),185(a,1,2)),flip(a)].
822 c2 = c1 | c1 v c2 = c2.  [back_rewrite(807),rewrite([811(15),12(7),13(6)])].
833 c2 = c1 | c1 v c(c1 v c(c2)) = c1 v c2.
[para(822(b,1),82(a,1,2,1,2,1)),rewrite([13(13)])].
6066 c2 = c1 | c1 v c2 = c1.
[para(146(b,1),833(b,1,2,1)),rewrite([69(9),13(9),154(9)]),flip(c),merge(b)].
6286 c2 = c1.  [para(6066(b,1),822(b,1)),flip(c),merge(b),merge(c)].
6287 $F.
[back_rewrite(68),rewrite([6286(2),20(4),69(2),6286(6),147(8),12(6),13(5),20(5),69(3),147
(3),108(2),6286(4)]),xx(a),xx(b)].

============================== end of proof ==========================



============================== PROOF =================================

% Proof 1 at 0.11 (+ 0.01) seconds: "OMA".
% Length of proof is 36.
% Level of proof is 8.

4 id2(x,y) = 1 <-> x = y # label("Hypothesis for Proposition 2.10id2") #
label(non_clause).  [assumption].
5 y v (c(y) ^ (x v y)) = x v y # label("OMA") # label(non_clause) # label(goal).  [goal].
12 x = c(c(x)) # label("AxL1").  [assumption].
13 c(c(x)) = x.  [copy(12),flip(a)].
14 x v y = y v x # label("AxL2").  [assumption].
15 (x v y) v z = x v (y v z) # label("AxL3").  [assumption].
17 x v (x ^ y) = x # label("AxL5").  [assumption].
18 x ^ (x v y) = x # label("AxL6").  [assumption].
19 c(x) ^ x = 0 # label("AxOL1").  [assumption].
20 c(x) v x = 1 # label("AxOL2").  [assumption].
21 x v c(x) = 1.  [copy(20),rewrite([14(2)])].
22 x ^ y = c(c(x) v c(y)) # label("AxOL3").  [assumption].
23 i0(x,y) = c(x) v y # label("Df: i0").  [assumption].
26 i2(x,y) = c(c(y)) v (c(y) ^ c(x)) # label("Df: i2").  [assumption].
27 i2(x,y) = y v c(y v x).  [copy(26),rewrite([13(3),22(4),13(3),13(3)])].
56 id2(x,y) = i2(x,y) ^ i0(y,x) # label("Df: id2").  [assumption].
57 id2(x,y) = c(c(c(y) v x) v c(y v c(y v x))).
[copy(56),rewrite([27(2),23(5),22(7),14(9)])].
64 id2(x,y) != 1 | y = x # label("Hypothesis for Proposition 2.10id2").  [clausify(4)].
65 c(c(c(x) v y) v c(x v c(x v y))) != 1 | x = y.  [copy(64),rewrite([57(1)])].
68 c1 v (c(c1) ^ (c2 v c1)) != c2 v c1 # label("OMA") # answer("OMA").  [deny(5)].
69 c1 v c(c1 v c(c1 v c2)) != c1 v c2 # answer("OMA").
[copy(68),rewrite([14(6),22(7),13(4),14(12)])].
70 c(1) = 0.  [back_rewrite(19),rewrite([22(2),13(2),21(2)])].
71 c(c(x) v c(x v y)) = x.  [back_rewrite(18),rewrite([22(2)])].
72 x v c(c(x) v c(y)) = x.  [back_rewrite(17),rewrite([22(1)])].
74 x v (y v z) = y v (x v z).  [para(14(a,1),15(a,1,1)),rewrite([15(2)])].
82 x v (y v c(x v y)) = 1.  [para(21(a,1),15(a,1)),flip(a)].
106 c(0 v c(c1 v (c2 v c(c1 v (c1 v (c2 v c(c1 v c(c1 v c2))))))) != 1 # answer("OMA").
[ur(65,b,69,a(flip)),rewrite([74(14),82(14),70(2),74(17),15(16),15(19)])].
114 c(0 v c(x)) = x.  [para(21(a,1),71(a,1,1,2,1)),rewrite([70(3),14(3)])].
119 c1 v (c2 v c(c1 v (c1 v (c2 v c(c1 v c(c1 v c2)))))) != 1 # answer("OMA").
[back_rewrite(106),rewrite([114(22)])].
```

```
130 x v c(c(x) v y) = x.  [para(13(a,1),72(a,1,2,1,2))].
136 x v x = x.  [para(70(a,1),72(a,1,2,1,2)),rewrite([14(3),114(4)])].
156 x v (x v y) = x v y.  [para(136(a,1),15(a,1,1)),flip(a)].
164 c1 v (c2 v c(c1 v (c2 v c(c1 v c(c1 v c2))))) != 1 # answer("OMA").
[back_rewrite(119),rewrite([156(15)])].
181 x v c(y v c(x)) = x.  [para(14(a,1),130(a,1,2,1))].
197 x v (y v c(z v c(x v y))) = x v y.  [para(181(a,1),15(a,1)),flip(a)].
203 $F # answer("OMA").  [back_rewrite(164),rewrite([197(13),82(8)]),xx(a)].

============================== end of proof ==========================
```

Figure 3. Summary of a *prover9* ([2]) proof of Proposition 2.13 from orthomodular lattice theory, for each of i = 1,2. The proofs assume the default inference rules of *prover9*. The general form of a line in this proof is "*line_number conclusion [derivation]*", where *line_number* is a unique identifier of a line in the proof, and *conclusion* is the result of applying the *prover9* inference rules (such as *para*modulation, *copy*ing, and *rewrit*ing), noted in square brackets (denoting the *derivation*), to the lines cited in those brackets. Note that some of "logical" proof lines in the above have been transformed to two text lines, with the *derivation* appearing on a text line following a text line containing the first part of that logical line. The detailed syntax and semantics of these notations can be found in [2]. All *prover9* proofs are by default proofs by contradiction.

The total time to produce the proofs in Figure 3 on the platform described in Section 2.0 was approximately 2 seconds.

4.0 Discussion

The results of Section 3.0 motivate several observations:

1. The proofs in Figure 3 use L1, L2, L3, L5, L6, OL1, OL2, and OL3.

2. The proofs in Section 3.0 may be novel.

3. Companion papers provide proofs for i = 4,5, and for orthomodular lattice theory implies Propositions 2.13i, i = 1,2,3,4,5.

4. Proposition 2.13 can be regarded as a definition of quantum identities; thus, this paper together with papers mentioned in (3), constitute a proof that the definition of quantum intersection is equivalent to the OMA in orthomodular quantum logic. Companion papers derive equivalences for the OMA with definitions of quantum-intersection and quantum-union. Collectively, these papers provide a theory of equivalence of the OMA with the quantum connectives. In light of these equivalences, QL without the OMA would hardly qualify as a logic.

5.0 Acknowledgements

This work benefited from discussions with Tom Oberdan, Frank Pecchioni, Tony Pawlicki, and the late John K. Prentice, whose passion for foundations of physics inspired those of us privileged to have known him. For any infelicities that remain, I am solely responsible.

6.0 References

[1] von Neumann J. *Mathematical Foundations of Quantum Mechanics*. 1936. Translated by R. T. Beyer. Princeton. 1983.

[2] McCune WW. *prover9 and mace4.* URL http://www.cs.unm.edu/~mccune/prover9/. 2009.

[3] Horner JK. MHT *prover9* scripts. 2011. Available from the author on request.

[4] Dalla Chiara ML and Giuntini R. *Quantum Logics*. URL http://xxx.lanl.gov/abs/quant-ph/0101028. 2004.

[5] Megill ND and Pavičić M. Orthomodular lattices and quantum algebra. *International Journal of Theoretical Physics* 40 (2001), pp. 1387-1410.

[6] Akhiezer NI and Glazman IM. *Theory of Linear Operators in Hilbert Space. Volume I*. Translated by M. Nestell. Frederick Ungar. 1961.

[7] Holland, Jr. SS Orthomodularity in infinite dimensions: a theorem of M. Solèr. *Bulletin of the American Mathematical Society* 32 (1995), pp. 205-234.

[8] Marsden EL and Herman LM. *A condition for distribution in orthomodular lattices.* Kansas State University Technical Report #40. 1974.

[9] Knuth DE and Bendix PB. Simple word problems in universal algebras. In J. Leech, ed. *Computational Problems in Abstract Algebra*. Pergamon Press. 1970. pp. 263-297.

[10] Chang CC and Keisler HJ. *Model Theory*. North-Holland. 1990. pp. 38-39.

[11] Birkhoff G. *Lattice Theory*. Third Edition. American Mathematical Society. 1967.

[12] Church A. *Introduction to Mathematical Logic. Volume I.* Princeton. 1956.

[13] Jauch J. *Foundations of Quantum Mechanics.* Addison-Wesley. 1968.

[14] Megill ND. *Metamath.* URL http://us.metamath.org/qlegif/mmql.html#unify. 2004.

[15] Horner JK. An automated deduction system for orthomodular lattice theory. *Proceedings of the 2005 International Conference on Artificial Intelligence*. CSREA Press. 2005. pp. 260-265.

[16] Horner JK. An automated equational logic deduction of join elimination in orthomodular lattice theory. *Proceedings of the 2007 International Conference on Artificial Intelligence*. CSREA Press. 2007. pp. 481-488.

[17] Messiah A. *Quantum Mechanics*. Dover. 1958.

[18] Horner JK. Using automated theorem-provers to aid the design of efficient compilers for quantum computing. *Los Alamos National Laboratory Quantum Institute Workshop.* December 9–10, 2002. URL http://www.lanl.gov/science/centers/quantum/qls_pdfs/horner.pdf.

[19] Birkhoff G and von Neumann J. The logic of quantum mechanics. *Annals of Mathematics* 37 (1936), 823-243.

[20] Nielsen MA and Chuang L . *Quantum Computation and Quantum Information*. Cambridge. 2000.

[21] Pavičić M and Megill N. Quantum and classical implicational algebras with primitive implication. *International Journal of Theoretical Physics* 37 (1998), 2091-2098. ftp://m3k.grad.hr/pavicic/quantum-logic/1998-int-j-theor-phys-2.ps.gz.

[22] Horner JK. An automated deduction of the relative strength of orthomodular and weakly orthomodular lattice theory. *Proceedings of the 2009 International Conference on Artificial Intelligence*. CSREA Press. 2009. pp. 525-530.

Quantum-Identity Equivalents of the Orthomodularity Law in Quantum Logic: Part 4

Jack K. Horner
P. O. Box 266
Los Alamos, New Mexico 87544 USA
jhorner@cybermesa.com

FCS 2014

Abstract

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of quantum-mechanical behaviors and operations. In much the same way that the structure of conventional propositional (Boolean) logic (BL) is the logic of the description of the behavior of classical physical systems and is isomorphic to a Boolean algebra (BA), so also the algebra, C(H), of closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space is a logic of the descriptions of the behavior of quantum mechanical systems and is a model of an ortholattice (OL). An OL can thus be thought of as a kind of "quantum logic" (QL). C(H) is also a model of an orthomodular lattice, which is an OL conjoined with the orthomodularity axiom (OMA). The rationalization of the OMA as a claim proper to physics has proven problematic, motivating the question of whether the OMA and its equivalents are required in an adequate characterization of QL. Here I provide an automated deduction of two quantum- identity-based equivalents of the OMA. The proofs may be novel.

Keywords: automated deduction, quantum computing, orthomodular lattice, Hilbert space

1.0 Introduction

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of the description of quantum-mechanical behaviors ([1], [17], [18], [20]). In much the same way that conventional propositional (Boolean) logic (BL,[12]) is the logical structure of description of the behavior of classical physical systems (e.g. "the measurements of the position and momentum of particle P are commutative", i.e., can be measured in either other, yielding the same results) and is isomorphic to a Boolean lattice ([10], [11], [19]), so also the algebra, *C(H)*, of the closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space *H* ([1], [4], [6], [9], [13]) is *a* logic of the descriptions of the behavior of quantum mechanical systems (e.g., "the measurements of the position and momentum of particle P are *not* commutative") and is a model ([10]) of an ortholattice (OL; [4]). An OL can thus be thought of as a kind of "quantum logic" (QL; [19]). *C(H)* is also a model of (i.e., isomorphic to a set of sentences which hold in) an orthomodular lattice (OML; [4], [7]), which is an OL conjoined with the orthomodularity axiom (OMA; see Figure 1). The rationalization of the OMA as a claim proper to physics has proven problematic ([13], Section 5-6), motivating the question of whether the OMA is required in an adequate characterization of QL. Thus formulated, the question suggests that the OMA and its equivalents are specific to an OML, and that as a consequence, banning the OMA from QL yields a "truer" quantum logic.

Lattice axioms

```
      x = c(c(x))                       (AxLat1)
      x v y = y v x                     (AxLat2)
      (x v y) v z = x v (y v z)          (AxLat3)
      (x ^ y) ^ z = x ^ (y ^ z)          (AxLat4)
      x v (x ^ y) = x                   (AxLat5)
      x ^ (x v y) = x                   (AxLat6)
```

Ortholattice axioms

```
      c(x) ^ x = 0                      (AxOL1)
      c(x) v x = 1                      (AxOL2)
      x ^ y = c(c(x) v c(y))            (AxOL3)
```

Orthomodularity axiom

```
      y v (c(y) ^ (x v y)) = x v y      (OMA)
```

Definitions of implications and partial order

```
   i1(x,y) = c(x) v (x ^ y).
   i2(x,y) = i1(c(y), c(x).
   i3(x,y) = (c(x) ^ y) v (c(x) ^ c(y)) v i1(x,y).
   i4(x,y) = i3(c(y), c(x)).
   i5(x,y) = (x ^ y) v (c(x) ^ y) v (c(x) ^ c(y)).
   le(x,y) = (x = (x ^ y)).
```

Definitions of indexed identities

```
    id1(x,y) = i1(x,y) ^ i0(y, x)
    id2(x,y) = i2(x,y) ^ i0(y, x)
    id3(x,y) = i3(x,y) ^ i0(y, x)
    id4(x,y) = i4(x,y) ^ i0(y, x)
    id5(x,y) = i5(x,y) ^ i0(y, x)
```

```
where
   x, y are variables ranging over lattice nodes
   ^ is lattice meet
   v is lattice join
   c(x) is the orthocomplement of x
   i1(x,y) means x →1 y (Sasaki implication)
   i2(x,y) means x →2 y (Dishkant implication)
   i3(x,y) means x →3 y (Kalmbach implication)
   i4(x,y) means x →4 y (non-tollens implication)
   i5(x,y) means x →5 y (relevance implication)
   le(x,y) means x ≤ y
   <-> means if and only if
   =  is equivalence ([12])
   1 is the maximum lattice element (= x v c(x))
   0 is the minimum lattice element (= c(1))
```

Figure 1. Lattice, ortholattice, orthomodularity axioms, and some definitions.

There are at least 21 nominal equivalents (in the sense that ortholattice theory, together with these "equivalents", imply the OMA, and vice versa) of the OMA in quantum logic ([5], Theorem 2.5); as nominal equivalents, they are thus of import to optimizing quantum circuit design. Among these is the Proposition shown in Figure 2:

$$x \equiv_i y \quad <-> \quad x = y$$

where

$x \equiv_i y$ means $(x \to_i y) \wedge (x \to_0 y)$
$x \to_0 y$ means $c(x) \vee y$
$i = 1,2,3,4,5$

Figure 2. Proposition 2.13

2.0 Method

The OML axiomatizations of Megill, Pavičić, and Horner ([5], [14], [15], [16], [21], [22]) were implemented in a *prover9* ([2]) script ([3]) configured to derive Proposition 2.13 of [5], for each of i = 3,4, from orthomodular lattice theory, then executed in that framework on a Dell Inspiron 545 with an Intel Core2 Quad CPU Q8200 (clocked @ 2.33 GHz) and 8.00 GB RAM, running under the *Windows Vista Home Premium /Cygwin* operating environment.

3.0 Results

Figure 3 shows the proofs, generated by [3] on the platform described in Section 2.0, that orthomodular lattice theory implies Proposition 2.13 of [5] (for each of i = 3,4).

```
============================== PROOF =================================

% Proof 1 at 9.98 (+ 0.05) seconds.
% Length of proof is 51.
% Level of proof is 13.

4 id3(x,y) = 1 <-> x = y # label("Proposition 2.10id3") # label(non_clause) #
label(goal).  [goal].
11 x = c(c(x)) # label("AxL1").  [assumption].
12 c(c(x)) = x.  [copy(11),flip(a)].
13 x v y = y v x # label("AxL2").  [assumption].
14 (x v y) v z = x v (y v z) # label("AxL3").  [assumption].
16 x v (x ^ y) = x # label("AxL5").  [assumption].
17 x ^ (x v y) = x # label("AxL6").  [assumption].
18 c(x) ^ x = 0 # label("AxOL1").  [assumption].
19 c(x) v x = 1 # label("AxOL2").  [assumption].
20 x v c(x) = 1.  [copy(19),rewrite([13(2)])].
21 x ^ y = c(c(x) v c(y)) # label("AxOL3").  [assumption].
22 x v (c(x) ^ (y v x)) = y v x # label("OMA").  [assumption].
23 x v c(x v c(y v x)) = y v x.  [copy(22),rewrite([21(3),12(2)])].
24 i0(x,y) = c(x) v y # label("Df: i0").  [assumption].
25 i1(x,y) = c(x) v (x ^ y) # label("Df: i1").  [assumption].
26 i1(x,y) = c(x) v c(c(x) v c(y)).  [copy(25),rewrite([21(3)])].
29 i3(x,y) = ((c(x) ^ y) v (c(x) ^ c(y))) v (c(x) v (x ^ y)) # label("Df: i3").
[assumption].
30 i3(x,y) = c(x v y) v (c(x v c(y)) v (c(x) v c(c(x) v c(y)))).
[copy(29),rewrite([21(3),12(3),21(7),12(6),12(6),13(7),21(9),14(14)])].
59 id3(x,y) = i3(x,y) ^ i0(y,x) # label("Df: id3").  [assumption].
60 id3(x,y) = c(c(c(y) v x) v c(c(x v y) v (c(x v c(y)) v (c(x) v c(c(x) v c(y)))))).
[copy(59),rewrite([30(2),24(15),21(17),13(19)])].
65 id3(c1,c2) = 1 | c2 = c1 # label("Proposition 2.10id3").  [deny(4)].
66 c(c(c1 v c(c2)) v c(c(c1 v c2) v (c(c1 v c(c2)) v (c(c1) v c(c(c1) v c(c2)))))) = 1 |
c2 = c1.  [copy(65),rewrite([60(3),13(4)])].
67 id3(c1,c2) != 1 | c2 != c1 # label("Proposition 2.10id3").  [deny(4)].
```

```
68 c(c(c1 v c(c2)) v c(c(c1 v c2) v (c(c1 v c(c2)) v (c(c1) v c(c(c1) v c(c2)))))) != 1 |
c2 != c1.  [copy(67),rewrite([60(3),13(4)])].
69 c(1) = 0.  [back_rewrite(18),rewrite([21(2),12(2),20(2)])].
70 c(c(x) v c(x v y)) = x.  [back_rewrite(17),rewrite([21(2)])].
71 x v c(c(x) v c(y)) = x.  [back_rewrite(16),rewrite([21(1)])].
73 x v (y v z) = y v (x v z).  [para(13(a,1),14(a,1,1)),rewrite([14(2)])].
74 c(c(c1 v c(c2)) v c(c(c1) v (c(c1 v c2) v (c(c1 v c(c2)) v c(c(c1) v c(c2)))))) != 1 |
c2 != c1.  [back_rewrite(68),rewrite([73(24),73(25)])].
75 c(c(c1 v c(c2)) v c(c(c1) v (c(c1 v c2) v (c(c1 v c(c2)) v c(c(c1) v c(c2)))))) = 1 |
c2 = c1.  [back_rewrite(66),rewrite([73(24),73(25)])].
84 x v c(x v c(x v y)) = y v x.  [para(13(a,1),23(a,1,2,1,2,1))].
90 c(0) = 1.  [para(69(a,1),12(a,1,1))].
92 c(x) v c(x v y) = c(x).  [para(70(a,1),12(a,1,1)),flip(a)].
96 c(0 v c(x)) = x.  [para(20(a,1),70(a,1,1,2,1)),rewrite([69(3),13(3)])].
98 1 v x = 1.  [para(69(a,1),70(a,1,1,1)),rewrite([96(6)])].
101 x v c(c(x) v y) = x.  [para(12(a,1),71(a,1,2,1,2))].
105 x v 0 = x.  [para(20(a,1),71(a,1,2,1)),rewrite([69(2)])].
106 x v c(y v c(x)) = x.  [para(23(a,1),71(a,1,2,1))].
107 x v x = x.  [para(69(a,1),71(a,1,2,1,2)),rewrite([13(3),96(4)])].
113 0 v x = x.  [para(105(a,1),13(a,1)),flip(a)].
147 c2 = c1 | c1 v c(c2) = 1.  [para(75(a,1),101(a,1,2)),rewrite([13(9),98(9)]),flip(b)].
156 c2 = c1 | c(c1) v (c(c1 v c2) v (c(c1 v c(c2)) v c(c(c1) v c(c2)))) = 1.
[para(75(a,1),106(a,1,2)),rewrite([13(25),98(25)]),flip(b)].
180 c(x) v (c(x v y) v z) = c(x) v z.  [para(92(a,1),14(a,1,1)),flip(a)].
185 c2 = c1 | c(c1) v c(c(c1) v c(c2)) = 1.
[back_rewrite(156),rewrite([180(23),180(18)])].
202 c(c(c1 v c(c2)) v c(c(c1) v c(c(c1) v c(c2)))) != 1 | c2 != c1.
[back_rewrite(74),rewrite([180(25),180(20)])].
1260 c2 = c1 | c(c1) v c(c2) = c(c1).
[para(185(b,1),26(a,2,2,1)),rewrite([26(10),84(16),13(8),69(12),13(12),113(12)])].
1331 c2 = c1 | c1 v c2 = c2.  [para(1260(b,1),106(a,1,2,1)),rewrite([12(7),13(6)])].
1359 c2 = c1 | c1 v c(c1 v c(c2)) = c1 v c2.
[para(1331(b,1),84(a,1,2,1,2,1)),rewrite([13(13)])].
8163 c2 = c1 | c1 v c2 = c1.
[para(147(b,1),1359(b,1,2,1)),rewrite([69(9),13(9),113(9)]),flip(c),merge(b)].
8252 c2 = c1.  [para(8163(b,1),1331(b,1)),flip(c),merge(b),merge(c)].
8253 $F.
[back_rewrite(202),rewrite([8252(2),20(4),69(2),8252(6),107(8),12(6),13(5),20(5),69(3),10
7(3),90(2),8252(4)]),xx(a),xx(b)].

============================== end of proof ==========================


============================== PROOF =================================

% Proof 1 at 0.09 (+ 0.05) seconds: "OMA".
% Length of proof is 46.
% Level of proof is 12.

4 id4(x,y) = 1 <-> x = y # label("Hypothesis for Proposition 2.10id4") #
label(non_clause).  [assumption].
5 y v (c(y) ^ (x v y)) = x v y # label("OMA") # label(non_clause) # label(goal).  [goal].
12 x = c(c(x)) # label("AxL1").  [assumption].
13 c(c(x)) = x.  [copy(12),flip(a)].
14 x v y = y v x # label("AxL2").  [assumption].
15 (x v y) v z = x v (y v z) # label("AxL3").  [assumption].
17 x v (x ^ y) = x # label("AxL5").  [assumption].
18 x ^ (x v y) = x # label("AxL6").  [assumption].
19 c(x) ^ x = 0 # label("AxOL1").  [assumption].
20 c(x) v x = 1 # label("AxOL2").  [assumption].
21 x v c(x) = 1.  [copy(20),rewrite([14(2)])].
22 x ^ y = c(c(x) v c(y)) # label("AxOL3").  [assumption].
23 i0(x,y) = c(x) v y # label("Df: i0").  [assumption].
30 i4(x,y) = ((c(c(y)) ^ c(x)) v (c(c(y)) ^ c(c(x)))) v (c(c(y)) v (c(y) ^ c(x))) #
label("Df: i4").  [assumption].
31 i4(x,y) = y v (c(y v x) v (c(c(y) v x) v c(c(y) v c(x)))).
[copy(30),rewrite([13(3),22(3),13(4),13(6),13(6),22(5),13(11),22(12),13(11),13(11),14(13)
,15(13)])].
60 id4(x,y) = i4(x,y) ^ i0(y,x) # label("Df: id4").  [assumption].
61 id4(x,y) = c(c(c(y) v x) v c(y v (c(y v x) v (c(c(y) v x) v c(c(y) v c(x)))))).
[copy(60),rewrite([31(2),23(14),22(16),14(18)])].
```

```
64 id4(x,y) != 1 | y = x # label("Hypothesis for Proposition 2.10id4").  [clausify(4)].
65 c(c(c(x) v y) v c(x v (c(x v y) v (c(c(x) v y) v c(c(x) v c(y)))))) != 1 | x = y.
[copy(64),rewrite([61(1)])].
68 c1 v (c(c1) ^ (c2 v c1)) != c2 v c1 # label("OMA") # answer("OMA").  [deny(5)].
69 c1 v c(c1 v c(c1 v c2)) != c1 v c2 # answer("OMA").
[copy(68),rewrite([14(6),22(7),13(4),14(12)])].
70 c(1) = 0.  [back_rewrite(19),rewrite([22(2),13(2),21(2)])].
71 c(c(x) v c(x v y)) = x.  [back_rewrite(18),rewrite([22(2)])].
72 x v c(c(x) v c(y)) = x.  [back_rewrite(17),rewrite([22(1)])].
74 x v (y v z) = y v (x v z).  [para(14(a,1),15(a,1,1)),rewrite([15(2)])].
81 x v (c(x) v y) = 1 v y.  [para(21(a,1),15(a,1,1)),flip(a)].
82 x v (y v c(x v y)) = 1.  [para(21(a,1),15(a,1)),flip(a)].
103 c(0 v c(0 v (c1 v (c2 v (c(c1 v (c1 v (c2 v c(c1 v c(c1 v c2))))) v c(c(c1 v c2) v
c(c1 v c(c1 v c(c1 v c2))))))))) != 1 # answer("OMA").
[ur(65,b,69,a(flip)),rewrite([74(14),82(14),70(2),74(17),15(16),74(32),82(32),70(20),74(3
7),74(38),15(37)])].
114 c(0 v c(x)) = x.  [para(21(a,1),71(a,1,1,2,1)),rewrite([70(3),14(3)])].
118 1 v x = 1.  [para(70(a,1),71(a,1,1,1)),rewrite([114(6)])].
122 0 v (c1 v (c2 v (c(c1 v (c1 v (c2 v c(c1 v c(c1 v c2))))) v c(c(c1 v c2) v c(c1 v
c(c1 v c(c1 v c2)))))) != 1 # answer("OMA").  [back_rewrite(103),rewrite([114(41)])].
134 x v (c(x) v y) = 1.  [back_rewrite(81),rewrite([118(5)])].
137 x v c(c(x) v y) = x.  [para(13(a,1),72(a,1,2,1,2))].
141 x v 0 = x.  [para(21(a,1),72(a,1,2,1)),rewrite([70(2)])].
144 x v x = x.  [para(70(a,1),72(a,1,2,1,2)),rewrite([14(3),114(4)])].
156 0 v (0 v (c1 v (c2 v (c(c1 v (c1 v (c2 v c(c1 v c(c1 v c2))))) v c(c(c1 v c2) v c(c1
v c(c1 v c(c1 v c2)))))))) != 1 # answer("OMA").
[ur(65,b,122,a(flip)),rewrite([70(2),118(80),70(43),70(44),70(84),114(123),14(120),15(120
),15(119),15(118),15(117),118(122),70(42),14(42),114(43)])].
157 0 v (0 v (0 v (0 v (c1 v (c2 v (c(c1 v (c1 v (c2 v c(c1 v c(c1 v c2))))) v c(c(c1 v
c2) v c(c1 v c(c1 v c(c1 v c2)))))))))) != 1 # answer("OMA").
[ur(65,b,156,a),rewrite([14(42),118(42),70(2),14(81),118(81),70(42),14(83),118(83),70(43)
,70(84),14(84),114(85),74(84),74(83),144(82),114(47)])].
159 0 v (0 v (0 v (0 v (0 v (0 v (c1 v (c2 v (c(c1 v (c1 v (c2 v c(c1 v c(c1 v c2))))) v
c(c(c1 v c2) v c(c1 v c(c1 v c(c1 v c2)))))))))))) != 1 # answer("OMA").
[ur(65,b,157,a),rewrite([14(46),118(46),70(2),14(89),118(89),70(46),14(91),118(91),70(47)
,70(92),14(92),114(93),74(92),74(91),144(90),114(51)])].
163 0 v x = x.  [para(141(a,1),14(a,1)),flip(a)].
164 c1 v (c2 v (c(c1 v (c1 v (c2 v c(c1 v c(c1 v c2))))) v c(c(c1 v c2) v c(c1 v c(c1 v
c(c1 v c2)))))) != 1 # answer("OMA").
[back_rewrite(159),rewrite([163(42),163(41),163(40),163(39),163(38),163(37)])].
175 x v (x v y) = x v y.  [para(144(a,1),15(a,1,1)),flip(a)].
180 c1 v (c2 v (c(c1 v (c2 v c(c1 v c(c1 v c2)))) v c(c(c1 v c2) v c(c1 v c(c1 v c(c1 v
c2)))))) != 1 # answer("OMA").  [back_rewrite(164),rewrite([175(15)])].
185 x v (y v (c(x v y) v z)) = 1.  [para(134(a,1),15(a,1)),flip(a)].
198 x v c(y v c(x)) = x.  [para(14(a,1),137(a,1,2,1))].
215 x v (y v c(z v c(x v y))) = x v y.  [para(198(a,1),15(a,1)),flip(a)].
221 $F # answer("OMA").  [back_rewrite(180),rewrite([215(13),185(25)]),xx(a)].

============================== end of proof ==========================
```

Figure 3. Summary of a *prover9* ([2]) proof of Proposition 2.13 from orthomodular lattice theory, for each of i = 3,4. The proofs assume the default inference rules of *prover9*. The general form of a line in this proof is "*line_number conclusion [derivation]*", where *line_number* is a unique identifier of a line in the proof, and *conclusion* is the result of applying the *prover9* inference rules (such as *para*modulation, *copy*ing, and *rewrit*ing), noted in square brackets (denoting the *derivation*), to the lines cited in those brackets. Note that some of "logical" proof lines in the above have been transformed to two text lines, with the *derivation* appearing on a text line following a text line containing the first part of that logical line. The detailed syntax and semantics of these notations can be found in [2]. All *prover9* proofs are by default proofs by contradiction.

The total time to produce the proofs in Figure 3 on the platform described in Section 2.0 was approximately 2 seconds.

4.0 Discussion

The results of Section 3.0 motivate several observations:

1. The proofs in Figure 3 use L1, L2, L3, L5, L6, OL1, OL2, and OL3.

2. The proofs in Section 3.0 may be novel.

3. Companion papers provide proofs for i = 1,2, and 5, and the converse propositions.

4. Proposition 2.13 can be regarded as a definition of quantum identity; thus, this paper together with the papers mentioned in (3), constitute a proof that the definition of quantum intersection is equivalent to the OMA in orthomodular quantum logic. Companion papers derive equivalences for the OMA with definitions of quantum-intersection and quantum-union. Collectively, these papers provide a theory of equivalence of the OMA with the quantum connectives. In light of these equivalences, QL without the OMA would hardly qualify as a logic.

5.0 Acknowledgements

This work benefited from discussions with Tom Oberdan, Frank Pecchioni, Tony Pawlicki, and the late John K. Prentice, whose passion for foundations of physics inspired those of us privileged to have known him. For any infelicities that remain, I am solely responsible.

6.0 References

[1] von Neumann J. *Mathematical Foundations of Quantum Mechanics*. 1936. Translated by R. T. Beyer. Princeton. 1983.

[2] McCune WW. *prover9 and mace4.* URL http://www.cs.unm.edu/~mccune/prover9/. 2009.

[3] Horner JK. MHT *prover9* scripts. 2011. Available from the author on request.

[4] Dalla Chiara ML and Giuntini R. *Quantum Logics*. URL http://xxx.lanl.gov/abs/quant-ph/0101028. 2004.

[5] Megill ND and Pavičić M. Orthomodular lattices and quantum algebra. *International Journal of Theoretical Physics* 40 (2001), pp. 1387-1410.

[6] Akhiezer NI and Glazman IM. *Theory of Linear Operators in Hilbert Space. Volume I.* Translated by M. Nestell. Frederick Ungar. 1961.

[7] Holland, Jr. SS Orthomodularity in infinite dimensions: a theorem of M. Solèr. *Bulletin of the American Mathematical Society* 32 (1995), pp. 205-234.

[8] Marsden EL and Herman LM. *A condition for distribution in orthomodular lattices.* Kansas State University Technical Report #40. 1974.

[9] Knuth DE and Bendix PB. Simple word problems in universal algebras. In J. Leech, ed. *Computational Problems in Abstract Algebra.* Pergamon Press. 1970. pp. 263-297.

[10] Chang CC and Keisler HJ. *Model Theory*. North-Holland. 1990. pp. 38-39.

[11] Birkhoff G. *Lattice Theory.* Third Edition. American Mathematical Society. 1967.

[12] Church A. *Introduction to Mathematical Logic. Volume I.* Princeton. 1956.

[13] Jauch J. *Foundations of Quantum Mechanics.* Addison-Wesley. 1968.

[14] Megill ND. *Metamath.* URL http://us.metamath.org/qlegif/mmql.html#unify. 2004.

[15] Horner JK. An automated deduction system for orthomodular lattice theory. *Proceedings of the 2005 International Conference on Artificial Intelligence.* CSREA Press. 2005. pp. 260-265.

[16] Horner JK. An automated equational logic deduction of join elimination in orthomodular lattice theory. *Proceedings of the 2007 International Conference on Artificial Intelligence.* CSREA Press. 2007. pp. 481-488.

[17] Messiah A. *Quantum Mechanics*. Dover. 1958.

[18] Horner JK. Using automated theorem-provers to aid the design of efficient compilers for quantum computing. *Los Alamos National Laboratory Quantum Institute Workshop.* December 9–10, 2002. URL http://www.lanl.gov/science/centers/quantum/qls_pdfs/horner.pdf.

[19] Birkhoff G and von Neumann J. The logic of quantum mechanics. *Annals of Mathematics* 37 (1936), 823-243.

[20] Nielsen MA and Chuang L . *Quantum Computation and Quantum Information*. Cambridge. 2000.

[21] Pavičić M and Megill N. Quantum and classical implicational algebras with primitive implication. *International Journal of Theoretical Physics* 37 (1998), 2091-2098. ftp://m3k.grad.hr/pavicic/quantum-logic/1998-int-j-theor-phys-2.ps.gz.

[22] Horner JK. An automated deduction of the relative strength of orthomodular and weakly orthomodular lattice theory. *Proceedings of the 2009 International Conference on Artificial Intelligence*. CSREA Press. 2009. pp. 525-530.

Quantum-Identity Equivalents of the Orthomodularity Law in Quantum Logic: Part 5

Jack K. Horner
P. O. Box 266
Los Alamos, New Mexico 87544 USA

FCS 2014

Abstract

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of quantum-mechanical behaviors and operations. In much the same way that the structure of conventional propositional (Boolean) logic (BL) is the logic of the description of the behavior of classical physical systems and is isomorphic to a Boolean algebra (BA), so also the algebra, C(H), of closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space is a logic of the descriptions of the behavior of quantum mechanical systems and is a model of an ortholattice (OL). An OL can thus be thought of as a kind of "quantum logic" (QL). C(H) is also a model of an orthomodular lattice, which is an OL conjoined with the orthomodularity axiom (OMA). The rationalization of the OMA as a claim proper to physics has proven problematic, motivating the question of whether the OMA and its equivalents are required in an adequate characterization of QL. Here I provide an automated deduction of a quantum-identity-based equivalents of the OMA. The proofs may be novel.

Keywords: automated deduction, quantum computing, orthomodular lattice, Hilbert space

1.0 Introduction

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of the description of quantum-mechanical behaviors ([1], [17], [18], [20]). In much the same way that conventional propositional (Boolean) logic (BL,[12]) is the logical structure of description of the behavior of classical physical systems (e.g. "the measurements of the position and momentum of particle P are commutative", i.e., can be measured in either other, yielding the same results) and is isomorphic to a Boolean lattice ([10], [11], [19]), so also the algebra, *C(H)*, of the closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space *H* ([1], [4], [6], [9], [13]) is *a* logic of the descriptions of the behavior of quantum mechanical systems (e.g., "the measurements of the position and momentum of particle P are *not* commutative") and is a model ([10]) of an ortholattice (OL; [4]). An OL can thus be thought of as a kind of "quantum logic" (QL; [19]). *C(H)* is also a model of (i.e., isomorphic to a set of sentences which hold in) an orthomodular lattice (OML; [4], [7]), which is an OL conjoined with the orthomodularity axiom (OMA; see Figure 1). The rationalization of the OMA as a claim proper to physics has proven problematic ([13], Section 5-6), motivating the question of whether the OMA is required in an adequate characterization of QL. Thus formulated, the question suggests that the OMA and its equivalents are specific to an OML, and that as a consequence, banning the OMA from QL yields a "truer" quantum logic.

Lattice axioms

```
      x = c(c(x))                       (AxLat1)
      x v y = y v x                     (AxLat2)
      (x v y) v z = x v (y v z)         (AxLat3)
      (x ^ y) ^ z = x ^ (y ^ z)         (AxLat4)
      x v (x ^ y) = x                   (AxLat5)
      x ^ (x v y) = x                   (AxLat6)
```

Ortholattice axioms

```
      c(x) ^ x = 0                      (AxOL1)
      c(x) v x = 1                      (AxOL2)
      x ^ y = c(c(x) v c(y))            (AxOL3)
```

Orthomodularity axiom

```
      y v (c(y) ^ (x v y)) = x v y      (OMA)
```

Definitions of implications and partial order

```
   i1(x,y) = c(x) v (x ^ y).
   i2(x,y) = i1(c(y), c(x).
   i3(x,y) = (c(x) ^ y) v (c(x) ^ c(y)) v i1(x,y).
   i4(x,y) = i3(c(y), c(x)).
   i5(x,y) = (x ^ y) v (c(x) ^ y) v (c(x) ^ c(y)).
   le(x,y) = (x = (x ^ y)).
```

Definitions of indexed identities

```
    id1(x,y) = i1(x,y) ^ i0(y, x)
    id2(x,y) = i2(x,y) ^ i0(y, x)
    id3(x,y) = i3(x,y) ^ i0(y, x)
    id4(x,y) = i4(x,y) ^ i0(y, x)
    id5(x,y) = i5(x,y) ^ i0(y, x)
```

where
- x, y are variables ranging over lattice nodes
- ^ is lattice meet
- v is lattice join
- c(x) is the orthocomplement of x
- i1(x,y) means $x \rightarrow_1 y$ (Sasaki implication)
- i2(x,y) means $x \rightarrow_2 y$ (Dishkant implication)
- i3(x,y) means $x \rightarrow_3 y$ (Kalmbach implication)
- i4(x,y) means $x \rightarrow_4 y$ (non-tollens implication)
- i5(x,y) means $x \rightarrow_5 y$ (relevance implication)
- le(x,y) means $x \leq y$
- <-> means if and only if
- = is equivalence ([12])
- 1 is the maximum lattice element (= x v c(x))
- 0 is the minimum lattice element (= c(1))

Figure 1. Lattice, ortholattice, orthomodularity axioms, and some definitions.

There are at least 21 nominal equivalents (in the sense that ortholattice theory, together with these "equivalents", imply the OMA, and vice versa) of the OMA in quantum logic ([5], Theorem 2.5); as nominal equivalents, they are thus of import to optimizing quantum circuit design. Among these is the Proposition shown in Figure 2:

$$x \equiv_i y \quad <-> \quad x = y$$

where

$x \equiv_i y$ means $(x \rightarrow_i y) \wedge (x \rightarrow_0 y)$
$x \rightarrow_0 y$ means $c(x) \vee y$
$i = 1,2,3,4,5$

Figure 2. Proposition 2.13 of [5]

2.0 Method

The OML axiomatizations of Megill, Pavičić, and Horner ([5], [14], [15], [16], [21], [22]) were implemented in a *prover9* ([2]) script ([3]) configured to derive Proposition 2.13 of [5], for i = 5, from orthomodular lattice theory, then executed in that framework on a Dell Inspiron 545 with an Intel Core2 Quad CPU Q8200 (clocked @ 2.33 GHz) and 8.00 GB RAM, running under the *Windows Vista Home Premium /Cygwin* operating environment.

3.0 Results

Figure 3 shows the proofs, generated by [3] on the platform described in Section 2.0, that orthomodular lattice theory implies Proposition 2.13 of [5] (for i = 5).

```
========== PROOF =================================

% Proof 1 at 4.48 (+ 0.06) seconds.
% Length of proof is 53.
% Level of proof is 11.
.

4 id5(x,y) = 1 <-> x = y # label("Proposition 2.10id5") #
label(non_clause) # label(goal).  [goal].
11 x = c(c(x)) # label("AxL1").  [assumption].
12 c(c(x)) = x.  [copy(11),flip(a)].
13 x v y = y v x # label("AxL2").  [assumption].
14 (x v y) v z = x v (y v z) # label("AxL3").  [assumption].
16 x v (x ^ y) = x # label("AxL5").  [assumption].
17 x ^ (x v y) = x # label("AxL6").  [assumption].
18 c(x) ^ x = 0 # label("AxOL1").  [assumption].
19 c(x) v x = 1 # label("AxOL2").  [assumption].
20 x v c(x) = 1.  [copy(19),rewrite([13(2)])].
21 x ^ y = c(c(x) v c(y)) # label("AxOL3").  [assumption].
22 x v (c(x) ^ (y v x)) = y v x # label("OMA").  [assumption].
23 x v c(x v c(y v x)) = y v x.  [copy(22),rewrite([21(3),12(2)])].
24 i0(x,y) = c(x) v y # label("Df: i0").  [assumption].
25 i1(x,y) = c(x) v (x ^ y) # label("Df: i1").  [assumption].
26 i1(x,y) = c(x) v c(c(x) v c(y)).  [copy(25),rewrite([21(3)])].
27 i2(x,y) = c(c(y)) v (c(y) ^ c(x)) # label("Df: i2").
[assumption].
```

```
28 i2(x,y) = y v c(y v x).
[copy(27),rewrite([12(3),21(4),12(3),12(3)])].
33 i5(x,y) = ((x ^ y) v (c(x) ^ y)) v (c(x) ^ c(y)) # label("Df:
i5").  [assumption].
34 i5(x,y) = c(x v y) v (c(x v c(y)) v c(c(x) v c(y))).
[copy(33),rewrite([21(2),21(7),12(7),13(9),21(12),12(11),12(11),13(1
2)])].
63 id5(x,y) = i5(x,y) ^ i0(y,x) # label("Df: id5").  [assumption].
64 id5(x,y) = c(c(c(y) v x) v c(c(x v y) v (c(x v c(y)) v c(c(x) v
c(y))))).  [copy(63),rewrite([34(2),24(13),21(15),13(17)])].
65 id5(c1,c2) = 1 | c2 = c1 # label("Proposition 2.10id5").
[deny(4)].
66 c(c(c1 v c(c2)) v c(c(c1 v c2) v (c(c1 v c(c2)) v c(c(c1) v
c(c2))))) = 1 | c2 = c1.  [copy(65),rewrite([64(3),13(4)])].
67 id5(c1,c2) != 1 | c2 != c1 # label("Proposition 2.10id5").
[deny(4)].
68 c(c(c1 v c(c2)) v c(c(c1 v c2) v (c(c1 v c(c2)) v c(c(c1) v
c(c2))))) != 1 | c2 != c1.  [copy(67),rewrite([64(3),13(4)])].
69 c(1) = 0.  [back_rewrite(18),rewrite([21(2),12(2),20(2)])].
70 c(c(x) v c(x v y)) = x.  [back_rewrite(17),rewrite([21(2)])].
71 x v c(c(x) v c(y)) = x.  [back_rewrite(16),rewrite([21(1)])].
73 x v (y v z) = y v (x v z).
[para(13(a,1),14(a,1,1)),rewrite([14(2)])].
82 x v c(x v c(x v y)) = y v x.  [para(13(a,1),23(a,1,2,1,2,1))].
100 c(0) = 1.  [para(69(a,1),12(a,1,1))].
102 c(x) v c(x v y) = c(x).  [para(70(a,1),12(a,1,1)),flip(a)].
106 c(0 v c(x)) = x.
[para(20(a,1),70(a,1,1,2,1)),rewrite([69(3),13(3)])].
110 1 v x = 1.  [para(69(a,1),70(a,1,1,1)),rewrite([106(6)])].
127 x v 0 = x.  [para(20(a,1),71(a,1,2,1)),rewrite([69(2)])].
128 x v c(y v c(x)) = x.  [para(23(a,1),71(a,1,2,1))].
130 c2 = c1 | c1 v c(c2) = 1.
[para(66(a,1),71(a,1,2)),rewrite([13(9),110(9)]),flip(b)].
131 x v x = x.
[para(69(a,1),71(a,1,2,1,2)),rewrite([13(3),106(4)])].
138 0 v x = x.  [para(127(a,1),13(a,1)),flip(a)].
158 c(x) v c(y v x) = c(x).  [para(12(a,1),128(a,1,2,1,2))].
163 c2 = c1 | c(c1 v c2) v (c(c1 v c(c2)) v c(c(c1) v c(c2))) = 1.
[para(66(a,1),128(a,1,2)),rewrite([13(22),110(22)]),flip(b)].
174 c2 = c1 | c(c(c1 v c(c2)) v c(c(c1 v c2) v c(c(c1) v c(c2)))) =
1.
[para(130(b,1),66(a,1,1,2,1,2,1,1)),rewrite([69(14),138(20)]),merge(
c)].
190 c(x) v (c(x v y) v z) = c(x) v z.
[para(102(a,1),14(a,1,1)),flip(a)].
281 c(x) v (y v c(z v x)) = y v c(x).
[para(158(a,1),73(a,1,2)),flip(a)].
1164 c2 = c1 | c(c1) v c(c(c1) v c(c2)) = 1.
[para(163(b,1),190(a,1,2)),rewrite([13(7),110(7),190(19)]),flip(b)].
1531 c(x v c(y)) v c(x v c(z v y)) = c(x v c(z v y)).
[para(281(a,1),158(a,1,2,1)),rewrite([13(8)])].
3557 c2 = c1 | c(c1) v c(c2) = c(c1).
[para(1164(b,1),26(a,2,2,1)),rewrite([26(10),82(16),13(8),69(12),13(
12),138(12)])].
```

```
3673 c2 = c1 | c1 v c2 = c2.
[para(3557(b,1),128(a,1,2,1)),rewrite([12(7),13(6)])].
3695 c2 = c1 | c1 v c(c1 v c2) = 1.
[para(3557(b,1),174(b,1,1,2,1,2,1)),rewrite([12(18),13(17),1531(19),
12(14)]),merge(b)].
4060 c2 = c1 | c1 v c2 = c1.
[para(3695(b,1),28(a,2,2,1)),rewrite([28(9),82(12),13(6),69(9),13(9)
,138(9)])].
4115 c2 = c1.
[para(4060(b,1),3673(b,1)),flip(c),merge(b),merge(c)].
4116 $F.
[back_rewrite(68),rewrite([4115(2),20(4),69(2),4115(3),131(4),4115(5
),20(7),69(5),4115(7),131(9),12(7),138(6),13(5),20(5),69(3),131(3),1
00(2),4115(4)]),xx(a),xx(b)].

============== end of proof ==========================
```

Figure 3. Summary of a *prover9* ([2]) proof of Proposition 2.13 from orthomodular lattice theory, for i = 5. The proofs assume the default inference rules of *prover9*. The general form of a line in this proof is "*line_number conclusion [derivation]*", where *line_number* is a unique identifier of a line in the proof, and *conclusion* is the result of applying the *prover9* inference rules (such as *para*modulation, *copy*ing, and *rewrit*ing), noted in square brackets (denoting the *derivation*), to the lines cited in those brackets. Note that some of "logical" proof lines in the above have been transformed to two text lines, with the *derivation* appearing on a text line following a text line containing the first part of that logical line. The detailed syntax and semantics of these notations can be found in [2]. All *prover9* proofs are by default proofs by contradiction.

The total time to produce the proofs in Figure 3 on the platform described in Section 2.0 was approximately 2 seconds.

4.0 Discussion

The results of Section 3.0 motivate several observations:

1. The proof in Figure 3 uses L1, L2, L3, L5, L6, OL1, OL2, and OL3.

2. The proofs in Section 3.0 may be novel.

3. Companion papers provide proofs for i = 1,2,3,4, and the converse propositions.

4. Proposition 2.13 can be regarded as a definition of quantum identity; thus, this paper together with the papers mentioned in (3), constitute a proof that the definition of quantum intersection is equivalent to the OMA in orthomodular quantum logic. Companion papers derive equivalences for the OMA with definitions of quantum-intersection and quantum-union. Collectively, these papers provide a theory of equivelance of the OMA with the quantum connectives. In light of these equivalences, QL without the OMA would hardly qualify as a logic.

5.0 Acknowledgements

This work benefited from discussions with Tom Oberdan, Frank Pecchioni, Tony Pawlicki, and the late John K. Prentice, whose passion for foundations of physics inspired those of us privileged to have known him. For any infelicities that remain, I am solely responsible.

6.0 References

[1] von Neumann J. *Mathematical Foundations of Quantum Mechanics*. 1936. Translated by R. T. Beyer. Princeton. 1983.

[2] McCune WW. *prover9 and mace4.* URL http://www.cs.unm.edu/~mccune/prover9/. 2009.

[3] Horner JK. MHT *prover9* scripts. 2011. Available from the author on request.

[4] Dalla Chiara ML and Giuntini R. *Quantum Logics*. URL http://xxx.lanl.gov/abs/quant-ph/0101028. 2004.

[5] Megill ND and Pavičić M. Orthomodular lattices and quantum algebra. *International Journal of Theoretical Physics* 40 (2001), pp. 1387-1410.

[6] Akhiezer NI and Glazman IM. *Theory of Linear Operators in Hilbert Space. Volume I*. Translated by M. Nestell. Frederick Ungar. 1961.

[7] Holland, Jr. SS Orthomodularity in infinite dimensions: a theorem of M. Solèr. *Bulletin of the American Mathematical Society* 32 (1995), pp. 205-234.

[8] Marsden EL and Herman LM. *A condition for distribution in orthomodular lattices.* Kansas State University Technical Report #40. 1974.

[9] Knuth DE and Bendix PB. Simple word problems in universal algebras. In J. Leech, ed. *Computational Problems in Abstract Algebra*. Pergamon Press. 1970. pp. 263-297.

[10] Chang CC and Keisler HJ. *Model Theory*. North-Holland. 1990. pp. 38-39.

[11] Birkhoff G. *Lattice Theory*. Third Edition. American Mathematical Society. 1967.

[12] Church A. *Introduction to Mathematical Logic. Volume I.* Princeton. 1956.

[13] Jauch J. *Foundations of Quantum Mechanics.* Addison-Wesley. 1968.

[14] Megill ND. *Metamath.* URL http://us.metamath.org/qlegif/mmql.html#unify. 2004.

[15] Horner JK. An automated deduction system for orthomodular lattice theory. *Proceedings of the 2005 International Conference on Artificial Intelligence.* CSREA Press. 2005. pp. 260-265.

[16] Horner JK. An automated equational logic deduction of join elimination in orthomodular lattice theory. *Proceedings of the 2007 International Conference on Artificial Intelligence*. CSREA Press. 2007. pp. 481-488.

[17] Messiah A. *Quantum Mechanics*. Dover. 1958.

[18] Horner JK. Using automated theorem-provers to aid the design of efficient compilers for quantum computing. *Los Alamos National Laboratory Quantum Institute Workshop.* December 9–10, 2002. URL http://www.lanl.gov/science/centers/quantum/qls_pdfs/horner.pdf.

[19] Birkhoff G and von Neumann J. The logic of quantum mechanics. *Annals of Mathematics* 37 (1936), 823-243.

[20] Nielsen MA and Chuang L . *Quantum Computation and Quantum Information*. Cambridge. 2000.

[21] Pavičić M and Megill N. Quantum and classical implicational algebras with

primitive implication. *International Journal of Theoretical Physics* 37 (1998), 2091-2098.
ftp://m3k.grad.hr/pavicic/quantum-logic/1998-int-j-theor-phys-2.ps.gz.

[22] Horner JK. An automated deduction of the relative strength of orthomodular and weakly orthomodular lattice theory. *Proceedings of the 2009 International Conference on Artificial Intelligence*. CSREA Press. 2009. pp. 525-530.

Quantum-Union Equivalents of the Orthomodularity Law in Quantum Logic: Part 1

Jack K. Horner
P. O. Box 266
Los Alamos, New Mexico 87544 USA
jhorner@cybermesa.com

FCS 2014

Abstract

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of quantum-mechanical behaviors and operations. In much the same way that the structure of conventional propositional (Boolean) logic (BL) is the logic of the description of the behavior of classical physical systems and is isomorphic to a Boolean algebra (BA), so also the algebra, C(H), of closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space is a logic of the descriptions of the behavior of quantum mechanical systems and is a model of an ortholattice (OL). An OL can thus be thought of as a kind of "quantum logic" (QL). C(H) is also a model of an orthomodular lattice, which is an OL conjoined with the orthomodularity axiom (OMA). The rationalization of the OMA as a claim proper to physics has proven problematic, motivating the question of whether the OMA and its equivalents are required in an adequate characterization of QL. Here I provide an automated deduction of the OMA from three quantum- union-based equivalents. The proofs may be novel.

Keywords: automated deduction, quantum computing, orthomodular lattice, Hilbert space

1.0 Introduction

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of the description of quantum-mechanical behaviors ([1], [17], [18], [20]). In much the same way that conventional propositional (Boolean) logic (BL,[12]) is the logical structure of description of the behavior of classical physical systems (e.g. "the measurements of the position and momentum of particle P are commutative", i.e., can be measured in either other, yielding the same results) and is isomorphic to a Boolean lattice ([10], [11], [19]), so also the algebra, *C(H)*, of the closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space *H* ([1], [4], [6], [9], [13]) is *a* logic of the descriptions of the behavior of quantum mechanical systems (e.g., "the measurements of the position and momentum of particle P are *not* commutative") and is a model ([10]) of an ortholattice (OL; [4]). An OL can thus be thought of as a kind of "quantum logic" (QL; [19]). *C(H)* is also a model of (i.e., isomorphic to a set of sentences which hold in) an orthomodular lattice (OML; [4], [7]), which is an OL conjoined with the orthomodularity axiom (OMA; see Figure 1). The rationalization of the OMA as a claim proper to physics has proven problematic ([13], Section 5-6), motivating the question of whether the OMA is required in an adequate characterization of QL. Thus formulated, the question suggests that the OMA and its equivalents are specific to an OML, and that as a consequence, banning the OMA from QL yields a "truer" quantum logic.

Lattice axioms

```
      x = c(c(x))                       (AxLat1)
      x v y = y v x                     (AxLat2)
      (x v y) v z = x v (y v z)         (AxLat3)
      (x ^ y) ^ z = x ^ (y ^ z)         (AxLat4)
      x v (x ^ y) = x                   (AxLat5)
      x ^ (x v y) = x                   (AxLat6)
```

Ortholattice axioms

```
      c(x) ^ x = 0                      (AxOL1)
      c(x) v x = 1                      (AxOL2)
      x ^ y = c(c(x) v c(y))            (AxOL3)
```

Orthomodularity axiom

```
      y v (c(y) ^ (x v y)) = x v y      (OMA)
```

Definitions of implications and partial order

```
   i1(x,y) = c(x) v (x ^ y).
   i2(x,y) = i1(c(y), c(x).
   i3(x,y) = (c(x) ^ y) v (c(x) ^ c(y)) v i1(x,y).
   i4(x,y) = i3(c(y), c(x)).
   i5(x,y) = (x ^ y) v (c(x) ^ y) v (c(x) ^ c(y)).
   le(x,y) = (x = (x ^ y)).
```

Definitions of indexed unions

```
   u1(x,y) =  i1(c(x),y)
   u2(x,y) =  i2(c(x),y)
   u3(x,y) =  i3(c(x),y)
   u4(x,y) =  i4(c(x),y)
   u5(x,y) =  i5(c(x),y)
```

where
   x, y are variables ranging over lattice nodes
   ^ is lattice meet
   v is lattice join
   c(x) is the orthocomplement of x
   i1(x,y) means $x \rightarrow_1 y$ (Sasaki implication)
   i2(x,y) means $x \rightarrow_2 y$ (Dishkant implication)
   i3(x,y) means $x \rightarrow_3 y$ (Kalmbach implication)
   i4(x,y) means $x \rightarrow_4 y$ (non-tollens implication)
   i5(x,y) means $x \rightarrow_5 y$ (relevance implication)
   le(x,y) means $x \leq y$
   <-> means if and only if
   = is equivalence ([12])
   1 is the maximum lattice element (= x v c(x))
   0 is the minimum lattice element (= c(1))

Figure 1. Lattice, ortholattice, orthomodularity axioms, and some definitions.

There are at least 21 nominal equivalents (in the sense that ortholattice theory, together with these "equivalents", imply the OMA, and vice versa) of the OMA in quantum logic ([5], Theorem 2.5); as nominal equivalents, they are thus of import to optimizing quantum circuit or compiler design. Among these is the Proposition shown in Figure 2:

$$x \cup_i y \quad <-> \quad c(x) \perp c(y)$$

where

$x \cup_i y$ means $c(x) \rightarrow_i y$
$x \perp y$ means le(x,c(y))
i = 1,2,3,4,5

Figure 2. Proposition 2.11 of [5]

2.0 Method

The OML axiomatizations of Megill, Pavičić, and Horner ([5], [14], [15], [16], [21], [22]) were implemented in a *prover9* ([2]) script ([3]) configured to derive the OMA from Proposition 2.11 of [5], for each of i = 1,2,3 together with ortholattice theory (orthomodular lattice theory, without the OMA), then executed in that framework on a Dell Inspiron 545 with an Intel Core2 Quad CPU Q8200 (clocked @ 2.33 GHz) and 8.00 GB RAM, running under the *Windows Vista Home Premium /Cygwin* operating environment.

3.0 Results

Figure 3 shows the proofs, generated by [3] on the platform described in Section 2.0, that Proposition 2.11 (for each of i = 1,2,3), together with ortholattice theory, imply the OMA.

```
============================== PROOF =================================

% Proof 1 at 26.43 (+ 0.86) seconds: "OMA".
% Length of proof is 40.
% Level of proof is 12.

1 le(x,y) <-> x = x ^ y # label("Df: less than") # label(non_clause).  [assumption].
2 perp(x,y) <-> le(x,c(y)) # label("Df: perpendicular") # label(non_clause).
[assumption].
4 u1(x,y) = 1 <-> perp(c(x),c(y)) # label("Hypothesis for Proposition 2.11u1") #
label(non_clause).  [assumption].
5 y v (c(y) ^ (x v y)) = x v y # label("OMA") # label(non_clause) # label(goal).  [goal].
8 x = c(c(x)) # label("AxL1").  [assumption].
9 c(c(x)) = x.  [copy(8),flip(a)].
10 x v y = y v x # label("AxL2").  [assumption].
11 (x v y) v z = x v (y v z) # label("AxL3").  [assumption].
13 x v (x ^ y) = x # label("AxL5").  [assumption].
16 c(x) v x = 1 # label("AxOL2").  [assumption].
17 x v c(x) = 1.  [copy(16),rewrite([10(2)])].
18 x ^ y = c(c(x) v c(y)) # label("AxOL3").  [assumption].
20 i1(x,y) = c(x) v (x ^ y) # label("Df: i1").  [assumption].
21 i1(x,y) = c(x) v c(c(x) v c(y)).  [copy(20),rewrite([18(3)])].
30 u1(x,y) = i1(c(x),y) # label("Df: u1").  [assumption].
31 u1(x,y) = x v c(x v c(y)).  [copy(30),rewrite([21(3),9(3),9(3)])].
40 -le(x,y) | x ^ y = x # label("Df: less than").  [clausify(1)].
41 -le(x,y) | c(c(x) v c(y)) = x.  [copy(40),rewrite([18(2)])].
44 -perp(x,y) | le(x,c(y)) # label("Df: perpendicular").  [clausify(2)].
46 u1(x,y) != 1 | perp(c(x),c(y)) # label("Hypothesis for Proposition 2.11u1").
[clausify(4)].
47 x v c(x v c(y)) != 1 | perp(c(x),c(y)).  [copy(46),rewrite([31(1)])].
50 c1 v (c(c1) ^ (c2 v c1)) != c2 v c1 # label("OMA") # answer("OMA").  [deny(5)].
51 c1 v c(c1 v c(c1 v c2)) != c1 v c2 # answer("OMA").
[copy(50),rewrite([10(6),18(7),9(4),10(12)])].
54 x v c(c(x) v c(y)) = x.  [back_rewrite(13),rewrite([18(1)])].
56 x v (y v z) = y v (x v z).  [para(10(a,1),11(a,1,1)),rewrite([11(2)])].
60 x v (y v c(x v y)) = 1.  [para(17(a,1),11(a,1)),flip(a)].
66 x v c(x v y) != 1 | perp(c(x),y).  [para(9(a,1),47(a,1,2,1,2)),rewrite([9(8)])].
88 x v c(c(x) v y) = x.  [para(9(a,1),54(a,1,2,1,2))].
113 x v (y v c(y v x)) = 1.  [para(10(a,1),60(a,1,2,2,1))].
144 x v c(y v x) != 1 | perp(c(x),y).  [para(10(a,1),66(a,1,2,1))].
```

```
191 x v c(y v c(x)) = x.  [para(10(a,1),88(a,1,2,1))].
193 x v (y v c(c(x v y) v z)) = x v y.  [para(88(a,1),11(a,1)),flip(a)].
280 x v (y v c(z v c(x v y))) = x v y.  [para(191(a,1),11(a,1)),flip(a)].
1222 x v (c(y v x) v c(c(y v x) v z)) != 1 | perp(c(x v c(c(y v x) v z)),y).
[para(193(a,1),144(a,1,2,1)),rewrite([10(8),56(8)])].
22614 perp(c(x v c(x v c(y v x))),y).  [hyper(1222,a,113,a),rewrite([10(3)])].
22615 le(c(x v c(x v c(y v x))),c(y)).  [hyper(44,a,22614,a)].
22635 c(x v c(x v c(y v x))) = c(y v x).
[hyper(41,a,22615,a),rewrite([9(7),9(7),10(6),280(6)]),flip(a)].
22644 x v c(x v c(y v x)) = y v x.  [para(22635(a,1),9(a,1,1)),rewrite([9(3)]),flip(a)].
22655 x v c(x v c(x v y)) = y v x.  [para(10(a,1),22644(a,1,2,1,2,1))].
22695 $F # answer("OMA").  [back_rewrite(51),rewrite([22655(9),10(3)]),xx(a)].

============================== end of proof ==========================


============================== PROOF =================================

% Proof 1 at 24.51 (+ 0.89) seconds: "OMA".
% Length of proof is 51.
% Level of proof is 13.

1 le(x,y) <-> x = x ^ y # label("Df: less than") # label(non_clause).  [assumption].
2 perp(x,y) <-> le(x,c(y)) # label("Df: perpendicular") # label(non_clause).
[assumption].
4 u2(x,y) = 1 <-> perp(c(x),c(y)) # label("Hypothesis for Proposition 2.11u2") #
label(non_clause).  [assumption].
5 y v (c(y) ^ (x v y)) = x v y # label("OMA") # label(non_clause) # label(goal).  [goal].
8 x = c(c(x)) # label("AxL1").  [assumption].
9 c(c(x)) = x.  [copy(8),flip(a)].
10 x v y = y v x # label("AxL2").  [assumption].
11 (x v y) v z = x v (y v z) # label("AxL3").  [assumption].
13 x v (x ^ y) = x # label("AxL5").  [assumption].
14 x ^ (x v y) = x # label("AxL6").  [assumption].
15 c(x) ^ x = 0 # label("AxOL1").  [assumption].
16 c(x) v x = 1 # label("AxOL2").  [assumption].
17 x v c(x) = 1.  [copy(16),rewrite([10(2)])].
18 x ^ y = c(c(x) v c(y)) # label("AxOL3").  [assumption].
22 i2(x,y) = c(c(y)) v (c(y) ^ c(x)) # label("Df: i2").  [assumption].
23 i2(x,y) = y v c(y v x).  [copy(22),rewrite([9(3),18(4),9(3),9(3)])].
32 u2(x,y) = i2(c(x),y) # label("Df: u2").  [assumption].
33 u2(x,y) = y v c(y v c(x)).  [copy(32),rewrite([23(3)])].
40 -le(x,y) | x ^ y = x # label("Df: less than").  [clausify(1)].
41 -le(x,y) | c(c(x) v c(y)) = x.  [copy(40),rewrite([18(2)])].
44 -perp(x,y) | le(x,c(y)) # label("Df: perpendicular").  [clausify(2)].
46 u2(x,y) != 1 | perp(c(x),c(y)) # label("Hypothesis for Proposition 2.11u2").
[clausify(4)].
47 x v c(x v c(y)) != 1 | perp(c(y),c(x)).  [copy(46),rewrite([33(1)])].
50 c1 v (c(c1) ^ (c2 v c1)) != c2 v c1 # label("OMA") # answer("OMA").  [deny(5)].
51 c1 v c(c1 v c(c1 v c2)) != c1 v c2 # answer("OMA").
[copy(50),rewrite([10(6),18(7),9(4),10(12)])].
52 c(1) = 0.  [back_rewrite(15),rewrite([18(2),9(2),17(2)])].
53 c(c(x) v c(x v y)) = x.  [back_rewrite(14),rewrite([18(2)])].
54 x v c(c(x) v c(y)) = x.  [back_rewrite(13),rewrite([18(1)])].
56 x v (y v z) = y v (x v z).  [para(10(a,1),11(a,1,1)),rewrite([11(2)])].
60 x v (y v c(x v y)) = 1.  [para(17(a,1),11(a,1)),flip(a)].
66 x v c(x v y) != 1 | perp(y,c(x)).  [para(9(a,1),47(a,1,2,1,2)),rewrite([9(7)])].
78 c(x) v c(x v y) = c(x).  [para(53(a,1),9(a,1,1)),flip(a)].
82 c(0 v c(x)) = x.  [para(17(a,1),53(a,1,1,2,1)),rewrite([52(3),10(3)])].
88 x v c(c(x) v y) = x.  [para(9(a,1),54(a,1,2,1,2))].
93 x v x = x.  [para(52(a,1),54(a,1,2,1,2)),rewrite([10(3),82(4)])].
110 x v (x v y) = x v y.  [para(93(a,1),11(a,1,1)),flip(a)].
113 x v (y v c(y v x)) = 1.  [para(10(a,1),60(a,1,2,2,1))].
144 x v c(y v x) != 1 | perp(y,c(x)).  [para(10(a,1),66(a,1,2,1))].
160 c(x) v c(y v x) = c(x).  [para(10(a,1),78(a,1,2,1))].
183 x v c(y v c(x)) = x.  [para(10(a,1),88(a,1,2,1))].
185 x v (y v c(c(x v y) v z)) = x v y.  [para(88(a,1),11(a,1)),flip(a)].
273 x v (y v c(z v c(x v y))) = x v y.  [para(183(a,1),11(a,1)),flip(a)].
397 c(x v y) v c(x v (z v y)) = c(x v y).  [para(56(a,1),160(a,1,2,1))].
```

```
1173 x v (c(y v x) v c(c(y v x) v z)) != 1 | perp(y,c(x v c(c(y v x) v z))).
[para(185(a,1),144(a,1,2,1)),rewrite([10(8),56(8)])].
7388 c(x v y) v c(x v c(z v c(x v y))) = c(x v c(z v c(x v y))).
[para(273(a,1),397(a,1,2,1)),rewrite([10(9)])].
22475 perp(x,c(y v c(y v c(x v y)))).  [hyper(1173,a,113,a),rewrite([10(3)])].
22476 le(x,y v c(y v c(x v y))).  [hyper(44,a,22475,a),rewrite([9(7)])].
22486 le(x,y v c(y v c(y v x))).  [para(10(a,1),22476(a,2,2,1,2,1))].
22497 le(x v y,x v c(x v c(x v y))).  [para(110(a,1),22486(a,2,2,1,2,1))].
22518 x v c(x v c(x v y)) = x v y.  [hyper(41,a,22497,a),rewrite([7388(9),9(7)])].
22519 $F # answer("OMA").  [resolve(22518,a,51,a)].

============================== end of proof ==========================



============================== PROOF =================================

% Proof 1 at 91.76 (+ 2.37) seconds: "OMA".
% Length of proof is 56.
% Level of proof is 14.

1 le(x,y) <-> x = x ^ y # label("Df: less than") # label(non_clause).  [assumption].
2 perp(x,y) <-> le(x,c(y)) # label("Df: perpendicular") # label(non_clause).
[assumption].
4 u3(x,y) = 1 <-> perp(c(x),c(y)) # label("Hypothesis for Proposition 2.11u3") #
label(non_clause).  [assumption].
5 y v (c(y) ^ (x v y)) = x v y # label("OMA") # label(non_clause) # label(goal).  [goal].
8 x = c(c(x)) # label("AxL1").  [assumption].
9 c(c(x)) = x.  [copy(8),flip(a)].
10 x v y = y v x # label("AxL2").  [assumption].
11 (x v y) v z = x v (y v z) # label("AxL3").  [assumption].
13 x v (x ^ y) = x # label("AxL5").  [assumption].
14 x ^ (x v y) = x # label("AxL6").  [assumption].
15 c(x) ^ x = 0 # label("AxOL1").  [assumption].
16 c(x) v x = 1 # label("AxOL2").  [assumption].
17 x v c(x) = 1.  [copy(16),rewrite([10(2)])].
18 x ^ y = c(c(x) v c(y)) # label("AxOL3").  [assumption].
24 i3(x,y) = ((c(x) ^ y) v (c(x) ^ c(y))) v (c(x) v (x ^ y)) # label("Df: i3").
[assumption].
25 i3(x,y) = c(x v y) v (c(x v c(y)) v (c(x) v c(c(x) v c(y)))).
[copy(24),rewrite([18(3),9(3),18(7),9(6),9(6),10(7),18(9),11(14)])].
34 u3(x,y) = i3(c(x),y) # label("Df: u3").  [assumption].
35 u3(x,y) = c(c(x) v y) v (x v (c(x v c(y)) v c(c(x) v c(y)))).
[copy(34),rewrite([25(3),9(10),9(10),10(13),11(13)])].
40 -le(x,y) | x ^ y = x # label("Df: less than").  [clausify(1)].
41 -le(x,y) | c(c(x) v c(y)) = x.  [copy(40),rewrite([18(2)])].
44 -perp(x,y) | le(x,c(y)) # label("Df: perpendicular").  [clausify(2)].
46 u3(x,y) != 1 | perp(c(x),c(y)) # label("Hypothesis for Proposition 2.11u3").
[clausify(4)].
47 c(c(x) v y) v (x v (c(x v c(y)) v c(c(x) v c(y)))) != 1 | perp(c(x),c(y)).
[copy(46),rewrite([35(1)])].
50 c1 v (c(c1) ^ (c2 v c1)) != c2 v c1 # label("OMA") # answer("OMA").  [deny(5)].
51 c1 v c(c1 v c(c1 v c2)) != c1 v c2 # answer("OMA").
[copy(50),rewrite([10(6),18(7),9(4),10(12)])].
52 c(1) = 0.  [back_rewrite(15),rewrite([18(2),9(2),17(2)])].
53 c(c(x) v c(x v y)) = x.  [back_rewrite(14),rewrite([18(2)])].
54 x v c(c(x) v c(y)) = x.  [back_rewrite(13),rewrite([18(1)])].
56 x v (y v z) = y v (x v z).  [para(10(a,1),11(a,1,1)),rewrite([11(2)])].
58 x v (c(c(x) v y) v (c(x v c(y)) v c(c(x) v c(y)))) != 1 | perp(c(x),c(y)).
[back_rewrite(47),rewrite([56(13)])].
62 x v (y v c(x v y)) = 1.  [para(17(a,1),11(a,1)),flip(a)].
74 c(x) v c(x v y) = c(x).  [para(53(a,1),9(a,1,1)),flip(a)].
78 c(0 v c(x)) = x.  [para(17(a,1),53(a,1,1,2,1)),rewrite([52(3),10(3)])].
79 1 v x = 1.  [para(52(a,1),53(a,1,1,1)),rewrite([78(6)])].
84 x v c(c(x) v y) = x.  [para(9(a,1),54(a,1,2,1,2))].
88 x v 0 = x.  [para(17(a,1),54(a,1,2,1)),rewrite([52(2)])].
89 x v x = x.  [para(52(a,1),54(a,1,2,1,2)),rewrite([10(3),78(4)])].
90 x v (y v c(x)) = y v 1.  [para(17(a,1),56(a,1,2)),flip(a)].
103 x v 1 = 1.  [para(79(a,1),10(a,1)),flip(a)].
105 x v (y v c(x)) = 1.  [back_rewrite(90),rewrite([103(5)])].
106 0 v x = x.  [para(88(a,1),10(a,1)),flip(a)].
```

```
111 x v (c(x v y) v (c(c(x) v y) v c(c(x) v c(y)))) != 1 | perp(c(x),y).
[para(9(a,1),58(a,1,2,2,1,1,2)),rewrite([9(9),10(11),11(11),9(17)])].
125 x v (x v y) = x v y.  [para(89(a,1),11(a,1,1)),flip(a)].
127 x v (y v x) = y v x.  [para(89(a,1),11(a,2,2)),rewrite([10(2)])].
130 x v (y v c(y v x)) = 1.  [para(10(a,1),62(a,1,2,2,1))].
168 c(x) v c(y v x) = c(x).  [para(10(a,1),74(a,1,2,1))].
193 x v c(y v c(x)) = x.  [para(10(a,1),84(a,1,2,1))].
195 x v (y v c(c(x v y) v z)) = x v y.  [para(84(a,1),11(a,1)),flip(a)].
250 x v c(y v x) != 1 | perp(c(x),y v x).
[para(105(a,1),111(a,1,2,2,1,1)),rewrite([9(2),127(2),52(4),9(6),168(7),9(5),106(4),10(3)
,125(4),9(8)])].
1917 x v (c(y v x) v c(c(y v x) v z)) != 1 | perp(c(x v c(c(y v x) v z)),y v x).
[para(195(a,1),250(a,1,2,1)),rewrite([10(8),56(8),195(22)])].
25067 perp(c(x v c(x v c(y v x))),y v x).  [hyper(1917,a,130,a),rewrite([10(3)])].
25068 le(c(x v c(x v c(y v x))),c(y v x)).  [hyper(44,a,25067,a)].
25087 c(x v c(x v c(y v x))) = c(y v x).
[hyper(41,a,25068,a),rewrite([9(7),9(8),10(7),56(7),193(6),127(2)]),flip(a)].
25091 x v c(x v c(y v x)) = y v x.  [para(25087(a,1),9(a,1,1)),rewrite([9(3)]),flip(a)].
25119 x v c(x v c(x v y)) = y v x.  [para(10(a,1),25091(a,1,2,1,2,1))].
25164 $F # answer("OMA").  [back_rewrite(51),rewrite([25119(9),10(3)]),xx(a)].

============================== end of proof ==========================
```

Figure 3. Summary of a *prover9* ([2]) proof of Proposition 2.11 ([5]), for each of i = 1,2,3 . The proofs assume the default inference rules of *prover9*. The general form of a line in this proof is "*line_number conclusion [derivation]*", where *line_number* is a unique identifier of a line in the proof, and *conclusion* is the result of applying the *prover9* inference rules (such as *para*modulation, *copy*ing, and *rewrit*ing), noted in square brackets (denoting the *derivation*), to the lines cited in those brackets. The detailed syntax and semantics of these notations can be found in [2]. All *prover9* proofs are by default proofs by contradiction.

The total time to produce the proofs in Figure 3 on the platform described in Section 2.0 was approximately 140 seconds.

4.0 Discussion

The results of Section 3.0 motivate several observations:

1. With the exception of Proposition 2.11, the proofs in Figure 3 for i=2 and i=3 are symmetric in their derivational dependencies and useL1, L2, L5, L6, OL1, OL2, and OL3. The proof for i=1 uses L1, L2, L3, L5, OL2, and OL3.

2. The proofs in Section 3.0 may be novel.

3. Companion papers provide proofs for i = 4,5, and for orthomodular lattice theory implies Propositions 2.11i, i = 1,2,3,4,5.

4. Proposition 2.13 can be regarded as a definition of quantum union; thus, this paper together with the papers mentioned in (3), constitute a proof that the definition of quantum intersection is equivalent to the OMA in orthomodular quantum logic. Companion papers derive equivalences for the OMA with definitions of quantum-intersection and quantum-identity. Collectively, these papers provide a theory of equivalence of the OMA with the quantum connectives. In light of these equivalences, QL without the OMA would hardly qualify as a logic.

5.0 Acknowledgements

This work benefited from discussions with Tom Oberdan, Frank Pecchioni, Tony Pawlicki, and the late John K. Prentice, whose passion for foundations of physics inspired those of us privileged to have known him. For any infelicities that remain, I am solely responsible.

6.0 References

[1] von Neumann J. *Mathematical Foundations of Quantum Mechanics*. 1936. Translated by R. T. Beyer. Princeton. 1983.
[2] McCune WW. *prover9 and mace4.* URL http://www.cs.unm.edu/~mccune/prover9/. 2009.
[3] Horner JK. MHT *prover9* scripts. 2011. Available from the author on request.
[4] Dalla Chiara ML and Giuntini R. *Quantum Logics*. URL http://xxx.lanl.gov/abs/quant-ph/0101028. 2004.
[5] Megill ND and Pavičić M. Orthomodular lattices and quantum algebra. *International Journal of Theoretical Physics* 40 (2001), pp. 1387-1410.
[6] Akhiezer NI and Glazman IM. *Theory of Linear Operators in Hilbert Space. Volume I.* Translated by M. Nestell. Frederick Ungar. 1961.
[7] Holland, Jr. SS Orthomodularity in infinite dimensions: a theorem of M. Solèr. *Bulletin of the American Mathematical Society* 32 (1995), pp. 205-234.
[8] Marsden EL and Herman LM. *A condition for distribution in orthomodular lattices.* Kansas State University Technical Report #40. 1974.
[9] Knuth DE and Bendix PB. Simple word problems in universal algebras. In J. Leech, ed. *Computational Problems in Abstract Algebra.* Pergamon Press. 1970. pp. 263-297.
[10] Chang CC and Keisler HJ. *Model Theory.* North-Holland. 1990. pp. 38-39.
30.
[11] Birkhoff G. *Lattice Theory.* Third Edition. American Mathematical Society. 1967.
[12] Church A. *Introduction to Mathematical Logic. Volume I.* Princeton. 1956.
[13] Jauch J. *Foundations of Quantum Mechanics.* Addison-Wesley. 1968.
[14] Megill ND. *Metamath.* URL http://us.metamath.org/qlegif/mmql.html#unify. 2004.
[15] Horner JK. An automated deduction system for orthomodular lattice theory. *Proceedings of the 2005 International Conference on Artificial Intelligence.* CSREA Press. 2005. pp. 260-265.
[16] Horner JK. An automated equational logic deduction of join elimination in orthomodular lattice theory. *Proceedings of the 2007 International Conference on Artificial Intelligence.* CSREA Press. 2007. pp. 481-488.
[17] Messiah A. *Quantum Mechanics*. Dover. 1958.
[18] Horner JK. Using automated theorem-provers to aid the design of efficient compilers for quantum computing. *Los Alamos National Laboratory Quantum Institute Workshop.* December 9–10, 2002. URL http://www.lanl.gov/science/centers/quantum/qls_pdfs/horner.pdf.
[19] Birkhoff G and von Neumann J. The logic of quantum mechanics. *Annals of Mathematics* 37 (1936), 823-243.
[20] Nielsen MA and Chuang L . *Quantum Computation and Quantum Information*. Cambridge. 2000.
[21] Pavičić M and Megill N. Quantum and classical implicational algebras with primitive implication. *International Journal of Theoretical Physics* 37 (1998), 2091-2098. ftp://m3k.grad.hr/pavicic/quantum-logic/1998-int-j-theor-phys-2.ps.gz.
[22] Horner JK. An automated deduction of the relative strength of orthomodular and weakly orthomodular lattice theory. *Proceedings of the 2009 International Conference on Artificial Intelligence*. CSREA Press. 2009. pp. 525-5

Quantum-Union Equivalents of the Orthomodularity Law in Quantum Logic: Part 2

Jack K. Horner
P. O. Box 266
Los Alamos, New Mexico 87544 USA
jhorner@cybermesa.com

FCS 2014

Abstract

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of quantum-mechanical behaviors and operations. In much the same way that the structure of conventional propositional (Boolean) logic (BL) is the logic of the description of the behavior of classical physical systems and is isomorphic to a Boolean algebra (BA), so also the algebra, C(H), of closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space is a logic of the descriptions of the behavior of quantum mechanical systems and is a model of an ortholattice (OL). An OL can thus be thought of as a kind of "quantum logic" (QL). C(H) is also a model of an orthomodular lattice, which is an OL conjoined with the orthomodularity axiom (OMA). The rationalization of the OMA as a claim proper to physics has proven problematic, motivating the question of whether the OMA and its equivalents are required in an adequate characterization of QL. Here I provide an automated deduction of two quantum-union-based equivalents of the OMA. The proofs may be novel.

Keywords: automated deduction, quantum computing, orthomodular lattice, Hilbert space

1.0 Introduction

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of the description of quantum-mechanical behaviors ([1], [17], [18], [20]). In much the same way that conventional propositional (Boolean) logic (BL,[12]) is the logical structure of description of the behavior of classical physical systems (e.g. "the measurements of the position and momentum of particle P are commutative", i.e., can be measured in either other, yielding the same results) and is isomorphic to a Boolean lattice ([10], [11], [19]), so also the algebra, *C(H)*, of the closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space *H* ([1], [4], [6], [9], [13]) is *a* logic of the descriptions of the behavior of quantum mechanical systems (e.g., "the measurements of the position and momentum of particle P are *not* commutative") and is a model ([10]) of an ortholattice (OL; [4]). An OL can thus be thought of as a kind of "quantum logic" (QL; [19]). *C(H)* is also a model of (i.e., isomorphic to a set of sentences which hold in) an orthomodular lattice (OML; [4], [7]), which is an OL conjoined with the orthomodularity axiom (OMA; see Figure 1). The rationalization of the OMA as a claim proper to physics has proven problematic ([13], Section 5-6), motivating the question of whether the OMA is required in an adequate characterization of QL. Thus formulated, the question suggests that the OMA and its equivalents are specific to an OML, and that as a consequence, banning the OMA from QL yields a "truer" quantum logic.

Lattice axioms

```
      x = c(c(x))                       (AxLat1)
      x v y = y v x                     (AxLat2)
      (x v y) v z = x v (y v z)         (AxLat3)
      (x ^ y) ^ z = x ^ (y ^ z)         (AxLat4)
      x v (x ^ y) = x                   (AxLat5)
      x ^ (x v y) = x                   (AxLat6)
```

Ortholattice axioms

```
      c(x) ^ x = 0                      (AxOL1)
      c(x) v x = 1                      (AxOL2)
      x ^ y = c(c(x) v c(y))            (AxOL3)
```

Orthomodularity axiom

```
      y v (c(y) ^ (x v y)) = x v y      (OMA)
```

Definitions of implications and partial order

```
   i1(x,y) = c(x) v (x ^ y).
   i2(x,y) = i1(c(y), c(x).
   i3(x,y) = (c(x) ^ y) v (c(x) ^ c(y)) v i1(x,y).
   i4(x,y) = i3(c(y), c(x)).
   i5(x,y) = (x ^ y) v (c(x) ^ y) v (c(x) ^ c(y)).
   le(x,y) = (x = (x ^ y)).
```

Definitions of indexed unions

```
   u1(x,y) =  i1(c(x),y)
   u2(x,y) =  i2(c(x),y)
   u3(x,y) =  i3(c(x),y)
   u4(x,y) =  i4(c(x),y)
   u5(x,y) =  i5(c(x),y)
```

where
   x, y are variables ranging over lattice nodes
   ^ is lattice meet
   v is lattice join
   c(x) is the orthocomplement of x
   i1(x,y) means $x \rightarrow_1 y$ (Sasaki implication)
   i2(x,y) means $x \rightarrow_2 y$ (Dishkant implication)
   i3(x,y) means $x \rightarrow_3 y$ (Kalmbach implication)
   i4(x,y) means $x \rightarrow_4 y$ (non-tollens implication)
   i5(x,y) means $x \rightarrow_5 y$ (relevance implication)
   le(x,y) means $x \leq y$
   <-> means if and only if
   = is equivalence ([12])
   1 is the maximum lattice element (= x v c(x))
   0 is the minimum lattice element (= c(1))

Figure 1. Lattice, ortholattice, orthomodularity axioms, and some definitions.

There are at least 21 nominal equivalents (in the sense that ortholattice theory, together with these "equivalents", imply the OMA, and vice versa) of the OMA in quantum logic ([5], Theorem 2.5); as nominal equivalents, they are thus of import to optimizing quantum circuit design. Among these is the Proposition shown in Figure 2:

$$x \cup_i y \quad <-> \quad c(x) \perp c(y)$$

where

$x \cup_i y$ means $c(x) \rightarrow_i y$
$x \perp y$ means $le(x,c(y))$
$i = 1,2,3,4,5$

Figure 2. Proposition 2.11 of [5]

2.0 Method

The OML axiomatizations of Megill, Pavičić, and Horner ([5], [14], [15], [16], [21], [22]) were implemented in a *prover9* ([2]) script ([3]) configured to derive OMA from Proposition 2.11 of [5], for each of i = 4,5 together with ortholattice theory (orthomodular lattice theory, without the OMA), then executed in that framework on a Dell Inspiron 545 with an Intel Core2 Quad CPU Q8200 (clocked @ 2.33 GHz) and 8.00 GB RAM, running under the *Windows Vista Home Premium /Cygwin* operating environment.

3.0 Results

Figure 3 shows the proofs, generated by [3] on the platform described in Section 2.0, that Proposition 2.11 (for each of i = 4,5), together with ortholattice theory, imply the OMA.

```
============================== PROOF =================================

% Proof 1 at 169.67 (+ 3.56) seconds: "OMA".
% Length of proof is 54.
% Level of proof is 14.

1 le(x,y) <-> x = x ^ y # label("Df: less than") # label(non_clause).  [assumption].
2 perp(x,y) <-> le(x,c(y)) # label("Df: perpendicular") # label(non_clause).
[assumption].
4 u4(x,y) = 1 <-> perp(c(x),c(y)) # label("Hypothesis for Proposition 2.10u4") #
label(non_clause).  [assumption].
5 y v (c(y) ^ (x v y)) = x v y # label("OMA") # label(non_clause) # label(goal).  [goal].
8 x = c(c(x)) # label("AxL1").  [assumption].
9 c(c(x)) = x.  [copy(8),flip(a)].
10 x v y = y v x # label("AxL2").  [assumption].
11 (x v y) v z = x v (y v z) # label("AxL3").  [assumption].
13 x v (x ^ y) = x # label("AxL5").  [assumption].
14 x ^ (x v y) = x # label("AxL6").  [assumption].
15 c(x) ^ x = 0 # label("AxOL1").  [assumption].
16 c(x) v x = 1 # label("AxOL2").  [assumption].
17 x v c(x) = 1.  [copy(16),rewrite([10(2)])].
18 x ^ y = c(c(x) v c(y)) # label("AxOL3").  [assumption].
26 i4(x,y) = ((c(c(y)) ^ c(x)) v (c(c(y)) ^ c(c(x)))) v (c(c(y)) v (c(y) ^ c(x))) #
label("Df: i4").  [assumption].
27 i4(x,y) = y v (c(y v x) v (c(c(y) v x) v c(c(y) v c(x)))).
[copy(26),rewrite([9(3),18(3),9(4),9(6),9(6),18(5),9(11),18(12),9(11),9(11),10(13),11(13)
])].
36 u4(x,y) = i4(c(x),y) # label("Df: u4").  [assumption].
37 u4(x,y) = y v (c(y v c(x)) v (c(c(y) v x) v c(c(y) v c(x)))).
[copy(36),rewrite([27(3),9(11),10(12)])].
40 -le(x,y) | x ^ y = x # label("Df: less than").  [clausify(1)].
41 -le(x,y) | c(c(x) v c(y)) = x.  [copy(40),rewrite([18(2)])].
44 -perp(x,y) | le(x,c(y)) # label("Df: perpendicular").  [clausify(2)].
```

```
46 u4(x,y) != 1 | perp(c(x),c(y)) # label("Hypothesis for Proposition 2.10u4").
[clausify(4)].
47 x v (c(x v c(y)) v (c(c(x) v y) v c(c(x) v c(y)))) != 1 | perp(c(y),c(x)).
[copy(46),rewrite([37(1)])].
50 c1 v (c(c1) ^ (c2 v c1)) != c2 v c1 # label("OMA") # answer("OMA").  [deny(5)].
51 c1 v c(c1 v c(c1 v c2)) != c1 v c2 # answer("OMA").
[copy(50),rewrite([10(6),18(7),9(4),10(12)])].
52 c(1) = 0.  [back_rewrite(15),rewrite([18(2),9(2),17(2)])].
53 c(c(x) v c(x v y)) = x.  [back_rewrite(14),rewrite([18(2)])].
54 x v c(c(x) v c(y)) = x.  [back_rewrite(13),rewrite([18(1)])].
56 x v (y v z) = y v (x v z).  [para(10(a,1),11(a,1,1)),rewrite([11(2)])].
60 x v (y v c(x v y)) = 1.  [para(17(a,1),11(a,1)),flip(a)].
68 x v (c(c(y) v x) v (c(c(x) v y) v c(c(x) v c(y)))) != 1 | perp(c(y),c(x)).
[para(10(a,1),47(a,1,2,1,1))].
86 c(x) v c(x v y) = c(x).  [para(53(a,1),9(a,1,1)),flip(a)].
90 c(0 v c(x)) = x.  [para(17(a,1),53(a,1,1,2,1)),rewrite([52(3),10(3)])].
94 1 v x = 1.  [para(52(a,1),53(a,1,1,1)),rewrite([90(6)])].
107 x v c(c(x) v y) = x.  [para(9(a,1),54(a,1,2,1,2))].
113 x v x = x.  [para(52(a,1),54(a,1,2,1,2)),rewrite([10(3),90(4)])].
144 x v (y v c(y v x)) = 1.  [para(10(a,1),60(a,1,2,2,1))].
150 x v (x v y) = x v y.  [para(113(a,1),11(a,1,1)),flip(a)].
218 x v (y v (c(c(z) v (x v y)) v (c(c(x v y) v z) v c(c(x v y) v c(z))))) != 1 |
perp(c(z),c(x v y)).  [para(11(a,1),68(a,1))].
289 x v c(y v c(x)) = x.  [para(10(a,1),107(a,1,2,1))].
291 x v (y v c(c(x v y) v z)) = x v y.  [para(107(a,1),11(a,1)),flip(a)].
345 c(x) v c(y v x) = c(x).  [para(10(a,1),86(a,1,2,1))].
389 x v (y v (c(y v x) v z)) = 1.
[para(144(a,1),11(a,1,1)),rewrite([94(2),11(5)]),flip(a)].
401 x v (y v c(z v c(x v y))) = x v y.  [para(289(a,1),11(a,1)),flip(a)].
616 c(x v y) v c(x v (z v y)) = c(x v y).  [para(56(a,1),345(a,1,2,1))].
2746 x v (c(c(y) v x) v (c(c(c(y) v x) v z) v (c(y v c(x v c(c(c(y) v x) v z))) v c(c(y)
v c(x v c(c(c(y) v x) v z)))))) != 1 | perp(c(y),c(x v c(c(c(y) v x) v z))).
[para(291(a,1),218(a,1,2,2,1,1)),rewrite([10(16),10(26),56(30)])].
11022 c(x v y) v c(x v c(z v c(x v y))) = c(x v c(z v c(x v y))).
[para(401(a,1),616(a,1,2,1)),rewrite([10(9)])].
26625 perp(c(x),c(y v c(y v c(c(x) v y)))).  [hyper(2746,a,389,a),rewrite([10(5)])].
26627 perp(x,c(y v c(y v c(x v y)))).  [para(9(a,1),26625(a,1)),rewrite([9(2)])].
26631 le(x,y v c(y v c(x v y))).  [hyper(44,a,26627,a),rewrite([9(7)])].
26639 le(x,y v c(y v c(y v x))).  [para(10(a,1),26631(a,2,2,1,2,1))].
26650 le(x v y,x v c(x v c(x v y))).  [para(150(a,1),26639(a,2,2,1,2,1))].
26669 x v c(x v c(x v y)) = x v y.  [hyper(41,a,26650,a),rewrite([11022(9),9(7)])].
26670 $F # answer("OMA").  [resolve(26669,a,51,a)].

============================== end of proof ==========================



============================== PROOF =================================

% Proof 1 at 244.24 (+ 5.18) seconds: "OMA".
% Length of proof is 53.
% Level of proof is 14.

1 le(x,y) <-> x = x ^ y # label("Df: less than") # label(non_clause).  [assumption].
2 perp(x,y) <-> le(x,c(y)) # label("Df: perpendicular") # label(non_clause).
[assumption].
4 u5(x,y) = 1 <-> perp(c(x),c(y)) # label("Hypothesis for Proposition 2.10u5") #
label(non_clause).  [assumption].
5 y v (c(y) ^ (x v y)) = x v y # label("OMA") # label(non_clause) # label(goal).  [goal].
8 x = c(c(x)) # label("AxL1").  [assumption].
9 c(c(x)) = x.  [copy(8),flip(a)].
10 x v y = y v x # label("AxL2").  [assumption].
11 (x v y) v z = x v (y v z) # label("AxL3").  [assumption].
13 x v (x ^ y) = x # label("AxL5").  [assumption].
14 x ^ (x v y) = x # label("AxL6").  [assumption].
15 c(x) ^ x = 0 # label("AxOL1").  [assumption].
16 c(x) v x = 1 # label("AxOL2").  [assumption].
17 x v c(x) = 1.  [copy(16),rewrite([10(2)])].
18 x ^ y = c(c(x) v c(y)) # label("AxOL3").  [assumption].
28 i5(x,y) = ((x ^ y) v (c(x) ^ y)) v (c(x) ^ c(y)) # label("Df: i5").  [assumption].
```

```
29 i5(x,y) = c(x v y) v (c(x v c(y)) v c(c(x) v c(y))).
[copy(28),rewrite([18(2),18(7),9(7),10(9),18(12),9(11),9(11),10(12)])].
38 u5(x,y) = i5(c(x),y) # label("Df: u5").  [assumption].
39 u5(x,y) = c(c(x) v y) v (c(x v c(y)) v c(c(x) v c(y))).
[copy(38),rewrite([29(3),9(10),10(12)])].
40 -le(x,y) | x ^ y = x # label("Df: less than").  [clausify(1)].
41 -le(x,y) | c(c(x) v c(y)) = x.  [copy(40),rewrite([18(2)])].
44 -perp(x,y) | le(x,c(y)) # label("Df: perpendicular").  [clausify(2)].
46 u5(x,y) != 1 | perp(c(x),c(y)) # label("Hypothesis for Proposition 2.10u5").
[clausify(4)].
47 c(c(x) v y) v (c(x v c(y)) v c(c(x) v c(y))) != 1 | perp(c(x),c(y)).
[copy(46),rewrite([39(1)])].
50 c1 v (c(c1) ^ (c2 v c1)) != c2 v c1 # label("OMA") # answer("OMA").  [deny(5)].
51 c1 v c(c1 v c(c1 v c2)) != c1 v c2 # answer("OMA").
[copy(50),rewrite([10(6),18(7),9(4),10(12)])].
52 c(1) = 0.  [back_rewrite(15),rewrite([18(2),9(2),17(2)])].
53 c(c(x) v c(x v y)) = x.  [back_rewrite(14),rewrite([18(2)])].
54 x v c(c(x) v c(y)) = x.  [back_rewrite(13),rewrite([18(1)])].
56 x v (y v z) = y v (x v z).  [para(10(a,1),11(a,1,1)),rewrite([11(2)])].
60 x v (y v c(x v y)) = 1.  [para(17(a,1),11(a,1)),flip(a)].
67 c(x v y) v (c(c(x) v y) v c(c(x) v c(y))) != 1 | perp(c(x),y).
[para(9(a,1),47(a,1,2,1,1,2)),rewrite([9(9),10(11),11(11),9(16)])].
88 c(c(x) v c(y v x)) = x.  [para(10(a,1),53(a,1,1,2,1))].
90 c(0 v c(x)) = x.  [para(17(a,1),53(a,1,1,2,1)),rewrite([52(3),10(3)])].
94 1 v x = 1.  [para(52(a,1),53(a,1,1,1)),rewrite([90(6)])].
106 x v c(c(x) v y) = x.  [para(9(a,1),54(a,1,2,1,2))].
110 x v 0 = x.  [para(17(a,1),54(a,1,2,1)),rewrite([52(2)])].
111 x v x = x.  [para(52(a,1),54(a,1,2,1,2)),rewrite([10(3),90(4)])].
113 x v (y v c(x)) = y v 1.  [para(17(a,1),56(a,1,2)),flip(a)].
132 x v 1 = 1.  [para(94(a,1),10(a,1)),flip(a)].
135 x v (y v c(x)) = 1.  [back_rewrite(113),rewrite([132(5)])].
136 x v (y v c(y v x)) = 1.  [para(10(a,1),60(a,1,2,2,1))].
142 0 v x = x.  [para(110(a,1),10(a,1)),flip(a)].
151 x v (y v x) = y v x.  [para(111(a,1),11(a,2,2)),rewrite([10(2)])].
218 x v c(y v x) != 1 | perp(c(x),y v x).
[para(135(a,1),67(a,1,2,1,1)),rewrite([9(2),151(2),52(4),9(6),88(8),142(4),10(3),9(8)])].
246 x v c(y v c(x)) = x.  [para(10(a,1),106(a,1,2,1))].
248 x v (y v c(c(x v y) v z)) = x v y.  [para(106(a,1),11(a,1)),flip(a)].
2992 x v (c(y v x) v c(c(y v x) v z)) != 1 | perp(c(x v c(c(y v x) v z)),y v x).
[para(248(a,1),218(a,1,2,1)),rewrite([10(8),56(8),248(22)])].
28598 perp(c(x v c(x v c(y v x))),y v x).  [hyper(2992,a,136,a),rewrite([10(3)])].
28599 le(c(x v c(x v c(y v x))),c(y v x)).  [hyper(44,a,28598,a)].
28618 c(x v c(x v c(y v x))) = c(y v x).
[hyper(41,a,28599,a),rewrite([9(7),9(8),10(7),56(7),246(6),151(2)]),flip(a)].
28622 x v c(x v c(y v x)) = y v x.  [para(28618(a,1),9(a,1,1)),rewrite([9(3)]),flip(a)].
28766 x v c(x v c(x v y)) = y v x.  [para(10(a,1),28622(a,1,2,1,2,1))].
28818 $F # answer("OMA").  [back_rewrite(51),rewrite([28766(9),10(3)]),xx(a)].

============================== end of proof ==========================
```

Figure 3. Summary of a *prover9* ([2]) proof of Proposition 2.11, for each of i = 4,5. The proofs assume the default inference rules of *prover9*. The general form of a line in this proof is "*line_number conclusion [derivation]*", where *line_number* is a unique identifier of a line in the proof, and *conclusion* is the result of applying the *prover9* inference rules (such as *para*modulation, *copy*ing, and *rewrit*ing), noted in square brackets (denoting the *derivation*), to the lines cited in those brackets. Note that some of "logical" proof lines in the above have been transformed to two text lines, with the *derivation* appearing on a text line following a text line containing the first part of that logical line. The detailed syntax and semantics of these notations can be found in [2]. All *prover9* proofs are by default proofs by contradiction.

The total time to produce the proofs in Figure 3 on the platform described in Section 2.0 was approximately 410 seconds.

4.0 Discussion

The results of Section 3.0 motivate several observations:

1. With the exception of Proposition 2.11, the proofs in Figure 3 for i=4 are symmetric and use L1, L1, L2, L5, L6, OL1, OL2, and OL3.

2. The proofs in Section 3.0 may be novel.

3. Companion papers provide proofs for i = 1,2,3, and for orthomodular lattice theory implies Propositions 2.11i of [5], i = 1,2,3,4,5.

4. Proposition 2.13 can be regarded as a definition of quantum union; thus, this paper together with the papers mentioned in (3), constitute a proof that the definition of quantum intersection is equivalent to the OMA in orthomodular quantum logic. Companion papers derive equivalences for the OMA with definitions of quantum-intersection and quantum-identity. Collectively, these papers provide a theory of equivalence of the OMA with the quantum connectives. In light of these equivalences, QL without the OMA would hardly qualify as a logic.

5.0 Acknowledgements

This work benefited from discussions with Tom Oberdan, Frank Pecchioni, Tony Pawlicki, and the late John K. Prentice, whose passion for foundations of physics inspired those of us privileged to have known him. For any infelicities that remain, I am solely responsible.

6.0 References

[1] von Neumann J. *Mathematical Foundations of Quantum Mechanics*. 1936. Translated by R. T. Beyer. Princeton. 1983.

[2] McCune WW. *prover9 and mace4.* URL http://www.cs.unm.edu/~mccune/prover9/. 2009.

[3] Horner JK. MHT *prover9* scripts. 2011. Available from the author on request.

[4] Dalla Chiara ML and Giuntini R. *Quantum Logics*. URL http://xxx.lanl.gov/abs/quant-ph/0101028. 2004.

[5] Megill ND and Pavičić M. Orthomodular lattices and quantum algebra. *International Journal of Theoretical Physics* 40 (2001), pp. 1387-1410.

[6] Akhiezer NI and Glazman IM. *Theory of Linear Operators in Hilbert Space. Volume I.* Translated by M. Nestell. Frederick Ungar. 1961.

[7] Holland, Jr. SS Orthomodularity in infinite dimensions: a theorem of M. Solèr. *Bulletin of the American Mathematical Society* 32 (1995), pp. 205-234.

[8] Marsden EL and Herman LM. *A condition for distribution in orthomodular lattices.* Kansas State University Technical Report #40. 1974.

[9] Knuth DE and Bendix PB. Simple word problems in universal algebras. In J. Leech, ed. *Computational Problems in Abstract Algebra.* Pergamon Press. 1970. pp. 263-297.

[10] Chang CC and Keisler HJ. *Model Theory*. North-Holland. 1990. pp. 38-39.

[11] Birkhoff G. *Lattice Theory*. Third Edition. American Mathematical Society. 1967.

[12] Church A. *Introduction to Mathematical Logic. Volume I.* Princeton. 1956.

[13] Jauch J. *Foundations of Quantum Mechanics.* Addison-Wesley. 1968.

[14] Megill ND. *Metamath.* URL http://us.metamath.org/qlegif/mmql.html#unify. 2004.

[15] Horner JK. An automated deduction system for orthomodular lattice theory. *Proceedings of the 2005 International Conference on Artificial Intelligence.* CSREA Press. 2005. pp. 260-265.

[16] Horner JK. An automated equational logic deduction of join elimination in orthomodular lattice theory. *Proceedings of the 2007 International Conference on Artificial Intelligence.* CSREA Press. 2007. pp. 481-488.
[17] Messiah A. *Quantum Mechanics*. Dover. 1958.
[18] Horner JK. Using automated theorem-provers to aid the design of efficient compilers for quantum computing. *Los Alamos National Laboratory Quantum Institute Workshop.* December 9–10, 2002. URL http://www.lanl.gov/science/centers/quantum/qls_pdfs/horner.pdf.
[19] Birkhoff G and von Neumann J. The logic of quantum mechanics. *Annals of Mathematics* 37 (1936), 823-243.
[20] Nielsen MA and Chuang L . *Quantum Computation and Quantum Information*. Cambridge. 2000.
[21] Pavičić M and Megill N. Quantum and classical implicational algebras with primitive implication. *International Journal of Theoretical Physics* 37 (1998), 2091-2098. ftp://m3k.grad.hr/pavicic/quantum-logic/1998-int-j-theor-phys-2.ps.gz.
[22] Horner JK. An automated deduction of the relative strength of orthomodular and weakly orthomodular lattice theory. *Proceedings of the 2009 International Conference on Artificial Intelligence*. CSREA Press. 2009. pp. 525-530.

Quantum-Union Equivalents of the Orthomodularity Law in Quantum Logic: Part 3

Jack K. Horner
P. O. Box 266
Los Alamos, New Mexico 87544 USA
jhorner@cybermesa.com

FCS 2014

Abstract

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of quantum-mechanical behaviors and operations. In much the same way that the structure of conventional propositional (Boolean) logic (BL) is the logic of the description of the behavior of classical physical systems and is isomorphic to a Boolean algebra (BA), so also the algebra, C(H), of closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space is a logic of the descriptions of the behavior of quantum mechanical systems and is a model of an ortholattice (OL). An OL can thus be thought of as a kind of "quantum logic" (QL). C(H) is also a model of an orthomodular lattice, which is an OL conjoined with the orthomodularity axiom (OMA). The rationalization of the OMA as a claim proper to physics has proven problematic, motivating the question of whether the OMA and its equivalents are required in an adequate characterization of QL. Here I provide an automated deduction of two quantum-union-based equivalents of the OMA from orthomodular lattice theory. The proofs may be novel.

Keywords: automated deduction, quantum computing, orthomodular lattice, Hilbert space

1.0 Introduction

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of the description of quantum-mechanical behaviors ([1], [17], [18], [20]). In much the same way that conventional propositional (Boolean) logic (BL,[12]) is the logical structure of description of the behavior of classical physical systems (e.g. "the measurements of the position and momentum of particle P are commutative", i.e., can be measured in either other, yielding the same results) and is isomorphic to a Boolean lattice ([10], [11], [19]), so also the algebra, *C(H)*, of the closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space *H* ([1], [4], [6], [9], [13]) is *a* logic of the descriptions of the behavior of quantum mechanical systems (e.g., "the measurements of the position and momentum of particle P are *not* commutative") and is a model ([10]) of an ortholattice (OL; [4]). An OL can thus be thought of as a kind of "quantum logic" (QL; [19]). *C(H)* is also a model of (i.e., isomorphic to a set of sentences which hold in) an orthomodular lattice (OML; [4], [7]), which is an OL conjoined with the orthomodularity axiom (OMA; see Figure 1). The rationalization of the OMA as a claim proper to physics has proven problematic ([13], Section 5-6), motivating the question of whether the OMA is required in an adequate characterization of QL. Thus formulated, the question suggests that the OMA and its equivalents are specific to an OML, and that as a consequence, banning

the OMA from QL yields a "truer" quantum logic.

Lattice axioms

```
      x = c(c(x))                       (AxLat1)
      x v y = y v x                     (AxLat2)
      (x v y) v z = x v (y v z)          (AxLat3)
      (x ^ y) ^ z = x ^ (y ^ z)          (AxLat4)
      x v (x ^ y) = x                   (AxLat5)
      x ^ (x v y) = x                   (AxLat6)
```

Ortholattice axioms

```
      c(x) ^ x = 0                      (AxOL1)
      c(x) v x = 1                      (AxOL2)
      x ^ y = c(c(x) v c(y))            (AxOL3)
```

Orthomodularity axiom

```
      y v (c(y) ^ (x v y)) = x v y      (OMA)
```

Definitions of implications and partial order

```
   i1(x,y) = c(x) v (x ^ y).
   i2(x,y) = i1(c(y), c(x).
   i3(x,y) = (c(x) ^ y) v (c(x) ^ c(y)) v i1(x,y).
   i4(x,y) = i3(c(y), c(x)).
   i5(x,y) = (x ^ y) v (c(x) ^ y) v (c(x) ^ c(y)).
   le(x,y) = (x = (x ^ y)).
```

Definitions of indexed unions

```
   u1(x,y) =  i1(c(x),y)
   u2(x,y) =  i2(c(x),y)
   u3(x,y) =  i3(c(x),y)
   u4(x,y) =  i4(c(x),y)
   u5(x,y) =  i5(c(x),y)
```

where
   x, y are variables ranging over lattice nodes
   ^ is lattice meet
   v is lattice join
   c(x) is the orthocomplement of x
   i1(x,y) means $x \rightarrow_1 y$ (Sasaki implication)
   i2(x,y) means $x \rightarrow_2 y$ (Dishkant implication)
   i3(x,y) means $x \rightarrow_3 y$ (Kalmbach implication)
   i4(x,y) means $x \rightarrow_4 y$ (non-tollens implication)
   i5(x,y) means $x \rightarrow_5 y$ (relevance implication)
   le(x,y) means $x \leq y$
   <-> means if and only if
   = is equivalence ([12])
   1 is the maximum lattice element (= x v c(x))
   0 is the minimum lattice element (= c(1))

Figure 1. Lattice, ortholattice, orthomodularity axioms, and some definitions.

There are at least 21 nominal equivalents (in the sense that ortholattice theory, together with these "equivalents", imply the OMA, and vice versa) of the OMA in quantum logic ([5], Theorem 2.5); as nominal equivalents, they are thus of import to optimizing quantum circuit design. Among these is the Proposition shown in Figure 2:

$$x \cup_i y \quad <-> \quad c(x) \perp c(y)$$

where

$x \cup_i y$ means $c(x) \rightarrow_i y$
$x \perp y$ means $le(x,c(y))$
$i = 1,2,3,4,5$

Figure 2. Proposition 2.11 of [5]

2.0 Method

The OML axiomatizations of Megill, Pavičić, and Horner ([5], [14], [15], [16], [21], [22]) were implemented in a *prover9* ([2]) script ([3]) configured to derive Proposition 2.11 of [5], for each of i = 1,2, from orthomodular lattice theory, then executed in that framework on a Dell Inspiron 545 with an Intel Core2 Quad CPU Q8200 (clocked @ 2.33 GHz) and 8.00 GB RAM, running under the *Windows Vista Home Premium /Cygwin* operating environment.

3.0 Results

Figure 3 shows the proofs, generated by [3] on the platform described in Section 2.0, that orthomodular lattice theory implies Proposition 2.11 of [5] (for each of i = 1,2).

```
============================== PROOF =================================

% Proof 1 at 7.44 (+ 0.23) seconds.
% Length of proof is 54.
% Level of proof is 9.

1 le(x,y) <-> x = x ^ y # label("Df: less than") # label(non_clause).  [assumption].
2 perp(x,y) <-> le(x,c(y)) # label("Df: perpendicular") # label(non_clause).
[assumption].
4 u1(x,y) = 1 <-> perp(c(x),c(y)) # label("Proposition 2.10u1") # label(non_clause) #
label(goal).  [goal].
7 x = c(c(x)) # label("AxL1").  [assumption].
8 c(c(x)) = x.  [copy(7),flip(a)].
9 x v y = y v x # label("AxL2").  [assumption].
10 (x v y) v z = x v (y v z) # label("AxL3").  [assumption].
12 x v (x ^ y) = x # label("AxL5").  [assumption].
13 x ^ (x v y) = x # label("AxL6").  [assumption].
14 c(x) ^ x = 0 # label("AxOL1").  [assumption].
15 c(x) v x = 1 # label("AxOL2").  [assumption].
16 x v c(x) = 1.  [copy(15),rewrite([9(2)])].
17 x ^ y = c(c(x) v c(y)) # label("AxOL3").  [assumption].
18 x v (c(x) ^ (y v x)) = y v x # label("OMA").  [assumption].
19 x v c(x v c(y v x)) = y v x.  [copy(18),rewrite([17(3),8(2)])].
21 i1(x,y) = c(x) v (x ^ y) # label("Df: i1").  [assumption].
22 i1(x,y) = c(x) v c(c(x) v c(y)).  [copy(21),rewrite([17(3)])].
29 i5(x,y) = ((x ^ y) v (c(x) ^ y)) v (c(x) ^ c(y)) # label("Df: i5").  [assumption].
30 i5(x,y) = c(x v y) v (c(x v c(y)) v c(c(x) v c(y))).
[copy(29),rewrite([17(2),17(7),8(7),9(9),17(12),8(11),8(11),9(12)])].
31 u1(x,y) = i1(c(x),y) # label("Df: u1").  [assumption].
32 u1(x,y) = x v c(x v c(y)).  [copy(31),rewrite([22(3),8(3),8(3)])].
41 -le(x,y) | x ^ y = x # label("Df: less than").  [clausify(1)].
42 -le(x,y) | c(c(x) v c(y)) = x.  [copy(41),rewrite([17(2)])].
43 le(x,y) | x ^ y != x # label("Df: less than").  [clausify(1)].
44 le(x,y) | c(c(x) v c(y)) != x.  [copy(43),rewrite([17(2)])].
45 -perp(x,y) | le(x,c(y)) # label("Df: perpendicular").  [clausify(2)].
46 perp(x,y) | -le(x,c(y)) # label("Df: perpendicular").  [clausify(2)].
47 u1(c1,c2) = 1 | perp(c(c1),c(c2)) # label("Proposition 2.10u1").  [deny(4)].
48 c1 v c(c1 v c(c2)) = 1 | perp(c(c1),c(c2)).  [copy(47),rewrite([32(3)])].
49 u1(c1,c2) != 1 | -perp(c(c1),c(c2)) # label("Proposition 2.10u1").  [deny(4)].
```

```
50 c1 v c(c1 v c(c2)) != 1 | -perp(c(c1),c(c2)).  [copy(49),rewrite([32(3)])].
51 c(1) = 0.  [back_rewrite(14),rewrite([17(2),8(2),16(2)])].
52 c(c(x) v c(x v y)) = x.  [back_rewrite(13),rewrite([17(2)])].
53 x v c(c(x) v c(y)) = x.  [back_rewrite(12),rewrite([17(1)])].
55 x v (y v z) = y v (x v z).  [para(9(a,1),10(a,1,1)),rewrite([10(2)])].
58 x v (c(x) v y) = 1 v y.  [para(16(a,1),10(a,1,1)),flip(a)].
66 le(c(x),y) | c(x v c(y)) != c(x).  [para(8(a,1),44(b,1,1,1))].
71 perp(x,c(y)) | -le(x,y).  [para(8(a,1),46(b,2))].
72 c1 v c(c1 v c(c2)) = 1 | le(c(c1),c2).  [resolve(48,b,45,a),rewrite([8(14)])].
79 c(x) v c(x v y) = c(x).  [para(52(a,1),8(a,1,1)),flip(a)].
83 c(0 v c(x)) = x.  [para(16(a,1),52(a,1,1,2,1)),rewrite([51(3),9(3)])].
84 c(x v y) v c(x v c(x v y)) = c(x).
[para(52(a,1),19(a,1,2,1,2)),rewrite([9(5),79(11)])].
85 1 v x = 1.  [para(51(a,1),52(a,1,1,1)),rewrite([83(6)])].
89 x v (c(x) v y) = 1.  [back_rewrite(58),rewrite([85(5)])].
94 x v 0 = x.  [para(16(a,1),53(a,1,2,1)),rewrite([51(2)])].
96 x v x = x.  [para(51(a,1),53(a,1,2,1,2)),rewrite([9(3),83(4)])].
109 0 v x = x.  [para(94(a,1),9(a,1)),flip(a)].
113 x v (x v y) = x v y.  [para(96(a,1),10(a,1,1)),flip(a)].
263 le(c(x),y) | c(c(y) v x) != c(x).  [para(9(a,1),66(b,1,1))].
371 c1 v c(c1 v c(c2)) = 1 | c(c1 v c(c2)) = c(c1).
[resolve(72,b,42,a),rewrite([8(12)])].
18968 c(c1 v c(c2)) = c(c1).
[para(371(a,1),30(a,2,1,1)),rewrite([30(15),8(23),113(22),8(29),55(28),89(28),51(23),9(23
),109(23),9(22),84(22),51(12),8(18),113(17),8(24),55(23),89(23),51(18),9(18),109(18),109(
17)]),flip(b),merge(b)].
18969 -perp(c(c1),c(c2)).  [back_rewrite(50),rewrite([18968(6),16(4)]),xx(a)].
18970 -le(c(c1),c2).  [ur(71,a,18969,a)].
18971 $F.  [ur(263,a,18970,a),rewrite([9(4),18968(5)]),xx(a)].

============================== end of proof ==========================


============================== PROOF =================================

% Proof 1 at 69.53 (+ 1.26) seconds.
% Length of proof is 52.
% Level of proof is 10.

1 le(x,y) <-> x = x ^ y # label("Df: less than") # label(non_clause).  [assumption].
2 perp(x,y) <-> le(x,c(y)) # label("Df: perpendicular") # label(non_clause).
[assumption].
4 u2(x,y) = 1 <-> perp(c(x),c(y)) # label("Proposition 2.10u2") # label(non_clause) #
label(goal).  [goal].
7 x = c(c(x)) # label("AxL1").  [assumption].
8 c(c(x)) = x.  [copy(7),flip(a)].
9 x v y = y v x # label("AxL2").  [assumption].
10 (x v y) v z = x v (y v z) # label("AxL3").  [assumption].
12 x v (x ^ y) = x # label("AxL5").  [assumption].
13 x ^ (x v y) = x # label("AxL6").  [assumption].
14 c(x) ^ x = 0 # label("AxOL1").  [assumption].
15 c(x) v x = 1 # label("AxOL2").  [assumption].
16 x v c(x) = 1.  [copy(15),rewrite([9(2)])].
17 x ^ y = c(c(x) v c(y)) # label("AxOL3").  [assumption].
18 x v (c(x) ^ (y v x)) = y v x # label("OMA").  [assumption].
19 x v c(x v c(y v x)) = y v x.  [copy(18),rewrite([17(3),8(2)])].
23 i2(x,y) = c(c(y)) v (c(y) ^ c(x)) # label("Df: i2").  [assumption].
24 i2(x,y) = y v c(y v x).  [copy(23),rewrite([8(3),17(4),8(3),8(3)])].
33 u2(x,y) = i2(c(x),y) # label("Df: u2").  [assumption].
34 u2(x,y) = y v c(y v c(x)).  [copy(33),rewrite([24(3)])].
41 -le(x,y) | x ^ y = x # label("Df: less than").  [clausify(1)].
42 -le(x,y) | c(c(x) v c(y)) = x.  [copy(41),rewrite([17(2)])].
43 le(x,y) | x ^ y != x # label("Df: less than").  [clausify(1)].
44 le(x,y) | c(c(x) v c(y)) != x.  [copy(43),rewrite([17(2)])].
45 -perp(x,y) | le(x,c(y)) # label("Df: perpendicular").  [clausify(2)].
46 perp(x,y) | -le(x,c(y)) # label("Df: perpendicular").  [clausify(2)].
47 u2(c1,c2) = 1 | perp(c(c1),c(c2)) # label("Proposition 2.10u2").  [deny(4)].
48 c2 v c(c2 v c(c1)) = 1 | perp(c(c1),c(c2)).  [copy(47),rewrite([34(3)])].
49 u2(c1,c2) != 1 | -perp(c(c1),c(c2)) # label("Proposition 2.10u2").  [deny(4)].
50 c2 v c(c2 v c(c1)) != 1 | -perp(c(c1),c(c2)).  [copy(49),rewrite([34(3)])].
51 c(1) = 0.  [back_rewrite(14),rewrite([17(2),8(2),16(2)])].
```

```
52 c(c(x) v c(x v y)) = x.  [back_rewrite(13),rewrite([17(2)])].
53 x v c(c(x) v c(y)) = x.  [back_rewrite(12),rewrite([17(1)])].
72 c2 v c(c2 v c(c1)) = 1 | le(c(c1),c2).  [resolve(48,b,45,a),rewrite([8(14)])].
78 le(x,x v y).  [resolve(52,a,44,b)].
79 c(x) v c(x v y) = c(x).  [para(52(a,1),8(a,1,1)),flip(a)].
84 c(x v y) v c(x v c(x v y)) = c(x).
[para(52(a,1),19(a,1,2,1,2)),rewrite([9(5),79(11)])].
90 x v c(c(x) v y) = x.  [para(8(a,1),53(a,1,2,1,2))].
94 x v 0 = x.  [para(16(a,1),53(a,1,2,1)),rewrite([51(2)])].
95 x v c(y v c(x)) = x.  [para(19(a,1),53(a,1,2,1))].
104 le(x,y v x).  [para(9(a,1),78(a,2))].
109 0 v x = x.  [para(94(a,1),9(a,1)),flip(a)].
219 le(c(c(x) v y),x).  [para(90(a,1),104(a,2))].
229 perp(c(x v y),x).  [resolve(219,a,46,b),rewrite([8(2)])].
234 perp(c(x v y),y).  [para(9(a,1),229(a,1,1))].
290 x v (c(y v c(x)) v z) = x v z.  [para(95(a,1),10(a,1,1)),flip(a)].
371 c2 v c(c2 v c(c1)) = 1 | c(c1 v c(c2)) = c(c1).
[resolve(72,b,42,a),rewrite([8(12)])].
12922 x v c(y v c(y v c(x))) = x v c(y).  [para(84(a,1),290(a,1,2)),flip(a)].
18964 c2 v c(c2 v c(c1)) = 1 | c1 v c(c2) = c1.
[para(371(b,1),8(a,1,1)),rewrite([8(12)]),flip(b)].
65210 c1 v c(c2) = c1.
[para(18964(a,1),12922(a,1,2,1)),rewrite([51(9),9(9),109(9)]),flip(b),merge(b)].
65229 perp(c(c1),c(c2)).  [para(65210(a,1),234(a,1,1))].
65232 c2 v c(c1) = c2.  [para(65210(a,1),95(a,1,2,1))].
65887 $F.  [back_unit_del(50),rewrite([65232(5),16(4)]),xx(a),unit_del(a,65229)].

============================== end of proof ==========================
```

Figure 3. Summary of a *prover9* ([2]) proof of Proposition 2.11from orthomodular lattice theory, for each of i = 1,2. The proofs assume the default inference rules of *prover9*. The general form of a line in this proof is "*line_number conclusion [derivation]*", where *line_number* is a unique identifier of a line in the proof, and *conclusion* is the result of applying the *prover9* inference rules (such as *para*modulation, *copy*ing, and *rewrit*ing), noted in square brackets (denoting the *derivation*), to the lines cited in those brackets. Note that some of "logical" proof lines in the above have been transformed to two text lines, with the *derivation* appearing on a text line following a text line containing the first part of that logical line. The detailed syntax and semantics of these notations can be found in [2]. All *prover9* proofs are by default proofs by contradiction.

The total time to produce the proofs in Figure 3 on the platform described in Section 2.0 was approximately 74 seconds.

4.0 Discussion

The results of Section 3.0 motivate several observations:

1. Both of the proofs in Figure 3 use L1, L2, L3, L5, L6, OL1, OL2, and OL3.

2. The proofs in Section 3.0 may be novel.

3. Companion papers provide proofs for i = 4,5, and for Propositions 2.11i, i = 1,2,3,4,5, implies the OMA.

4. Proposition 2.13 can be regarded as a definition of quantum union; thus, this paper together with the papers mentioned in (3), constitutes a proof that the definition of quantum intersection is equivalent to the OMA in orthomodular quantum logic. Companion papers derive equivalences for the OMA with definitions of quantum-intersection and quantum-identity. Collectively, these papers provide a theory of equivalence of the OMA with the quantum connectives. In light of these equivalences, QL without the OMA would hardly qualify as a logic.

5.0 Acknowledgements

This work benefited from discussions with Tom Oberdan, Frank Pecchioni, Tony Pawlicki, and the late John K. Prentice, whose passion for foundations of physics inspired those of us privileged to have known him. For any infelicities that remain, I am solely responsible.

6.0 References

[1] von Neumann J. *Mathematical Foundations of Quantum Mechanics*. 1936. Translated by R. T. Beyer. Princeton. 1983.

[2] McCune WW. *prover9 and mace4.* URL http://www.cs.unm.edu/~mccune/prover9/. 2009.

[3] Horner JK. MHT *prover9* scripts. 2011. Available from the author on request.

[4] Dalla Chiara ML and Giuntini R. *Quantum Logics*. URL http://xxx.lanl.gov/abs/quant-ph/0101028. 2004.

[5] Megill ND and Pavičić M. Orthomodular lattices and quantum algebra. *International Journal of Theoretical Physics* 40 (2001), pp. 1387-1410.

[6] Akhiezer NI and Glazman IM. *Theory of Linear Operators in Hilbert Space. Volume I.* Translated by M. Nestell. Frederick Ungar. 1961.

[7] Holland, Jr. SS Orthomodularity in infinite dimensions: a theorem of M. Solèr. *Bulletin of the American Mathematical Society* 32 (1995), pp. 205-234.

[8] Marsden EL and Herman LM. *A condition for distribution in orthomodular lattices.* Kansas State University Technical Report #40. 1974.

[9] Knuth DE and Bendix PB. Simple word problems in universal algebras. In J. Leech, ed. *Computational Problems in Abstract Algebra.* Pergamon Press. 1970. pp. 263-297.

[10] Chang CC and Keisler HJ. *Model Theory*. North-Holland. 1990. pp. 38-39.

[11] Birkhoff G. *Lattice Theory*. Third Edition. American Mathematical Society. 1967.

[12] Church A. *Introduction to Mathematical Logic. Volume I.* Princeton. 1956.

[13] Jauch J. *Foundations of Quantum Mechanics.* Addison-Wesley. 1968.

[14] Megill ND. *Metamath.* URL http://us.metamath.org/qlegif/mmql.html#unify. 2004.

[15] Horner JK. An automated deduction system for orthomodular lattice theory. *Proceedings of the 2005 International Conference on Artificial Intelligence*. CSREA Press. 2005. pp. 260-265.

[16] Horner JK. An automated equational logic deduction of join elimination in orthomodular lattice theory. *Proceedings of the 2007 International Conference on Artificial Intelligence*. CSREA Press. 2007. pp. 481-488.

[17] Messiah A. *Quantum Mechanics*. Dover. 1958.

[18] Horner JK. Using automated theorem-provers to aid the design of efficient compilers for quantum computing. *Los Alamos National Laboratory Quantum Institute Workshop.* December 9–10, 2002.

URL
http://www.lanl.gov/science/centers/quantum/qls_pdfs/horner.pdf.

[19] Birkhoff G and von Neumann J. The logic of quantum mechanics. *Annals of Mathematics* 37 (1936), 823-243.
[20] Nielsen MA and Chuang L . *Quantum Computation and Quantum Information*. Cambridge. 2000.

[21] Pavičić M and Megill N. Quantum and classical implicational algebras with primitive implication. *International Journal of Theoretical Physics* 37 (1998), 2091-2098.
ftp://m3k.grad.hr/pavicic/quantum-logic/1998-int-j-theor-phys-2.ps.gz.

[22] Horner JK. An automated deduction of the relative strength of orthomodular and weakly orthomodular lattice theory. *Proceedings of the 2009 International Conference on Artificial Intelligence*. CSREA Press. 2009. pp. 525-530.

Quantum-Union Equivalents of the Orthomodularity Law in Quantum Logic: Part 4

Jack K. Horner
P. O. Box 266
Los Alamos, New Mexico 87544 USA
jhorner@cybermesa.com

FCS 2014

Abstract

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of quantum-mechanical behaviors and operations. In much the same way that the structure of conventional propositional (Boolean) logic (BL) is the logic of the description of the behavior of classical physical systems and is isomorphic to a Boolean algebra (BA), so also the algebra, C(H), of closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space is a logic of the descriptions of the behavior of quantum mechanical systems and is a model of an ortholattice (OL). An OL can thus be thought of as a kind of "quantum logic" (QL). C(H) is also a model of an orthomodular lattice, which is an OL conjoined with the orthomodularity axiom (OMA). The rationalization of the OMA as a claim proper to physics has proven problematic, motivating the question of whether the OMA and its equivalents are required in an adequate characterization of QL. Here I provide an automated deduction of two quantum-union-based equivalents from orthomodularity theory. The proofs may be novel.

Keywords: automated deduction, quantum computing, orthomodular lattice, Hilbert space

1.0 Introduction

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of the description of quantum-mechanical behaviors ([1], [17], [18], [20]). In much the same way that conventional propositional (Boolean) logic (BL,[12]) is the logical structure of description of the behavior of classical physical systems (e.g. "the measurements of the position and momentum of particle P are commutative", i.e., can be measured in either other, yielding the same results) and is isomorphic to a Boolean lattice ([10], [11], [19]), so also the algebra, *C(H)*, of the closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space *H* ([1], [4], [6], [9], [13]) is *a* logic of the descriptions of the behavior of quantum mechanical systems (e.g., "the measurements of the position and momentum of particle P are *not* commutative") and is a model ([10]) of an ortholattice (OL; [4]). An OL can thus be thought of as a kind of "quantum logic" (QL; [19]). *C(H)* is also a model of (i.e., isomorphic to a set of sentences which hold in) an orthomodular lattice (OML; [4], [7]), which is an OL conjoined with the orthomodularity axiom (OMA; see Figure 1). The rationalization of the OMA as a claim proper to physics has proven problematic ([13], Section 5-6), motivating the question of whether the OMA is required in an adequate characterization of QL. Thus formulated, the question suggests that the OMA and its equivalents are specific to an OML, and that as a consequence, banning the OMA from QL yields a "truer" quantum logic.

Lattice axioms

```
      x = c(c(x))                       (AxLat1)
      x v y = y v x                     (AxLat2)
      (x v y) v z = x v (y v z)          (AxLat3)
      (x ^ y) ^ z = x ^ (y ^ z)          (AxLat4)
      x v (x ^ y) = x                   (AxLat5)
      x ^ (x v y) = x                   (AxLat6)
```

Ortholattice axioms

```
      c(x) ^ x = 0                      (AxOL1)
      c(x) v x = 1                      (AxOL2)
      x ^ y = c(c(x) v c(y))            (AxOL3)
```

Orthomodularity axiom

```
      y v (c(y) ^ (x v y)) = x v y      (OMA)
```

Definitions of implications and partial order

```
   i1(x,y) = c(x) v (x ^ y).
   i2(x,y) = i1(c(y), c(x).
   i3(x,y) = (c(x) ^ y) v (c(x) ^ c(y)) v i1(x,y).
   i4(x,y) = i3(c(y), c(x)).
   i5(x,y) = (x ^ y) v (c(x) ^ y) v (c(x) ^ c(y)).
   le(x,y) = (x = (x ^ y)).
```

Definitions of indexed unions

```
   u1(x,y) =  i1(c(x),y)
   u2(x,y) =  i2(c(x),y)
   u3(x,y) =  i3(c(x),y)
   u4(x,y) =  i4(c(x),y)
   u5(x,y) =  i5(c(x),y)
```

where
   x, y are variables ranging over lattice nodes
   ^ is lattice meet
   v is lattice join
   c(x) is the orthocomplement of x
   i1(x,y) means $x \rightarrow_1 y$ (Sasaki implication)
   i2(x,y) means $x \rightarrow_2 y$ (Dishkant implication)
   i3(x,y) means $x \rightarrow_3 y$ (Kalmbach implication)
   i4(x,y) means $x \rightarrow_4 y$ (non-tollens implication)
   i5(x,y) means $x \rightarrow_5 y$ (relevance implication)
   le(x,y) means $x \leq y$
   <-> means if and only if
   = is equivalence ([12])
   1 is the maximum lattice element (= x v c(x))
   0 is the minimum lattice element (= c(1))

Figure 1. Lattice, ortholattice, orthomodularity axioms, and some definitions.

There are at least 21 nominal equivalents (in the sense that ortholattice theory, together with these "equivalents", imply the OMA, and vice versa) of the OMA in quantum logic ([5], Theorem 2.5); as nominal equivalents, they are thus of import to optimizing quantum circuit design. Among these is the Proposition shown in Figure 2:

$$x \cup_i y \quad <-> \quad c(x) \perp c(y)$$

where

$x \cup_i y$ means $c(x) \rightarrow_i y$
$x \perp y$ means $le(x,c(y))$
$i = 1,2,3,4,5$

Figure 2. Proposition 2.11 of [5]

2.0 Method

The OML axiomatizations of Megill, Pavičić, and Horner ([5], [14], [15], [16], [21], [22]) were implemented in a *prover9* ([2]) script ([3]) configured to derive Proposition 2.11 of [5], for each of i = 3,4,5 from ortholattice theory, then executed in that framework on a Dell Inspiron 545 with an Intel Core2 Quad CPU Q8200 (clocked @ 2.33 GHz) and 8.00 GB RAM, running under the *Windows Vista Home Premium /Cygwin* operating environment.

3.0 Results

Figure 3 shows the proofs, generated by [3] on the platform described in Section 2.0, that Proposition 2.11 of [5] (for each of i = 3,4,5) is derivable from orthomodular lattice theory.

```
============================== PROOF =================================

% Proof 1 at 387.65 (+ 7.08) seconds.
% Length of proof is 63.
% Level of proof is 11.

1 le(x,y) <-> x = x ^ y # label("Df: less than") # label(non_clause).  [assumption].
2 perp(x,y) <-> le(x,c(y)) # label("Df: perpendicular") # label(non_clause).
[assumption].
4 u3(x,y) = 1 <-> perp(c(x),c(y)) # label("Proposition 2.10u3") # label(non_clause) #
label(goal).  [goal].
7 x = c(c(x)) # label("AxL1").  [assumption].
8 c(c(x)) = x.  [copy(7),flip(a)].
9 x v y = y v x # label("AxL2").  [assumption].
10 (x v y) v z = x v (y v z) # label("AxL3").  [assumption].
12 x v (x ^ y) = x # label("AxL5").  [assumption].
13 x ^ (x v y) = x # label("AxL6").  [assumption].
14 c(x) ^ x = 0 # label("AxOL1").  [assumption].
15 c(x) v x = 1 # label("AxOL2").  [assumption].
16 x v c(x) = 1.  [copy(15),rewrite([9(2)])].
17 x ^ y = c(c(x) v c(y)) # label("AxOL3").  [assumption].
18 x v (c(x) ^ (y v x)) = y v x # label("OMA").  [assumption].
19 x v c(x v c(y v x)) = y v x.  [copy(18),rewrite([17(3),8(2)])].
25 i3(x,y) = ((c(x) ^ y) v (c(x) ^ c(y))) v (c(x) v (x ^ y)) # label("Df: i3").
[assumption].
26 i3(x,y) = c(x v y) v (c(x v c(y)) v (c(x) v c(c(x) v c(y)))).
[copy(25),rewrite([17(3),8(3),17(7),8(6),8(6),9(7),17(9),10(14)])].
29 i5(x,y) = ((x ^ y) v (c(x) ^ y)) v (c(x) ^ c(y)) # label("Df: i5").  [assumption].
30 i5(x,y) = c(x v y) v (c(x v c(y)) v c(c(x) v c(y))).
[copy(29),rewrite([17(2),17(7),8(7),9(9),17(12),8(11),8(11),9(12)])].
35 u3(x,y) = i3(c(x),y) # label("Df: u3").  [assumption].
36 u3(x,y) = c(c(x) v y) v (x v (c(x v c(y)) v c(c(x) v c(y)))).
[copy(35),rewrite([26(3),8(10),8(10),9(13),10(13)])].
41 -le(x,y) | x ^ y = x # label("Df: less than").  [clausify(1)].
```

```
42 -le(x,y) | c(c(x) v c(y)) = x.  [copy(41),rewrite([17(2)])].
43 le(x,y) | x ^ y != x # label("Df: less than").  [clausify(1)].
44 le(x,y) | c(c(x) v c(y)) != x.  [copy(43),rewrite([17(2)])].
45 -perp(x,y) | le(x,c(y)) # label("Df: perpendicular").  [clausify(2)].
46 perp(x,y) | -le(x,c(y)) # label("Df: perpendicular").  [clausify(2)].
47 u3(c1,c2) = 1 | perp(c(c1),c(c2)) # label("Proposition 2.10u3").  [deny(4)].
48 c(c2 v c(c1)) v (c1 v (c(c1 v c(c2)) v c(c(c1) v c(c2)))) = 1 | perp(c(c1),c(c2)).
[copy(47),rewrite([36(3),9(4)])].
49 u3(c1,c2) != 1 | -perp(c(c1),c(c2)) # label("Proposition 2.10u3").  [deny(4)].
50 c(c2 v c(c1)) v (c1 v (c(c1 v c(c2)) v c(c(c1) v c(c2)))) != 1 | -perp(c(c1),c(c2)).
[copy(49),rewrite([36(3),9(4)])].
51 c(1) = 0.  [back_rewrite(14),rewrite([17(2),8(2),16(2)])].
52 c(c(x) v c(x v y)) = x.  [back_rewrite(13),rewrite([17(2)])].
53 x v c(c(x) v c(y)) = x.  [back_rewrite(12),rewrite([17(1)])].
55 x v (y v z) = y v (x v z).  [para(9(a,1),10(a,1,1)),rewrite([10(2)])].
56 c1 v (c(c2 v c(c1)) v (c(c1 v c(c2)) v c(c(c1) v c(c2)))) != 1 | -perp(c(c1),c(c2)).
[back_rewrite(50),rewrite([55(20)])].
57 c1 v (c(c2 v c(c1)) v (c(c1 v c(c2)) v c(c(c1) v c(c2)))) = 1 | perp(c(c1),c(c2)).
[back_rewrite(48),rewrite([55(20)])].
60 x v (c(x) v y) = 1 v y.  [para(16(a,1),10(a,1,1)),flip(a)].
79 le(x,x v y).  [resolve(52,a,44,b)].
80 c(x) v c(x v y) = c(x).  [para(52(a,1),8(a,1,1)),flip(a)].
84 c(0 v c(x)) = x.  [para(16(a,1),52(a,1,1,2,1)),rewrite([51(3),9(3)])].
85 c(x v y) v c(x v c(x v y)) = c(x).
[para(52(a,1),19(a,1,2,1,2)),rewrite([9(5),80(11)])].
86 1 v x = 1.  [para(51(a,1),52(a,1,1,1)),rewrite([84(6)])].
90 x v (c(x) v y) = 1.  [back_rewrite(60),rewrite([86(5)])].
91 x v c(c(x) v y) = x.  [para(8(a,1),53(a,1,2,1,2))].
95 x v 0 = x.  [para(16(a,1),53(a,1,2,1)),rewrite([51(2)])].
96 x v c(y v c(x)) = x.  [para(19(a,1),53(a,1,2,1))].
97 x v x = x.  [para(51(a,1),53(a,1,2,1,2)),rewrite([9(3),84(4)])].
105 c1 v (c(c2 v c(c1)) v (c(c1 v c(c2)) v c(c(c1) v c(c2)))) = 1 | le(c(c1),c2).
[resolve(57,b,45,a),rewrite([8(27)])].
106 le(x,y v x).  [para(9(a,1),79(a,2))].
111 0 v x = x.  [para(95(a,1),9(a,1)),flip(a)].
115 x v (x v y) = x v y.  [para(97(a,1),10(a,1,1)),flip(a)].
201 x v (y v c(c(x) v z)) = y v x.  [para(91(a,1),55(a,1,2)),flip(a)].
203 le(c(c(x) v y),x).  [para(91(a,1),106(a,2))].
211 perp(c(x v y),x).  [resolve(203,a,46,b),rewrite([8(2)])].
215 perp(c(x v y),y).  [para(9(a,1),211(a,1,1))].
274 x v (c(y v c(x)) v z) = x v z.  [para(96(a,1),10(a,1,1)),flip(a)].
286 c1 v c(c1 v c(c2)) = 1 | le(c(c1),c2).
[back_rewrite(105),rewrite([274(20),201(14),9(7)])].
288 c1 v c(c1 v c(c2)) != 1 | -perp(c(c1),c(c2)).
[back_rewrite(56),rewrite([274(20),201(14),9(7)])].
13482 c1 v c(c1 v c(c2)) = 1 | c(c1 v c(c2)) = c(c1).
[resolve(286,b,42,a),rewrite([8(12)])].
151209 c(c1 v c(c2)) = c(c1).
[para(13482(a,1),30(a,2,1,1)),rewrite([30(15),8(23),115(22),8(29),55(28),90(28),51(23),9(
23),111(23),9(22),85(22),51(12),8(18),115(17),8(24),55(23),90(23),51(18),9(18),111(18),11
1(17)]),flip(b),merge(b)].
151210 -perp(c(c1),c(c2)).  [back_rewrite(288),rewrite([151209(6),16(4)]),xx(a)].
151215 $F.  [para(151209(a,1),215(a,1)),unit_del(a,151210)].

============================== end of proof ==========================

============================== PROOF =================================

% Proof 1 at 420.06 (+ 7.00) seconds.
% Length of proof is 56.
% Level of proof is 11.

1 le(x,y) <-> x = x ^ y # label("Df: less than") # label(non_clause).  [assumption].
2 perp(x,y) <-> le(x,c(y)) # label("Df: perpendicular") # label(non_clause).
[assumption].
4 u4(x,y) = 1 <-> perp(c(x),c(y)) # label("Proposition 2.10u4") # label(non_clause) #
label(goal).  [goal].
7 x = c(c(x)) # label("AxL1").  [assumption].
8 c(c(x)) = x.  [copy(7),flip(a)].
9 x v y = y v x # label("AxL2").  [assumption].
10 (x v y) v z = x v (y v z) # label("AxL3").  [assumption].
```

```
12 x v (x ^ y) = x # label("AxL5").  [assumption].
13 x ^ (x v y) = x # label("AxL6").  [assumption].
14 c(x) ^ x = 0 # label("AxOL1").  [assumption].
15 c(x) v x = 1 # label("AxOL2").  [assumption].
16 x v c(x) = 1.  [copy(15),rewrite([9(2)])].
17 x ^ y = c(c(x) v c(y)) # label("AxOL3").  [assumption].
18 x v (c(x) ^ (y v x)) = y v x # label("OMA").  [assumption].
19 x v c(x v c(y v x)) = y v x.  [copy(18),rewrite([17(3),8(2)])].
27 i4(x,y) = ((c(c(y)) ^ c(x)) v (c(c(y)) ^ c(c(x)))) v (c(c(y)) v (c(y) ^ c(x))) #
label("Df: i4").  [assumption].
28 i4(x,y) = y v (c(y v x) v (c(c(y) v x) v c(c(y) v c(x)))).
[copy(27),rewrite([8(3),17(3),8(4),8(6),8(6),17(5),8(11),17(12),8(11),8(11),9(13),10(13)]
)].
37 u4(x,y) = i4(c(x),y) # label("Df: u4").  [assumption].
38 u4(x,y) = y v (c(y v c(x)) v (c(c(y) v x) v c(c(y) v c(x)))).
[copy(37),rewrite([28(3),8(11),9(12)])].
41 -le(x,y) | x ^ y = x # label("Df: less than").  [clausify(1)].
42 -le(x,y) | c(c(x) v c(y)) = x.  [copy(41),rewrite([17(2)])].
43 le(x,y) | x ^ y != x # label("Df: less than").  [clausify(1)].
44 le(x,y) | c(c(x) v c(y)) != x.  [copy(43),rewrite([17(2)])].
45 -perp(x,y) | le(x,c(y)) # label("Df: perpendicular").  [clausify(2)].
46 perp(x,y) | -le(x,c(y)) # label("Df: perpendicular").  [clausify(2)].
47 u4(c1,c2) = 1 | perp(c(c1),c(c2)) # label("Proposition 2.10u4").  [deny(4)].
48 c2 v (c(c2 v c(c1)) v (c(c1 v c(c2)) v c(c(c1) v c(c2)))) = 1 | perp(c(c1),c(c2)).
[copy(47),rewrite([38(3),9(10),9(16)])].
49 u4(c1,c2) != 1 | -perp(c(c1),c(c2)) # label("Proposition 2.10u4").  [deny(4)].
50 c2 v (c(c2 v c(c1)) v (c(c1 v c(c2)) v c(c(c1) v c(c2)))) != 1 | -perp(c(c1),c(c2)).
[copy(49),rewrite([38(3),9(10),9(16)])].
51 c(1) = 0.  [back_rewrite(14),rewrite([17(2),8(2),16(2)])].
52 c(c(x) v c(x v y)) = x.  [back_rewrite(13),rewrite([17(2)])].
53 x v c(c(x) v c(y)) = x.  [back_rewrite(12),rewrite([17(1)])].
55 x v (y v z) = y v (x v z).  [para(9(a,1),10(a,1,1)),rewrite([10(2)])].
72 c2 v (c(c2 v c(c1)) v (c(c1 v c(c2)) v c(c(c1) v c(c2)))) = 1 | le(c(c1),c2).
[resolve(48,b,45,a),rewrite([8(27)])].
78 le(x,x v y).  [resolve(52,a,44,b)].
79 c(x) v c(x v y) = c(x).  [para(52(a,1),8(a,1,1)),flip(a)].
84 c(x v y) v c(x v c(x v y)) = c(x).
[para(52(a,1),19(a,1,2,1,2)),rewrite([9(5),79(11)])].
90 x v c(c(x) v y) = x.  [para(8(a,1),53(a,1,2,1,2))].
94 x v 0 = x.  [para(16(a,1),53(a,1,2,1)),rewrite([51(2)])].
95 x v c(y v c(x)) = x.  [para(19(a,1),53(a,1,2,1))].
104 le(x,y v x).  [para(9(a,1),78(a,2))].
109 0 v x = x.  [para(94(a,1),9(a,1)),flip(a)].
219 le(c(c(x) v y),x).  [para(90(a,1),104(a,2))].
229 perp(c(x v y),x).  [resolve(219,a,46,b),rewrite([8(2)])].
234 perp(c(x v y),y).  [para(9(a,1),229(a,1,1))].
290 x v (c(y v c(x)) v z) = x v z.  [para(95(a,1),10(a,1,1)),flip(a)].
371 c2 v (c(c2 v c(c1)) v (c(c1 v c(c2)) v c(c(c1) v c(c2)))) = 1 | c(c1 v c(c2)) =
c(c1).  [resolve(72,b,42,a),rewrite([8(25)])].
12902 x v (y v (c(z v c(x)) v u)) = y v (x v u).  [para(290(a,1),55(a,1,2)),flip(a)].
12922 x v c(y v c(y v c(x))) = x v c(y).  [para(84(a,1),290(a,1,2)),flip(a)].
13162 c2 v c(c2 v c(c1)) = 1 | c(c1 v c(c2)) = c(c1).
[back_rewrite(371),rewrite([12902(20),95(13),9(7)])].
13164 c2 v c(c2 v c(c1)) != 1 | -perp(c(c1),c(c2)).
[back_rewrite(50),rewrite([12902(20),95(13),9(7)])].
158756 c2 v c(c2 v c(c1)) = 1 | c1 v c(c2) = c1.
[para(13162(b,1),8(a,1,1)),rewrite([8(12)]),flip(b)].
160444 c1 v c(c2) = c1.
[para(158756(a,1),12922(a,1,2,1)),rewrite([51(9),9(9),109(9)]),flip(b),merge(b)].
160455 perp(c(c1),c(c2)).  [para(160444(a,1),234(a,1,1))].
160458 c2 v c(c1) = c2.  [para(160444(a,1),95(a,1,2,1))].
161989 $F.  [back_unit_del(13164),rewrite([160458(5),16(4)]),xx(a),unit_del(a,160455)].

============================== end of proof ==========================
```

Figure 3. Summary of a *prover9* ([2]) derivation of Proposition 2.11, for each of i = 3,4,5, from orthomodular lattice theory. The proofs assume the default inference rules of *prover9*. The general

form of a line in this proof is "*line_number conclusion [derivation]*", where *line_number* is a unique identifier of a line in the proof, and *conclusion* is the result of applying the *prover9* inference rules (such as *para*modulation, *copy*ing, and *rewrit*ing), noted in square brackets (denoting the *derivation*), to the lines cited in those brackets. Note that some of "logical" proof lines in the above have been transformed to two text lines, with the *derivation* appearing on a text line following a text line containing the first part of that logical line. The detailed syntax and semantics of these notations can be found in [2]. All *prover9* proofs are by default proofs by contradiction.

The total time to produce the proofs in Figure 3 on the platform described in Section 2.0 was approximately 800 seconds.

4.0 Discussion

The results of Section 3.0 motivate several observations:

1. Each of the proofs in Figure 3 uses L1, L2, L3, L5, L6, OL1, OL2, and OL3.

2. The proofs in Section 3.0 may be novel.

3. Companion papers provide proofs for i = 1,2,5 and for Propositions 2.11i, i = 1,2,3,4,5 implies the OMA.

4. Proposition 2.13 can be regarded as a definition of quantum union; thus, this paper together with the papers mentioned in (3), constitute a proof that the definition of quantum intersection is equivalent to the OMA in orthomodular quantum logic. . Companion papers derive equivalences for the OMA with definitions of quantum-intersection and quantum-identity. Collectively, these papers provide a theory of equivalence of the OMA with the quantum connectives. In light of these equivalences, QL without the OMA would hardly qualify as a logic.

5.0 Acknowledgements

This work benefited from discussions with Tom Oberdan, Frank Pecchioni, Tony Pawlicki, and the late John K. Prentice, whose passion for foundations of physics inspired those of us privileged to have known him. For any infelicities that remain, I am solely responsible.

6.0 References

[1] von Neumann J. *Mathematical Foundations of Quantum Mechanics*. 1936. Translated by R. T. Beyer. Princeton. 1983.
[2] McCune WW. *prover9 and mace4.* URL http://www.cs.unm.edu/~mccune/prover9/. 2009.
[3] Horner JK. MHT *prover9* scripts. 2011. Available from the author on request.
[4] Dalla Chiara ML and Giuntini R. *Quantum Logics*. URL http://xxx.lanl.gov/abs/quant-ph/0101028. 2004.
[5] Megill ND and Pavičić M. Orthomodular lattices and quantum algebra. *International Journal of Theoretical Physics* 40 (2001), pp. 1387-1410.
[6] Akhiezer NI and Glazman IM. *Theory of Linear Operators in Hilbert Space. Volume I.* Translated by M. Nestell. Frederick Ungar. 1961.
[7] Holland, Jr. SS Orthomodularity in infinite dimensions: a theorem of M. Solèr. *Bulletin of the American Mathematical Society* 32 (1995), pp. 205-234.
[8] Marsden EL and Herman LM. *A condition for distribution in orthomodular lattices.* Kansas State University Technical Report #40. 1974.
[9] Knuth DE and Bendix PB. Simple word problems in universal algebras. In J. Leech, ed. *Computational Problems in Abstract*

Algebra. Pergamon Press. 1970. pp. 263-297.
[10] Chang CC and Keisler HJ. *Model Theory*. North-Holland. 1990. pp. 38-39.
[11] Birkhoff G. *Lattice Theory*. Third Edition. American Mathematical Society. 1967.
[12] Church A. *Introduction to Mathematical Logic. Volume I.* Princeton. 1956.
[13] Jauch J. *Foundations of Quantum Mechanics.* Addison-Wesley. 1968.
[14] Megill ND. *Metamath.* URL http://us.metamath.org/qlegif/mmql.html#unify. 2004.
[15] Horner JK. An automated deduction system for orthomodular lattice theory. *Proceedings of the 2005 International Conference on Artificial Intelligence.* CSREA Press. 2005. pp. 260-265.
[16] Horner JK. An automated equational logic deduction of join elimination in orthomodular lattice theory. *Proceedings of the 2007 International Conference on Artificial Intelligence.* CSREA Press. 2007. pp. 481-488.
[17] Messiah A. *Quantum Mechanics*. Dover. 1958.
[18] Horner JK. Using automated theorem-provers to aid the design of efficient compilers for quantum computing. *Los Alamos National Laboratory Quantum Institute Workshop.* December 9–10, 2002. URL http://www.lanl.gov/science/centers/quantum/qls_pdfs/horner.pdf.
[19] Birkhoff G and von Neumann J. The logic of quantum mechanics. *Annals of Mathematics* 37 (1936), 823-243.
[20] Nielsen MA and Chuang L . *Quantum Computation and Quantum Information*. Cambridge. 2000.
[21] Pavičić M and Megill N. Quantum and classical implicational algebras with primitive implication. *International Journal of Theoretical Physics* 37 (1998), 2091-2098. ftp://m3k.grad.hr/pavicic/quantum-logic/1998-int-j-theor-phys-2.ps.gz.
[22] Horner JK. An automated deduction of the relative strength of orthomodular and weakly orthomodular lattice theory. *Proceedings of the 2009 International Conference on Artificial Intelligence*. CSREA Press. 2009. pp. 525-530.

Quantum-Union Equivalents of the Orthomodularity Law in Quantum Logic: Part 5

Jack K. Horner
P. O. Box 266
Los Alamos, New Mexico 87544 USA
jhorner@cybermesa.com

FCS 2014

Abstract

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of quantum-mechanical behaviors and operations. In much the same way that the structure of conventional propositional (Boolean) logic (BL) is the logic of the description of the behavior of classical physical systems and is isomorphic to a Boolean algebra (BA), so also the algebra, C(H), of closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space is a logic of the descriptions of the behavior of quantum mechanical systems and is a model of an ortholattice (OL). An OL can thus be thought of as a kind of "quantum logic" (QL). C(H) is also a model of an orthomodular lattice, which is an OL conjoined with the orthomodularity axiom (OMA). The rationalization of the OMA as a claim proper to physics has proven problematic, motivating the question of whether the OMA and its equivalents are required in an adequate characterization of QL. Here I provide an automated deduction of a quantum-union-based equivalent of the OMA from orthomodularity theory. The proof may be novel.

Keywords: automated deduction, quantum computing, orthomodular lattice, Hilbert space

1.0 Introduction

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of the description of quantum-mechanical behaviors ([1], [17], [18], [20]). In much the same way that conventional propositional (Boolean) logic (BL,[12]) is the logical structure of description of the behavior of classical physical systems (e.g. "the measurements of the position and momentum of particle P are commutative", i.e., can be measured in either other, yielding the same results) and is isomorphic to a Boolean lattice ([10], [11], [19]), so also the algebra, *C(H)*, of the closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space *H* ([1], [4], [6], [9], [13]) is *a* logic of the descriptions of the behavior of quantum mechanical systems (e.g., "the measurements of the position and momentum of particle P are *not* commutative") and is a model ([10]) of an ortholattice (OL; [4]). An OL can thus be thought of as a kind of "quantum logic" (QL; [19]). *C(H)* is also a model of (i.e., isomorphic to a set of sentences which hold in) an orthomodular lattice (OML; [4], [7]), which is an OL conjoined with the orthomodularity axiom (OMA; see Figure 1). The rationalization of the OMA as a claim proper to physics has proven problematic ([13], Section 5-6), motivating the question of whether the OMA is required in an adequate characterization of QL. Thus formulated, the question suggests that the OMA and its equivalents are specific to an OML, and that as a consequence, banning the OMA from QL yields a "truer" quantum logic.

Lattice axioms

```
      x = c(c(x))                       (AxLat1)
      x v y = y v x                     (AxLat2)
      (x v y) v z = x v (y v z)          (AxLat3)
      (x ^ y) ^ z = x ^ (y ^ z)          (AxLat4)
      x v (x ^ y) = x                   (AxLat5)
      x ^ (x v y) = x                   (AxLat6)
```

Ortholattice axioms

```
      c(x) ^ x = 0                      (AxOL1)
      c(x) v x = 1                      (AxOL2)
      x ^ y = c(c(x) v c(y))            (AxOL3)
```

Orthomodularity axiom

```
      y v (c(y) ^ (x v y)) = x v y      (OMA)
```

Definitions of implications and partial order

```
   i1(x,y) = c(x) v (x ^ y).
   i2(x,y) = i1(c(y), c(x).
   i3(x,y) = (c(x) ^ y) v (c(x) ^ c(y)) v i1(x,y).
   i4(x,y) = i3(c(y), c(x)).
   i5(x,y) = (x ^ y) v (c(x) ^ y) v (c(x) ^ c(y)).
   le(x,y) = (x = (x ^ y)).
```

Definitions of indexed unions

```
   u1(x,y) =  i1(c(x),y)
   u2(x,y) =  i2(c(x),y)
   u3(x,y) =  i3(c(x),y)
   u4(x,y) =  i4(c(x),y)
   u5(x,y) =  i5(c(x),y)
```

```
where
   x, y are variables ranging over lattice nodes
   ^ is lattice meet
   v is lattice join
   c(x) is the orthocomplement of x
   i1(x,y) means x →1 y (Sasaki implication)
   i2(x,y) means x →2 y (Dishkant implication)
   i3(x,y) means x →3 y (Kalmbach implication)
   i4(x,y) means x →4 y (non-tollens implication)
   i5(x,y) means x →5 y (relevance implication)
   le(x,y) means x ≤ y
   <-> means if and only if
   =  is equivalence ([12])
   1 is the maximum lattice element (= x v c(x))
   0 is the minimum lattice element (= c(1))
```

Figure 1. Lattice, ortholattice, orthomodularity axioms, and some definitions.

There are at least 21 nominal equivalents (in the sense that ortholattice theory, together with these "equivalents", imply the OMA, and vice versa) of the OMA in quantum logic ([5], Theorem 2.5); as nominal equivalents, they are thus of import to optimizing quantum circuit design. Among these is the Proposition shown in Figure 2:

$$x \cup_i y \quad <-> \quad c(x) \perp c(y)$$

where

```
x ∪ᵢ y means c(x) →ᵢ y
x ⊥ y means le(x,c(y))
i = 1,2,3,4,5
```

Figure 2. Proposition 2.11 of [5]

2.0 Method

The OML axiomatizations of Megill, Pavičić, and Horner ([5], [14], [15], [16], [21], [22]) were implemented in a *prover9* ([2]) script ([3]) configured to derive Proposition 2.11 of [5], for i = 5 from ortholattice theory, then executed in that framework on a Dell Inspiron 545 with an Intel Core2 Quad CPU Q8200 (clocked @ 2.33 GHz) and 8.00 GB RAM, running under the *Windows Vista Home Premium /Cygwin* operating environment.

3.0 Results

Figure 3 shows the proofs, generated by [3] on the platform described in Section 2.0, that Proposition 2.11 of [5] (for i = 5) is derivable from orthomodular lattice theory.

```
============================== PROOF

% Proof 1 at 561.59 (+ 10.45) seconds.
% Length of proof is 62.
% Level of proof is 11.

1 le(x,y) <-> x = x ^ y # label("Df: less than") #
label(non_clause).  [assumption].
2 perp(x,y) <-> le(x,c(y)) # label("Df: perpendicular") #
label(non_clause).  [assumption].
4 u5(x,y) = 1 <-> perp(c(x),c(y)) # label("Proposition 2.10u5") #
label(non_clause) # label(goal).  [goal].
7 x = c(c(x)) # label("AxL1").  [assumption].
8 c(c(x)) = x.  [copy(7),flip(a)].
9 x v y = y v x # label("AxL2").  [assumption].
10 (x v y) v z = x v (y v z) # label("AxL3").  [assumption].
12 x v (x ^ y) = x # label("AxL5").  [assumption].
13 x ^ (x v y) = x # label("AxL6").  [assumption].
14 c(x) ^ x = 0 # label("AxOL1").  [assumption].
15 c(x) v x = 1 # label("AxOL2").  [assumption].
16 x v c(x) = 1.  [copy(15),rewrite([9(2)])].
17 x ^ y = c(c(x) v c(y)) # label("AxOL3").  [assumption].
18 x v (c(x) ^ (y v x)) = y v x # label("OMA").  [assumption].
19 x v c(x v c(y v x)) = y v x.  [copy(18),rewrite([17(3),8(2)])].
29 i5(x,y) = ((x ^ y) v (c(x) ^ y)) v (c(x) ^ c(y)) # label("Df:
i5").  [assumption].
```

```
30 i5(x,y) = c(x v y) v (c(x v c(y)) v c(c(x) v c(y))).
[copy(29),rewrite([17(2),17(7),8(7),9(9),17(12),8(11),8(11),9(12)])]
.
39 u5(x,y) = i5(c(x),y) # label("Df: u5").  [assumption].
40 u5(x,y) = c(c(x) v y) v (c(x v c(y)) v c(c(x) v c(y))).
[copy(39),rewrite([30(3),8(10),9(12)])].
41 -le(x,y) | x ^ y = x # label("Df: less than").  [clausify(1)].
42 -le(x,y) | c(c(x) v c(y)) = x.  [copy(41),rewrite([17(2)])].
43 le(x,y) | x ^ y != x # label("Df: less than").  [clausify(1)].
44 le(x,y) | c(c(x) v c(y)) != x.  [copy(43),rewrite([17(2)])].
45 -perp(x,y) | le(x,c(y)) # label("Df: perpendicular").
[clausify(2)].
46 perp(x,y) | -le(x,c(y)) # label("Df: perpendicular").
[clausify(2)].
47 u5(c1,c2) = 1 | perp(c(c1),c(c2)) # label("Proposition 2.10u5").
[deny(4)].
48 c(c2 v c(c1)) v (c(c1 v c(c2)) v c(c(c1) v c(c2))) = 1 |
perp(c(c1),c(c2)).  [copy(47),rewrite([40(3),9(4)])].
49 u5(c1,c2) != 1 | -perp(c(c1),c(c2)) # label("Proposition
2.10u5").  [deny(4)].
50 c(c2 v c(c1)) v (c(c1 v c(c2)) v c(c(c1) v c(c2))) != 1 | -
perp(c(c1),c(c2)).  [copy(49),rewrite([40(3),9(4)])].
51 c(1) = 0.  [back_rewrite(14),rewrite([17(2),8(2),16(2)])].
52 c(c(x) v c(x v y)) = x.  [back_rewrite(13),rewrite([17(2)])].
53 x v c(c(x) v c(y)) = x.  [back_rewrite(12),rewrite([17(1)])].
55 x v (y v z) = y v (x v z).
[para(9(a,1),10(a,1,1)),rewrite([10(2)])].
58 x v (c(x) v y) = 1 v y.  [para(16(a,1),10(a,1,1)),flip(a)].
72 c(c2 v c(c1)) v (c(c1 v c(c2)) v c(c(c1) v c(c2))) = 1 |
le(c(c1),c2).  [resolve(48,b,45,a),rewrite([8(25)])].
78 le(x,x v y).  [resolve(52,a,44,b)].
79 c(x) v c(x v y) = c(x).  [para(52(a,1),8(a,1,1)),flip(a)].
83 c(0 v c(x)) = x.
[para(16(a,1),52(a,1,1,2,1)),rewrite([51(3),9(3)])].
84 c(x v y) v c(x v c(x v y)) = c(x).
[para(52(a,1),19(a,1,2,1,2)),rewrite([9(5),79(11)])].
85 1 v x = 1.  [para(51(a,1),52(a,1,1,1)),rewrite([83(6)])].
89 x v (c(x) v y) = 1.  [back_rewrite(58),rewrite([85(5)])].
90 x v c(c(x) v y) = x.  [para(8(a,1),53(a,1,2,1,2))].
94 x v 0 = x.  [para(16(a,1),53(a,1,2,1)),rewrite([51(2)])].
95 x v c(y v c(x)) = x.  [para(19(a,1),53(a,1,2,1))].
96 x v x = x.  [para(51(a,1),53(a,1,2,1,2)),rewrite([9(3),83(4)])].
104 le(x,y v x).  [para(9(a,1),78(a,2))].
109 0 v x = x.  [para(94(a,1),9(a,1)),flip(a)].
113 x v (x v y) = x v y.  [para(96(a,1),10(a,1,1)),flip(a)].
217 x v (y v c(c(x) v z)) = y v x.
[para(90(a,1),55(a,1,2)),flip(a)].
219 le(c(c(x) v y),x).  [para(90(a,1),104(a,2))].
229 perp(c(x v y),x).  [resolve(219,a,46,b),rewrite([8(2)])].
234 perp(c(x v y),y).  [para(9(a,1),229(a,1,1))].
290 x v (c(y v c(x)) v z) = x v z.
[para(95(a,1),10(a,1,1)),flip(a)].
371 c(c2 v c(c1)) v (c(c1 v c(c2)) v c(c(c1) v c(c2))) = 1 | c(c1 v
c(c2)) = c(c1).  [resolve(72,b,42,a),rewrite([8(23)])].
```

```
18974 c(c1 v c(c2)) = c(c1) | c1 v c(c1 v c(c2)) = 1.
[para(371(a,1),290(a,1,2)),rewrite([9(11),85(11),217(23),9(16)]),fli
p(b)].
176457 c(c1 v c(c2)) = c(c1).
[para(18974(b,1),30(a,2,1,1)),rewrite([30(15),8(23),113(22),8(29),55
(28),89(28),51(23),9(23),109(23),9(22),84(22),51(12),8(18),113(17),8
(24),55(23),89(23),51(18),9(18),109(18),109(17)]),flip(b),merge(b)].
176458 c(c1) v (c(c2 v c(c1)) v c(c(c1) v c(c2))) != 1 | -
perp(c(c1),c(c2)).  [back_rewrite(50),rewrite([176457(10),55(15)])].
176460 c1 v c(c2) = c1.
[para(176457(a,1),8(a,1,1)),rewrite([8(3)]),flip(a)].
176461 c(c2) v c(c(c1) v c(c2)) = c1.
[para(176457(a,1),19(a,1,2,1,2)),rewrite([9(7),176460(13)])].
176464 perp(c(c1),c(c2)).  [para(176457(a,1),234(a,1))].
176465 c2 v c(c1) = c2.  [para(176457(a,1),95(a,1,2))].
177121 $F.
[back_unit_del(176458),rewrite([176465(6),176461(11),9(4),16(4)]),xx
(a),unit_del(a,176464)].

================== end of proof
```

Figure 3. Summary of a *prover9* ([2]) derivation of Proposition 2.11, for i = 5 from orthomodular lattice theory. The proofs assume the default inference rules of *prover9*. The general form of a line in this proof is "*line_number conclusion [derivation]*", where *line_number* is a unique identifier of a line in the proof, and *conclusion* is the result of applying the *prover9* inference rules (such as *para*modulation, *copy*ing, and *rewrit*ing), noted in square brackets (denoting the *derivation*), to the lines cited in those brackets. Note that some of "logical" proof lines in the above have been transformed to two text lines, with the *derivation* appearing on a text line following a text line containing the first part of that logical line. The detailed syntax and semantics of these notations can be found in [2]. All *prover9* proofs are by default proofs by contradiction.

The total time to produce the proofs in Figure 3 on the platform described in Section 2.0 was approximately 560 seconds.

4.0 Discussion

The results of Section 3.0 motivate several observations:

1. The poof in Figure 3 uses L1, L2, L3, L5, L6, OL1, OL2, and OL3.

2. The proof in Section 3.0 may be novel.

3. Companion papers provide proofs for i = 1,2,5 and for Propositions 2.11i of [5], i = 1,2,3,4,5 implies the OMA.

4. Proposition 2.13 can be regarded as a definition of quantum union; thus, this paper together with the papers mentioned in (3), constitute a proof that the definition of quantum intersection is equivalent to the OMA in orthomodular quantum logic. .Companion papers derive equivalences for the OMA with definitions of quantum-intersection and quantum-identity. Collectively, these papers provide a theory of equivalence of the OMA with the quantum connectives. In light of these equivalences, QL without the OMA would hardly qualify as a logic.

5.0 Acknowledgements

This work benefited from discussions with Tom Oberdan, Frank Pecchioni, Tony Pawlicki, and the late John K. Prentice, whose passion for foundations of physics inspired those of us privileged to have known him. For any infelicities that remain, I am solely responsible.

6.0 References

[1] von Neumann J. *Mathematical Foundations of Quantum Mechanics*. 1936. Translated by R. T. Beyer. Princeton. 1983.

[2] McCune WW. *prover9 and mace4.* URL http://www.cs.unm.edu/~mccune/prover9/. 2009.

[3] Horner JK. MHT *prover9* scripts. 2011. Available from the author on request.

[4] Dalla Chiara ML and Giuntini R. *Quantum Logics*. URL http://xxx.lanl.gov/abs/quant-ph/0101028. 2004.

[5] Megill ND and Pavičić M. Orthomodular lattices and quantum algebra. *International Journal of Theoretical Physics* 40 (2001), pp. 1387-1410.

[6] Akhiezer NI and Glazman IM. *Theory of Linear Operators in Hilbert Space. Volume I.* Translated by M. Nestell. Frederick Ungar. 1961.

[7] Holland, Jr. SS Orthomodularity in infinite dimensions: a theorem of M. Solèr. *Bulletin of the American Mathematical Society* 32 (1995), pp. 205-234.

[8] Marsden EL and Herman LM. *A condition for distribution in orthomodular lattices.* Kansas State University Technical Report #40. 1974.

[9] Knuth DE and Bendix PB. Simple word problems in universal algebras. In J. Leech, ed. *Computational Problems in Abstract Algebra.* Pergamon Press. 1970. pp. 263-297.

[10] Chang CC and Keisler HJ. *Model Theory*. North-Holland. 1990. pp. 38-39.

[11] Birkhoff G. *Lattice Theory*. Third Edition. American Mathematical Society. 1967.

[12] Church A. *Introduction to Mathematical Logic. Volume I.* Princeton. 1956.

[13] Jauch J. *Foundations of Quantum Mechanics.* Addison-Wesley. 1968.

[14] Megill ND. *Metamath.* URL http://us.metamath.org/qlegif/mmql.html#unify. 2004.

[15] Horner JK. An automated deduction system for orthomodular lattice theory. *Proceedings of the 2005 International Conference on Artificial Intelligence.* CSREA Press. 2005. pp. 260-265.

[16] Horner JK. An automated equational logic deduction of join elimination in orthomodular lattice theory. *Proceedings of the 2007 International Conference on Artificial Intelligence.* CSREA Press. 2007. pp. 481-488.

[17] Messiah A. *Quantum Mechanics*. Dover. 1958.

[18] Horner JK. Using automated theorem-provers to aid the design of efficient compilers for quantum computing. *Los Alamos National Laboratory Quantum Institute Workshop.* December 9–10, 2002.

URL
http://www.lanl.gov/science/centers/quantum/qls_pdfs/horner.pdf.

[19] Birkhoff G and von Neumann J. The logic of quantum mechanics. *Annals of Mathematics* 37 (1936), 823-243.

[20] Nielsen MA and Chuang L . *Quantum Computation and Quantum Information*. Cambridge. 2000.

[21] Pavičić M and Megill N. Quantum and classical implicational algebras with primitive implication. *International Journal of Theoretical Physics* 37 (1998), 2091-2098. ftp://m3k.grad.hr/pavicic/quantum-logic/1998-int-j-theor-phys-2.ps.gz.

[22] Horner JK. An automated deduction of the relative strength of orthomodular and weakly orthomodular lattice theory. *Proceedings of the 2009 International Conference on Artificial Intelligence*. CSREA Press. 2009. pp. 525-530.

SESSION

POSTERS

Chair(s)

TBA

A Study on the Public Opinion on the Issue of Cost Sharing in Higher Education in Taiwan

Hsiu-Hsi Liu
Assistant Research Fellow
Research Center for Educational System and Policy
National Academy for Educational Research

Introduction

Since the 90s, the numbers of schools and students in higher education have been surging in Taiwan. Net enrollment rate rate had exceeded 15% in the academic year of 1988 and later on exceeded 50% in the academic year of 2004. Ever since then, higher education in Taiwan entered the stage of what Trow (1974) called 'universal education.' However, the government budget was not able to increase at the same rate commensurate with the expansion of higher education. As a result, education resources were diluted and some schools even encountered financial difficulties. In order to maintain the quality of education, schools had to looked for other financial sources, one of which being tuition rise.

However, the rise in tuition caused much controversy among students, parents, and school management. Therefore, the present study aims to survey the general public in order to understand the issue of tuition rise from different perspectives, and thereby provides insights and solutions to the issue.

Literature Review

The idea of cost sharing in higher education

The idea of sharing and shifting the cost of education was proposed by Johnstone (1986), who argued that the beneficiaries of higher education include governments, individuals, parents, and donators. According to the benefits-received principle, the beneficiaries of higher education are responsible for sharing the cost of

the education based on the benefits received, under the assumption that the cost of higher education can be clearly defined.

First, as far as the government is concerned, one of the most important purposes of higher education is to create quality human resources and to provide the opportunity of upward social mobility for the lower socio-economic class, which can, in turn, facilitate democracy, economic development and social harmony.

Second, as for the individuals, due to the fact that higher education is not mandatory, college students are willing to sacrifice the time and money to receive higher education because they anticipate higher income and social status, as well as an improvement on the quality of life. Moreover, according to the OECD (2013) data, the private rate of return of higher education is in general greater than the social rate of return. Therefore, it is reasonable for individuals to share the cost of their education.

Finally, many NGOs in Taiwan have been urging the reform of the taxation system: taxing corporate organizations for their capital gain, and dedicating the money for enhancing higher education. Therefore, the present study also views corporate organizations as an entity responsible for sharing the cost of higher education. It is attributable to the fact schools create talented individuals who can generate profits for corporate organizations. Moreover, the increase of national quality can also enhance the social and economic development of a country, which is the fundamental to business growth. Therefore, corporate organizations ought to share the cost of higher education as well.

The status quo of the cost-sharing system in Taiwan

Before 1998, college tuition fees in Taiwan were set by the Ministry of Education, adjusted annually by the standard of governmental pay. Other fees were adjusted in accordance with the Consumer Price Index announced every April. Same adjustment was made for all schools. However, this system failed to reflect the actual cost of education: some outstanding schools were financially restricted whereas some underperformed schools were being overprotected. Moreover, private and public schools could not compete on an equal basis because private schools were usually not sufficiently subsidized by the government, and donation was not socially encouraged

at the time. Thus, in order to meet the trend of higher education, since the academic year of 1990, the Ministry of education ceased to stipulate college tuition rates. Accredited schools were then given the right to set their own tuition rates.

In early 1990s, with the rising economy in Taiwan and the intention to reduce the tuition difference between private and public schools, the tuition fees of public schools were raised by 52%, from NT34112 in 1995 to NT51954 in 2000. It then was raised again by 15% to NT59490 in 2005. As for private school, the tuition fees were raised by 13% from $92088 in 1995, and to $103950 in 2000. The fess were then raised again by 4% in 2005 to $108026, which lowered the ratio of public and private school tuition from 1:2.70 in 1995 to 1:1.82 in 2005 (Department of Statistics, MoE, 2012b). However, in recent years, due to the increasing poverty gap in Taiwan, both public and private schools ceased to raise the tuition since 2006 and 2008, respectively, under government's moral suasion.

Since college tuition rates were not able to be adjusted in accordance with the operation cost of schools, which may potentially affect the quality of education and the development of schools, Taiwanese government reinitiated the debate on the issue of college tuition in 2012, and proposed the College Tuition Adjustment Plan. However, due to the misunderstanding between the planners and stakeholders on the content of the policy, what the planners thought to be an excellent policy might not be thoroughly and effectively implemented. Therefore, it would be helpful for the planners to survey the general public as they are planning in order to understand the public opinion.

Research Design and Implementation

From a cost-sharing perspective, the present study was conducted through Computer-Assisted Telephone Interview (CATI), in order to understand participants' views on different entities sharing the cost of education. The research tool, framework, and participants are detailed as follows.

Research tool

The design of the questionnaire was two phases. Based on the literature, the

researchers first came up with four possible entities responsible for cost sharing. Then the researcher consulted experts to revise and finalize the questionnaire. After the questionnaire was finalized, a pilot study was carried out by a private survey company. Tested by F-test, the results of the pilot study were valid and coherent. The questionnaire was designed based on the research goals and previous literature, and was then revised based on experts' comments.

Participants

Participants of this study were recruited from the 22 counties in Taiwan, including Kinmen and Matsu counties. They must be at least 20 years old. Sampling from the Yellow book, the study conducted stratified random sampling based on the population in each county. A thousand and sixty-eight people were recruited. Random digit dialing was used to make sure numbers that were not registered had the same probability of being selected. The sampling period started from 05/01/2012 to 05/04/2012 during 6:30 to 10:00pm. Two-thousand-nine-hundred-and-eighty-three numbers were successfully sampled. One-thousand-and-eighty were interviewed. The success rate was 36.21%. In order to assure that the statistical analysis matched the demographic property, we conducted a variance test on the valid samples against the population. Although the geographical distribution of the sample was similar to that of the population, there were significant differences between the sample and population in gender and age. Therefore, we adopted *raking ratio estimation of multivariate* to adjust the weight of the variables gender and age in the valid samples until the sample and population were similar in structure, validated by chi-square.

Data Analysis

After the telephone interview, we analyzed the data through IBM SPSS for Statistics. First, we conducted number distribution, analyzed each question item and calculated their average and percentage to obtain a preliminary idea about the participants' opinions. Second, we conducted a t-test and ANOVA on different background variables (gender, age, level of education, family members who attend school and areas of residence). If the t-test was significant, a Levene homogeneity test

was then conducted. If the Levene test was not significant (<.05), the Scheffe's Method was adopted for comparison. If the Levene test was significant (>.05), Dunnett's T3 was adopted for comparison, in order to avoid the problem of homogenous variable hypothesis. In addition, since the participants sometimes misunderstood the questions, answers such as 'I don't know' or 'I don't want to answer' were taken as missing values.

Results and Discussion

1.As far as government responsibility is concerned, most of the participants suggested that schools should not be able to freely raise the tuition or adjust the tuition with price level. Instead of subsidizing public schools, government should subsidize underprivileged students. On the issue of government responsibility, more than half (57.7%) of the participants disagreed on the issue that schools should be able to freely raise the tuition. In addition, 66.9% of the participants were of the opinion that tuition rates should not be adjusted with inflation or price level. As for government subsidy to higher education, up to 84.4% of the participants were supportive of the government reducing the amount of money given in subsidy to public schools, and increase the financial aid for underprivileged students instead. Participants in general were of the opinion that government has the responsibility to monitor college tuition, and regulate the tuition from floating with price level or being raised by schools. Government should also economize and spend the money more efficiently by providing financial aids to underprivileged students.

2.In terms of corporate responsibility, participants suggested that corporate organizations should take the social responsibility to share part of the cost of higher education. Females and participants aged from 20-39 were more supportive of the government raising funds from corporate organizations to assist underprivileged students. In terms of corporate responsibility, the overall results showed that 88% of the participants agreed that corporate organizations have the social responsibility of cultivating individuals' talents and sharing the cost of higher education. As for the way of sharing cost, 72.8% of the participants suggested that government should impose bonus tax on corporate organizations and dedicate the money to education. Up

to 91.2% of the participants suggest that government should establish an education fund intended for underprivileged students.

3.As for the responsibility of individuals, most of the participants agreed that loaners can pay off the debts based on their future salary, but disagreed that the government should pay off the debt for the loaner if the debt is not paid off in 25 years. As for the responsibility of individuals, the overall results showed that 88.4% of the participants support the idea that loaners can pay off their debts based on their future salary. When asked if loaners can extend the timeline to pay off their debts, participants tend to be more reserved—only 49% of the participants agreed. Seventy-six percent of the participants disagreed that that the government should pay off the debt for the loaner if not paid off in 25 years.

Conclusion

Most of the participants thought that government should play a key role in the tuition issue, ensuring students' right to education. The results showed that 58% of the participants disagreed that universities can freely raise the tuition. Sixty-seven percent of the participants disagreed that tuition should float with the price level. Eighty-four percent of the participants agreed that government should reduce the amount of money given in subsidy to public schools, and provide more financial aid to underprivileged students. In other words, most of the participants hoped that the government can play a key role in the tuition policy in assisting students who are in need, in order to make sure that students' right to education is protected.

Participants from different generations showed different opinions: the youth disagreed more that tuition should float with price level, and agreed more that government should reduce the amount of money subsidized to public schools and provide more financial aid to underprivileged students. Analyzing the background of the participants, we found that age had an effect on participants' viewpoints on government responsibility. Compared to participants aged from 50-59, participants aged from 20-29 agreed more that government should reduce the money given in subsidy to public schools and provide more financial aid to underprivileged students. Also, they disapproved of tuition rising with price level. As such, during the time

when higher education was only available to the privileged, higher education was greatly subsidized by the government under the low tuition policy, the general public was not aware of the cost of higher education. However, after higher education was made available to the general public, individuals were required to share a larger proportion of the cost, which inflicted upon the younger generation a greater financial burden. Therefore, the government is expected to take the tuition issue under control. In addition, the majority of the participants are more supportive of the government subsidizing underprivileged students instead of public schools.

SESSION

LATE BREAKING PAPERS AND POSITION PAPERS: FOUNDATIONS OF COMPUTER SCIENCE

Chair(s)

TBA

Evolving systems

Igor Schagaev[1*], Luc Blaeser[2], Simon Monkman[3*],

[1] London Metropolitan University 166-220 Holloway Road, N7 8DB, London, England

[2] HSR Hochschule für Technik Rapperswil Oberseestrasse 10, PO box 1475 CH-8640 Rapperswil Switzerland

[3] ITACS Ltd, 157 Shephall View, SG1 1RR, Stevenage, England

*Author to whom correspondence should be addressed: E-Mail: i.schagaev@londonmet.ac.uk, or info@it-acs.co.uk

Abstract. *An approach to design of evolving computer system is presented. Drawbacks of existing systems are explained showing limitations of hardware and software. New design principles, models of representation for algorithms and architecture are introduced and described with explanation how to achieve better parallelism and reduce concurrency. Structure of required system software and hardware for reconfigurable adaptable system are presented and explained. Prototype architecture of evolving system of computer is discussed in terms of gain for performance, reliability and power-saving. Comparison with Berkley approach to future computer systems is given.*

Keywords: *Redundancy; Reconfigurability, Recoverability; Performance-, Reliability-, Energy-wise systems; System software, Hardware design.*

1. Fundamental problem

Human race sequential way of thinking was evolutionally supported by a structure of the languages and accompanied grammars, [1], where differences in dictionaries and alphabets and grammatical rules were visible, but not critical [2].

Indeed, we speak, write and listen each other sequentially - word after word, phase after phrase and, therefore, create *one-dimensional* sequence of information.

With appearance of technological support of logical and arithmetic calculations a processing implemented by computers did follow our habit of "sequentialism". Calculations were implemented by developed hardware.

Modern hardware technology squeezed logic elements down to nano-micron size, and did change hardware structure making electronics mapped into matrix of interconnected blocks, in other words, hardware topology became *two-dimensional*.

Recent attempt to make 3D chips of memory and processors - simple Internet search indicates 40 Millions hits on the subject - that since 2007 IBM, Intel, Samsung, Toshiba and others are progressing in making three-dimensional chips. There are declarations that Moore law will be finally overcame and boost of performance for future computers will be substantial.

It is worth to mention here that "Moore law" actually is not a law of nature or physics, thus discussion of "Moore law" limitations does not have any sense in the first place. This is a subject for research of mass media impact on technological developments, social science researchers might find it interesting also.

There is a strange feeling through, that declared progress will not be achieved; let us imagine a simple model: imagine a rope, with pretty big length, which we need to put into square box of much smaller size.

There is no doubt, this process will take time and boxed rope will not be as convenient to use as straight one. Now imagine that the box became three-dimensional… We are not sure that our ability to use rope efficiently will grow at all…

Adding the need of dynamic changes for some segments of rope, updating or deleting them and reassembling of the rope as a whole - typical functions of run-time system, can reduce an optimism about impact of 3D to next-to-none. "Rope" here means, of cause, a generalization of a program with code and required data.

In brief, Fig.1 illustrates problems we are facing in computer design. It is clear that marriage of one-dimensional program with two or three-dimensional hardware will not be a happy one. It is clear also that dynamic support of this collaboration will cause even more complications.

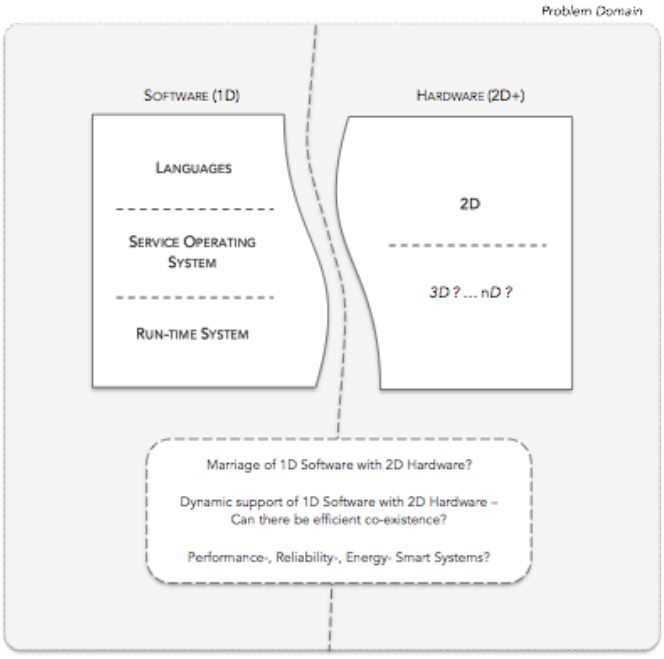


Fig. 1 Software vs Hardware dimension mismatch

2. Computer design problems

A. Known solutions (what we have)

Computer science reincarnates parallel computing since the late 40's - early 50's of the last century [3-12]. Real boom has started just several years ago. An absolutely stunning level of technology and hardware density was recently achieved [14,15], with processor frequencies up to 4.7 Ghz and beyond [16]. Surprisingly, the main questions of computer design and technology are still the same as half a century ago. Specifically, the most challenging goal is design and development of a system with proper use of the dependencies in performance, power consumption, cost, reliability, adaptability and flexibility. None of these dependencies are studied *together* except [17], nobody even set a goal to make system that will be efficient and flexible along requirements mentioned.

Relationship of performance, reliability, energy efficiency and ability to "trade" of this properties through design and observe changing of properties along the life cycle were only initially described by us in [17]. The theory and development of the system with required properties, their interdependencies and modifications (evolution) that consider properties as a processes and not static requirements is in the beginning.

What we have now in ICT does not looks encouraging: in the system design we failed to achieve even doubled performance by applying twice higher frequency and quadruple energy, we failed to achieve any visible system performance growth using the same programs.
Part of the reasons of cause is: computer technological progress was not accompanied or supported enough by a theoretical development, i.e. something serious was missing in the first place. Applying our mantra:
"Power of theory lies in prediction and applicability"
we regretfully accept that computer science has failed to lead ICT world and degraded down to description of technological developments.

In system software the situation is confusing and again, not really optimistic: overcrowded language families (2500 members) [18], slightly less run time systems (500+) [19] manifest full size confusion and an absence of breakthrough in terms of efficient system software and hardware designs and co-existence. Market domination of one or two operating systems or languages has nothing to do with "best player wins" rule.

In turn, state of the art in hardware can be described by example of complete loss of direction - Intel's attempt on an eighty-core processor system has ended up with statement [19]:
"*...Despite using such an efficient grid, the researchers found they could actually hurt performance by adding too many cores. Performance scaled up directly from two cores to four, eight and 16, performance began to drop with 32 and 64 cores. ...*".
Examples prove nothing, but it is clear: a new design strategy is required for next generation of computer architectures. It should start from design phase and pursued through and along the whole life cycle of the system and applications. New design strategy should aim an ability of a system to evolve with support from both: hardware and system software. This evolving feature should be available on demand of user applications and operation requirements.

It is also clear that next generation of computer systems have to address all three mentioned evolving requirements of performance-, reliability- and energy-smart functioning. They should be pursued from the first phases of system design, enabling an efficient trade between P,R,E if user or environment require.

B. Attempt to evolve

Our previous attempts to revise existing designs and find balanced development of ICT hardware and software were presented in [21],[22],[17]. This work summarizes our earlier concepts and developments.

At first, we propose a revision of the area of computer designs and development by introducing a new system paradigm. Any paradigm has to be useful - see our mantra above, thus we have to develop supporting theoretical models and prototype system in both: software and hardware. Several holistic principles we were using are:

- *simplicity and redundancy*
- *reconfiguration and scalability*
- *reliability and fault tolerance*
- *energy-wise design*

The whole process of concept, development, construction of algorithms, and further joint design of hardware and system software up to implementation and maintenance has to be revised pursuing simplicity and

introducing essential redundancy. This defines an ability of future system to reconfigure for the purposes of performance-, reliability- and energy-smart operation.

It means also that new systems have to provide efficient "trading" of reliability, performance and power consumption. *Reconfigurability* might be implemented efficiently only when hardware designs are supported by system software solutions. Semantics and structure of computer hardware vary. While area where information is transformed (we call it active zone) is complex and nothing near to regular, passive zone of hardware - where information is stored has regular structure and highest density of transistors. Interfacing zone as a bridge between mentioned two others serves to maximize speed of data exchange.

Table 1 illustrates how these principles can be supported during the development of new hardware architectures (HW) and system software (SSW).

Table 1 Principles and implementations

!		SIMPLICITY	REDUNDANCY	RECONFIGURABILITY!
HARDWARE ZONES	Active (Processors)	Only essential for instruction execution hardware should be developed, duplicated instruction " for convenience" or compatibility should be excluded; no not necessary back-sire actions of instruction execution, such as signs, flags etc.!	Extra elements and schemes should be added either for improvement of reliability or energy-wise functioning.	Designed processing element should be internally reconfigurable (not used hardware is disabled) and support of reconfigurability of the system
	Interactive (Internal and External Interfaces)	Minimum hardware with when possible state-less designs should be used, using asynchronous logic as much as possible.	Designed interconnections must be identical in terms of functioning, making maximum reconfigurability possible, able to be used as direct access in and onward.	Control of physical and virtual addresses.
	Passive (Memories)	Reduced complexity of address schemes (internal and external).	Virtual memory (of the same type).	Ability to switch data ands program areas, hierarchy of recovery points.
SYSTEM SOFTWARE	Language	At language level - cancelling of complex data structures, limitation and simplification of program descriptions and code structure	Generation of more than one code to run a program for various requirements P.R.E Automatic hierarchy formation of recovery point (also for reliability of operation)	Semantic modification of program (graph-logic and control data predicate models, see further) to reduce dependencies between code segments and data; support at the language and operating systems -service operating system and run-time system), processes.
	Monitor of processes	Simplified support of process synchronisation;	Reducing of number of states in synchronisation systems and hierarchy of processes Implementation of universal loop with ability to control state of hardware and system software at the same time, reducing resource wasting.	Implementation and control of software during run-time and hardware - making "integrity map" possible.
	Reliability support	Recovery points of a program, modules and tasks - to implement hot swap (reconfigurability) when needed. Saving only essential data for concrete execution of a program.	Reducing amount of supportive data in run-time system for program monitoring Introducing of reserved data blocks to run hardware with deficiency (for reliability purpose of view)	

In terms of information processing, computer architectures can be divided into three areas: the processing area – further called active zone, the storage area – called passive zone and the interconnection zone. All three zones have to be reconfigurable for their own purposes and other zones requirements.

Dependencies between zones define the level of reconfigurability and flexibility of hardware and system software parts of system architecture. Note here also that P-,R-,E are no longer static system requirements as they become *modifiable* during life cycle: from the first phases of system design [23] down to maintenance. Thus, evolving requirements themselves have to be considered as *processes*. P-R-E dictates that system software has to be modified, including language, run-time and service operating systems. Figure 2 below

qualitatively shows that system performance (P), Reliability (R) and Energy (E) smart design might be implemented by means of various methods (circles) and to achieve any development we should choose only not-mutually exclusive solutions.

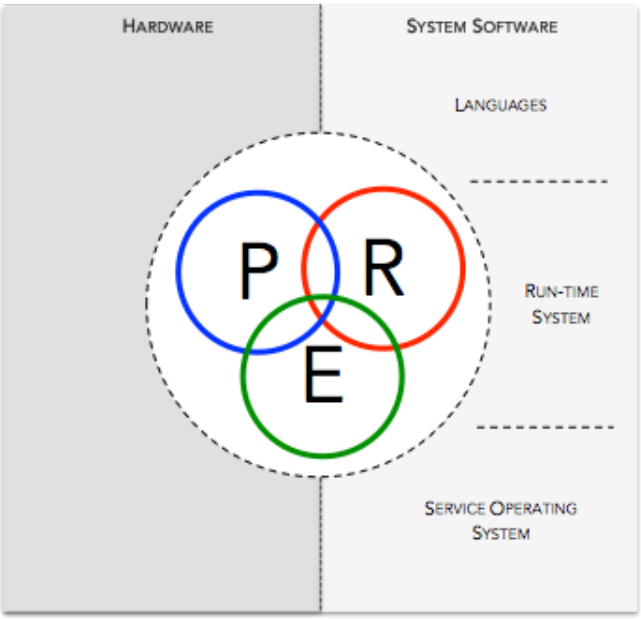


Fig. 2 System properties at system software

It is worth to mention that properties of system software that are required and convenient at program preparation time are not required at all during program run i.e. they should be "stripped off" program code and instead "fine tuning" of code to fit existing hardware applied. Communication or interaction between agents, processes should also be implemented taking into account P-,R-,E-requirements. One of immediate reflection from Fig.2 regarding performance is: parallelisation of program should be prepared and supported as much as program structure enables, or even more, while concurrency within program and during execution should reduced to minimum or excluded. Below we present just one approach on attempt of system software redevelopment for performance improvement.

3. Proposed approach (what we need and why)

At first, we should choose the nearest in rigorousness language and run-time system and define a sequence of steps enabling the system software involvement in reconfigurability, se Table above.

To make reconfigurability working we describe program and hardware using meta-models, that are independent from technological aspects but describing structural properties of the algorithm and programs: Control Data Predicate (CDP) and graph-logic (GLM) [22].

Languages of structural programming, modular programming [24], [25] and object oriented programming [26], [27] were invented to provide the programmer a higher level of abstraction.

There is no doubt, separation of concerns increases visibility, clarity and eases the process of programming. These properties, regretfully are not necessary and even reduce performance during the execution, when HW load and power consumption are crucial.
At the same time, in terms of P,R,E properties of the system all attempt to deal with parallelism and concurrency of programs and their execution are still far from providing substantial gain, in spite of extreme efforts [28],[29]. Dijkstra [30] has published extension of Dekker algorithm for solution of concurrent execution of multiple processes and this solution in one form or another is used within languages and hardware (instructions like "test and set", IBM360). The problem of automatic extraction of the program parts that are independently executable - parallelism is still "work in progress" and attracts attention of researches on a regular basis, all around the implementations of mutual exclusion [31],[32],[33].

In discussions of performance gain system from software parallelism vast majority of publications are oriented on Amdahl's law. Regretfully, Amdahl's simplification of efficiency of parallelism by cutting - see our mentioned earlier rope into ***n*** chunks does not reflect the fact that chunked segments of the rope *should be delivered* into processing areas, and results *after* processing *should be collected back* usually into one block of data - i.e. in Amdahl ratio denominator grows, *reducing* gain from parallelism even more.

Thus the roles and impacts from language, service operating system and run-time system on support of parallelisation were largely ignored and, therefore, promised by Amdahl gain was never achieved. Works and publications [14],[15],[34],[35] highlighted problems of massive parallelism in future hardware designs, showing that proposed approaches either limit seriously or make impossible to cope with requirements of configurability and scalability.

Due to architectural and technological limitations, known hardware designs are not flexible enough in the mapping of algorithms to the HW, and above all, do not support changing of software structures or hardware configurations "on the fly".

Commercial systems (Windows, Linux, etc.) have shown that attempts on effective use of HW features have largely been a failure: the chip frequency now goes beyond 5Ghz, but hardware design and existing software reduced useful system performance down to the range of hundreds Mhz. Therefore, future architectures must be re-designed with justification of concepts and arguments from the user level of abstractions down to the hardware level of representation maximizing required characteristics.

Technologically, new hardware chips have almost astronomical numbers of logic gates and high physical density giving substantial advantage for programming multiple-data problems when it is possible to use matrix algebra or multiple data tasks with simple instructions. a

the same time, density of hardware elements has reached technological limit of thermal density. As it was already mentioned, accordingly the principle of separation of concerns the programming language has to be hardware independent as this way the logic and complexity of an algorithm will be visible and separated from hardware implementation. From another point of view we should improve efficiency of hardware and software work. So the question remains:

How and what to modify in algorithms and implementations to achieve PRE-smartness of a system?

At first, the compilation process of a program should be "fine tuned" for execution on the specific hardware. Additionally, reconfiguring hardware before program execution also reduces overheads.

Examples prove nothing, unfortunately. In much more general terms program tuning for hardware performance or reliability or energy-smart operation is becoming some kind of reversed programming, when hardware, not the user (programmer) takes the dominant role. Thus we might separate explicitly "what is good for user from what is good for system and hardware" and, when possible, spread these requirements along life cycle of software and system development, as Table 2 illustrates.

Thus separation of concerns and goal of PRE-smartness requires supportive models of implementation from the system software point of view (language and run time systems) aiming improve hardware efficiency.

Table 2 Phases of software development

User oriented phases (UW)			
Concept	**Design**	**Compilation**	**Implementation**
System software and hardware oriented phases			
Analysis of the program through graph logic model to find intrinsic parallelism in control, data and predicate dependencies.	Recompilation of the program into parallel form with introduction of hardware configuration and reconfiguration features into program as well as concurrency management – formation of the execution string	Dual String Execution (New RT data structure that unites arrays and records)	

Two models we propose before for this are called control-data-predicate (CDP) and graph-logic- (GLM) models [2] .

3. Supportive models

Within CDP model each operator in a program has defined in terms of modifications of three connected graphs: control (C), data (D) and predicates (P). Thus operator is described by elements (nodes) in each graph; the volume of the resulting predicates, state registers, processor and program status words (PSW) physically represented in the hardware by so-called processor status registers, contributes into the "width" of each layer of these graphs.

To support and exploit parallelism, all three graphs should be simplified by separation of data and control, making them as independent as possible. Simplification of graph of predicates is achieved by deliberate fragmentation i.e. we propose that predicates should be processed when they needed for decision making and not stored in the hardware before that.

Support of reliability and fault tolerance requires detection of possible change of program state caused by hardware deviation and development of recovery schemes of hardware and software [22]. Therefore, any complex operator and hardware instruction respectively jeopardise the possibility of generating 'snapshots' of the previous hardware states as well as the flexibility of program segmentation, allocation and reconfiguration. Regretfully, in the vast majority of architectures, as well as languages (their compilers) and run-time systems the functions of data access and data processing are mixed, tightly coupled, and hardware state modification due to program execution is not controlled explicitly, making condition change latent and execution of a program unjustifiably complex.

In processors such ARM, Intel and SPARC, the arithmetic and logic unit (ALU), or even several of them, as well as shifters, registers, internal cache, special registers and pipeline sequencers are active during the execution of each instruction. The complexity of hardware handling becomes enormous: translation look ahead buffers, caches, synchronization and pipelining logic occupy 75% of the die size. However, none of these overhead required from the programming language and program operators points of view.

Any condition of hardware related to the operator or instruction representation requires checking. This makes parallelism or reliability unachievable. Reasoning is different through: while for parallelism hardware should be designed as "flat as possible", the reliability demands limitation of fault propagation through hardware schemes. The complexity of a system and the implementation cost of parallelisation or fault tolerance are directly related to the amount of the resulting modifications of the hardware and program states thus when P (predicate) is used only for the selection of the program flow, a special operator and instruction can be defined to generate the current value of P and store the result in a register, making system reconfigurability easier to achieve. To summarize, the implementation of parallelization at the level of the instruction set, the design objectives will be:

- *Simplification of predicate logic in a program;*
- *Mapping of language operators as close as possible to the modified instruction set of the processor;*
- *Reducing size of the program state required to be saved before instruction's execution.*

As a further improvement of parallelisation and at the same time concurrency reduction we need to handle the actions that were initiated in parallel, but eventually ended up in conflict of access to one or several resources, i.e. *concur* after resources. To describe this we introduce the graph-logic-model (GLM). Note that GLM might be applied for any of graph of the CDP model. Until now research in parallelism was mostly targeted at finding parallel branches of programs and independent data elements. However, expecting pure parallelism is hardly feasible: what is initiated as parallel segments ends up ultimately in concurrent mode, competing for a resource such as a socket, printer, data concentrator, etc. The rare exception, such as graphic processors with high numbers of SIMD-like processors just proves the rule. The simple notation similar to one introduced in http://www.it-acs.co.uk/files/Transaction_GLM.pdf can describe program structures and hardware structures consistently in terms of co-existing concurrency and parallelism. GLM *explicitly* separates parallel and concurrent elements in the system description by introducing logic operators in the program graph for incoming and outgoing ends of edges. The application of the logic operator XOR (exclusive OR) on an input or output of an edge defines ALL possible concurrencies in the program graphs. In turn, all possible parallelism in the control graph are defined by finding all outgoing or incoming edges explicitly described by the AND operator. The same approach might be applied for the data and predicate dependency graphs of the CDP model. Having a correct program, CDP and GLM can then be applied to extract the parallel segments and data paths and help in reducing concurrency. This reversed programming gives us a chance to play with the software and monitor software redundancy, (deliberately introduced at the recompilation phase) and hardware redundancy (introduced at the design phase) on the fly. It also enables to use reconfigurability of the system for performance, reliability, or energy-smart functioning when it is necessary.

Figure 5 proposes sequence that clearly separates properties required at the *program writing phase* with properties required *during execution* as latter are dependent on the available hardware resources available and changeable during execution. This way hardware features (that might be dynamically adapted during runtime) become represented in the program logic.

1. Debug sequential algorithm (SA)
2. Create a model of the SA as triple <C,D,P> where C,D,P are graphs of control, data and predicate dependencies of SA
3. Apply the graph logic model to modify C,D
4. Where possible substitute "strong" logic operators by "weak" ones. An "IF" might be substituted by XOR (to start) and OR, or AND or NOP, leaving an option to parallelise segments of the algorithm
5. Check all introduced XOR in sequence for potential concurrency in the algorithm – for the OS

Fig. 5 Making parallel

4. System software for evolving systems

A new language is needed to naturally express algorithms in a form that supports program recompilation into a redundant form that fits the "hardware view". This does not mean that other programming paradigms such as structured or OO are no longer useful. It means though that they have their limits and serve mostly for user convenience and interfacing. The proposed paradigm serves to improve performance of the end product by design of a new reconfigurable architecture, system software and transformation of existing software into a form that is efficiently executable by adaptable hardwaree. Note that parallelisation might be applied to all three graphs: control, data and predicate, deducing the best parallel form. The availability of hardware serves as the termination condition for this algorithm. The same algorithm might be reapplied during execution.

The system software for next generation of evolving systems includes a new modular programming language called Active Language (AL) and two operating systems: a runtime system, called the Active Reconfigurable Run Time System (ARRUS) and service operating system (SOS). AL and ARRUS are tightly coupled, as AL needs runtime support for some features, such as recovery go program.

The language AL derivates from Oberon [26], [27] and COMPOSITA [36,37], includes the following new features:

- *A new data structure is introduced that eases the mapping of data to memory (dual string model);*
- *GLM extensions is exploited in the control and operator model representation;*
- *Separation of interface and implementation to support dynamic software and hardware reconfiguration*
- *Reduction number of supported data structures*
- *Physical separation of constant, global and local variables and introduction of recovery points [18]*
- *Support for recovery and reconfiguration points at the module level using special program structures*

AL revises language constructs such as unbounded loops and introduces calculation of upper execution times and stack sizes to ease certification. Rigorous memory management implemented without pointers and references. AL inherited from COMPOSITA that programs are entirely composed of active components which govern strict encapsulation, dynamic wiring with a dual concept of offered and required interfaces and communication-based interactions (Fig.6).

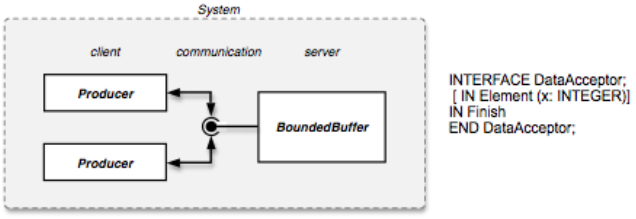


Fig. 6 Hierarchical component structures

As the language is based on hierarchical composition and does not employ any ordinary pointers or references, surrounding components properly controls the deletion of components and no garbage collection is needed for safe memory management [36,37].

Figure 7 shows an example of a component structure, where a component can contain an inner network of sub-components.

Communications follow a formal protocol written in an EBNF-like notation. Due to the strict encapsulation the components and can be easily mapped to various hardware architectures. It inherently enables parallelism (N components may be scheduled on up to N processors) as well as redundancy (the same components may be executed as multiple replicated instances). In the ongoing work, a prototype compiler and run-time system for evolving system were already developed.

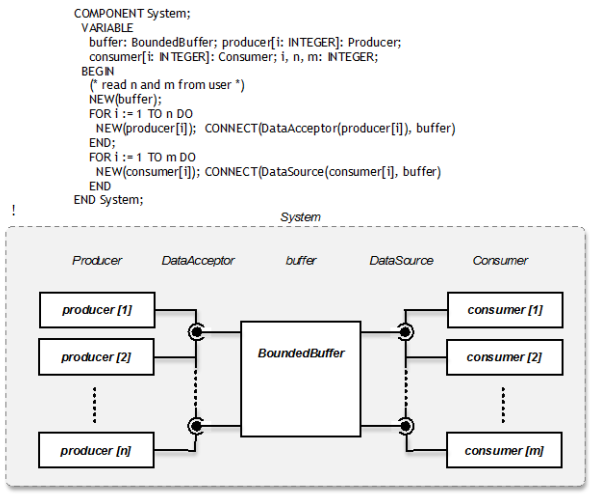


Fig.7 Component interactions

Direct support for reconfigurability and recoverability of program structures at the language level makes reconfiguration of system possible in case of hardware degradation due to faults or task special requirements of system power-saving operation or on the opposite boosting task by using maximum hardware resources for completion a task with required time limits.

For hardware fault tolerance especially due to substantial ratio of malfunctions over permanent fault system software at the language level should include recovery points for program at various levels of program presentation: procedure, module, task.

Detailed description of language support of hardware fault tolerance using recovery points is presented in [22] where we show that the recovery point scheme will be embedded in the language and oriented on the programs and data structure reducing the overhead for recovery after malfunctions and eases the impact of possible permanent faults.

ARRUS. Run time system ARRUS monitors real-time processing as well as real-time reconfiguration of the underlying hardware elements and the respective network topology, including:

- *Flexible dynamic hardware resource management*
- *Performing of software reconfiguration to adapt to changes of HW states and system conditions*
- *Management of hardware/software interactions in presence of hardware faults*
- *Hardware state monitoring and support of graceful degradation using testing and recovery procedures and reconfiguration management*

To match the required features such as reconfigurability, parallelisation, real time, resilience to hardware degradation and distributed control processing, the ARRUS itself is built in a strict hierarchical manner, as it is illustrated in Fig. 8.

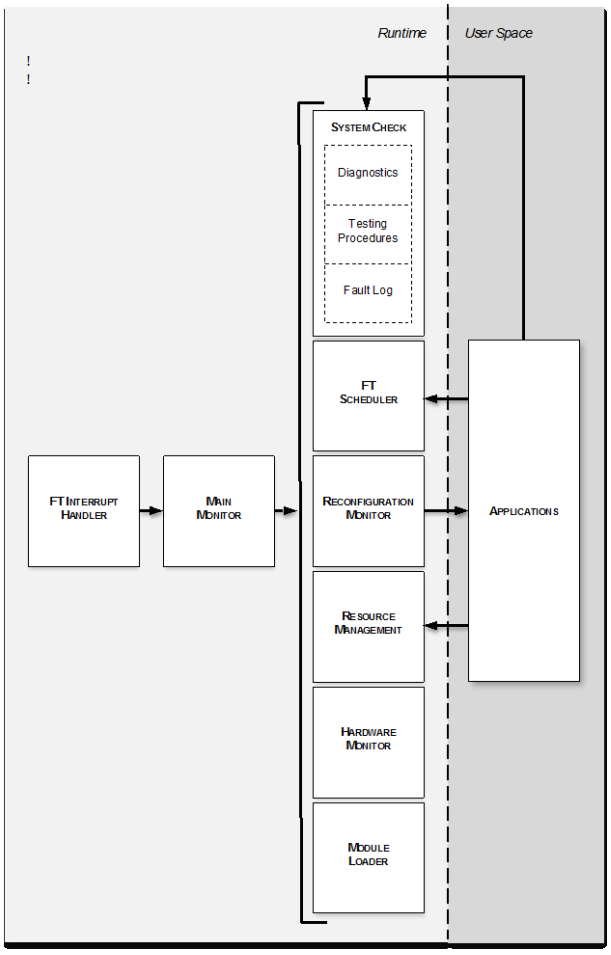


Fig.8. ARRUS architecture

The lowest level module has no dependencies at all and consists of the main system monitor which is responsible for the coordination of all activities, such as the initialization of reconfiguration entities, timer services (not shown), interrupt handling and all the remaining depicted functions.

ARRUS provides also all standard functions of a runtime system such as memory management, which are well known and explained in literature [34], [35]. These features are omitted in Fig.8 to keep the diagram

understandable. In a standard control loop system, it is up to the programmer and the applications to diagnose faults and react appropriately. In ARRUS however, this is not the responsibility of the application but of the runtime system.

Thus, ARRUS is responsible for diagnosing faults (software failure, malfunction, permanent fault, etc) and notifying the appropriate software and hardware monitoring services of any required changes. User applications are not allowed to communicate directly with the reconfiguration mechanisms. The rationale behind this principle is the idea that the runtime system is the only entity that knows the current hardware state, all ongoing processes and their resources. In case of a fault, it can thus based on the available resources reconfigure the applications.

The following developments will be crucial for the ARRUS fault handling mechanism:

- *Monitor HW state. Possible either with interrupts (HW signals changes), or periodic software initiated hardware checking (further both approaches are required to implement hierarchical HW check) [22].*
- *Reaction on HW state changes. The runtime system is responsible for the management of the hardware states and reconfigures the applications accordingly*
- *Collaboration of checking and recovery processes at the system software level and HW level. Introduction of fault resilient task scheduling and HW / SW fault handling strategies. (schedule simpler task versions)*
- *Fault tolerant semaphores: A new concept that eliminates deadlocks caused by HW deficiencies.*

ARRUS handles changes of hardware conditions using the notion of hardware states and transitions, as Figure 9 shows. A hardware element in the states Active, Master or Slave is included in the current working configuration. If an element is in the state Stand-by, it is not active but can be activated in a further reconfiguration step.

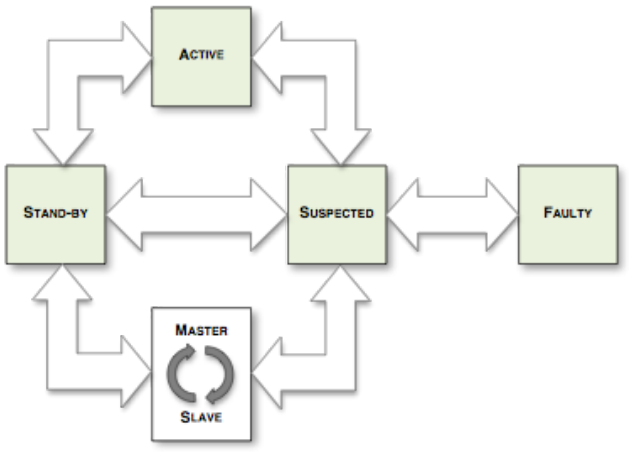


Fig. 9 Hardware state graph at ARRUS

If a fault is detected, the affected HW element is set into the state Suspected which means that at least for a while (until the generalized algorithm of fault tolerance [18] is completed and a new valid configuration is established), this HW element will not be included in any working configuration.

This representation of the hardware state for every ERA element defines the current configuration of the ERA hardware at the run-time system level. A configuration change can be triggered by changed application, power, reliability or performance requirements, or a detected error.

5. Evolving system: hardware

Basic schemes. The indicative structure of hardware element for evolving system is presented in Fig.10. Configurators called T-logic provide flexible use of processor and memory elements in the system configuration for performance and health conditions: when one processor is dealing with its own program (self testing, autonomous calculations) it disconnects from the other nodes. The same technique is used to form reconfigurable hardware that is capable to adjust to program requirements or react to other events such as detected permanent faults.

The memory configuration of EvSy element (Fig. 10) work for both: resilience or performance. A special logic scheme, called T-logic configures of the memory structure.

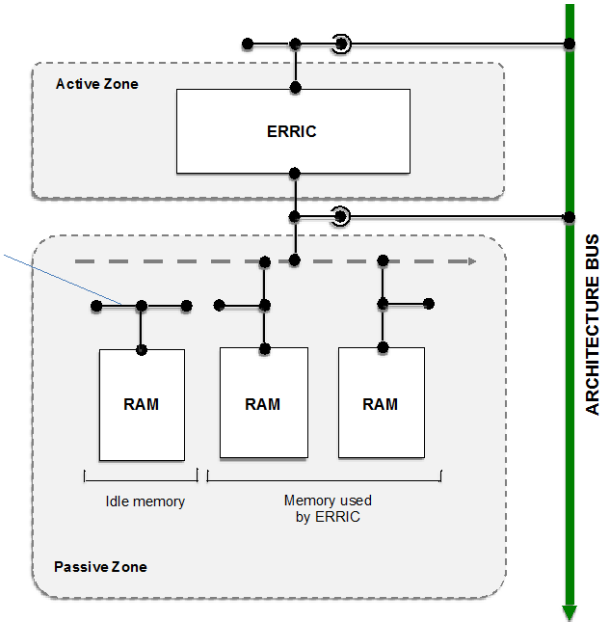


Figure 10 - EvSy element

When an application requires maximum reliability, the T- logic scheme might configure the memory as a 4, or 3 (shown) unit with voter. The configurations: two to compare and one spare or three independent memory elements are possible. The number of memory elements might vary, as for recent implementations four memory elements were proposed. However, the principles of configurability by using T-Logic elements remain the same.

The instruction set of elementary processor is designed to recover from hardware malfunctions by repetition of the instruction making malfunction tolerance efficient [18]. In comparison with Motorola, ARM, Intel proposed EvSy is much simpler, and a higher level of parallelism and frequency can be achieved. Due to simplicity by-design EvSy needs only 10% power compared to the competitors.

The same technique is used to form a hardware configuration adjustable to the program requirements or when a hardware element itself (or architecture) detects hardware faults and thus can't be involved in further program execution either on a temporary or permanent basis. The final decision about permanent isolation of an element is performed during a special mode of self–healing when testing and recovery procedures are executed.
Each element can be turned off individually to decrease power consumption. Note that the structure assumes only one leading element at a time enforced by a "rotation" of the T-logic element. T-logic makes the whole EvSy possible to operate until the last soldier stands: i.e. until at least one processor and memory element can communicate.
By design, the evolving recoverable reduced instruction computer is able to recover from major malfunctions and repeat the ongoing instruction when an error was detected. The specially designed instruction set and its declared top-down implementation of all required features reduces the impact of malfunctions on the performance and reliability of a system as a whole.
As it was shown in [21], the efficiency of malfunction tolerance depends on the fault coverage, and the amount of involved redundancy. The efficiency of malfunction tolerance grows together with the ratio of malfunction to permanent faults.
Finally, the absence by design of pipelining and ability of several accesses memory during instruction execution enables to monitor and vary processor frequencies when applications require extreme performance. Figure 11 below demonstrates first prototype of EvSy designed and developed by authors in collaboration with ITACS Ltd and initial run made by V.Castano.

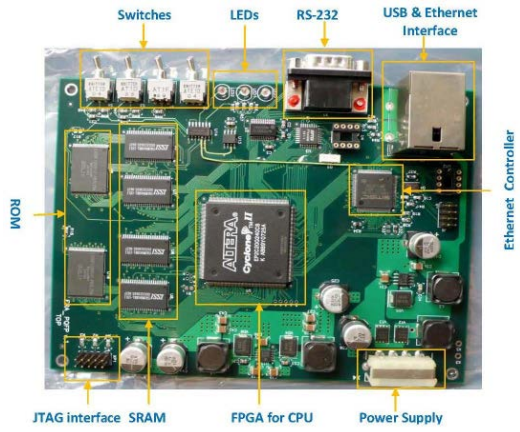


Fig.11 - EvSy element - first prototype

Memory organized as 4 mutually replaceable schemes with virtual addressing that configure EvSy for reliability (two pairs or triple structure with spare element), energy-smartness (one element is active the rest are disabled) or full capacity - when all elements form one memory bank for program. ROM is functions as one or both active. At the moment all reconfigurations and EvSy processor elements (4 of them) are assembled in Altera PLD shown. Further development obviously will lead to special chip for processors, memories with reconfiguration support and special design of T-logic element to achieve maximum simplicity, performance and reliability.

The design of proposed reconfiguration scheme has been completed, PLD level development in full progress; it is worth to note EvSy reconfiguration is able almost linear exchange or reliability/performance/power consumption, enabling user task define the actual configuration through special request to runtime system.

6. Multi-element configuration

There is no doubt - see "model of rope and box" above - one has to address *how* resources of multi-agent systems supported by multi-element architecture.

Conceptually, each element of EvSy might work and serve for providing *system* property of evolving: it can be turned off individually to decrease power consumption, or turn to be inactive but able to be included in required by application configuration, or aggregated with others for maxim capacity or reliability.

Applied through the whole system T-logic elements make the whole system to operate "until the last soldier stands": i.e. until a single processor, called ERRIC on the figure, and a single memory element can communicate through remained links.

Note that active and passive elements (processor and memory) can be at the far ends of the system. Various areas of Evolving Architecture can be involved in different type of tasks.
Thus some segments might run safety critical extremely demanding in terms of reliability tasks, while other execute heavy calculations of matrix algebra.
Proposed approach is promising. Initially it was presented at 2005 FET Infoday January 13,2005 in Brussels by Prof Schagaev (Londonmet) with assistance of Prof Gutknecht (ETH Zurich). Since then hardware prototype and essential system software was developed.

7. EvSy approach vs. Berkley view

Recently published Berkley view on parallel computing [38] differ with our approach. The Table 4 illustrates difference, is self-explanatory and show both – system software and hardware concepts and implementations.

Table 4 Berkley view vs evolving reconfigurability approach

Berkley view	Evolving reconfigurability approach
Seven goals (and 11 bullet points) • Future computers must be effectively parallel: • It is possible to consider 1000+ cores; • Performance measurement for parallel computing should be re-evaluated and new metrics introduced.	• New rigorously defined computing paradigm based on GLM exploits automatically maximum possible parallelism of both: algorithm and hardware available and miniml7es synchronisation complexity at the run time to reduce concurrency. This approach guarantees together with flexibility of hardware configuration for both: performance or reliability purposes. Adjust available hardware for maximum possible efficiency across the whole broad class of applications usually covered by semantically different architectures such as VLIW, SIMD. MIMD.
• Auto-timing of software and hardware; • Human centric computing in multi-core; • Application of wide range of data typing.	• Software will be written in traditional way, and debugged on a standard systems; • Parallelization of the algorithm will be done by backward compilation of a sequential program. using GLM again for representation of task potential parallelism and fine-tuning of hardware resources available on the wafer, before and during application run
• Three levels of parallelism should be pursued: task level word level and bit level • Parallel programs must be presented independently to the number of processors available • Limitations on features that reduce parallelism • OS functionality should be based on libraries and virtual machines.	• Task level parallelism is prerogative of run time system and heavily dependent on workload and available resources during application run: therefore, dynamic scheduling to optimise task parallelism is exception not the rule in evolving architecture. For evolving reconfigurable architecture dynamic support of parallelism should be an exception not a rule. • Application of GLM for both redesign of algorithms and timing on architecture available resources (supported by T-configurator at the element and architecture levels) maximize parallelism and minimise concurrency.
• Higher level rate of fault is expected for multi-core systems.	• Assumptions about higher rate of permanent hardware faults for the next generation of electronics are not correct. More details see, for example in Feynman lectures...
• SEC/DED options proposed	• Number of malfunctions caused externally by alpha particles and internally due to higher density of elements on the wafer does not lead to increased reliability by using SEC/DED; most likely 16+ bit errors will take place in hardware. System software contribution to the malfunction of the system caused by support of dynamic parallelism and complex concurrency monitoring.
• Synchronization overheads should be reduced.	• Overloaded by task parallelism, the monitoring OS will be deadlocked. We propose a reconfigurability of available hardware and tasks by recompilation of existing algorithms.
Wide range of data types should be implemented: • 1 bit (Boolean); • 8 bits (Integer, ASCII); • 16 bits (Integer, DSP fixed point, Unicode): • 32 bits (Integer, Single-precision FP, Unicode); • 64 bits (Integer, Double-precision FP); • 128 bits (Integer. Quad-Precision FP; Large integer (>128 bits))	• We propose *Dual string data structure* where for each data element of the array a special descriptor defines data type (altogether 232 types). • This covers all possible data types user can dream or imagine. Efficiency of access to the proposed data structure is equal to the array access.
• New models should support proven styles of parallelism, • FPGA systems are future HW platforms for multi-core computing.	• Styles of parallelism are application specific and vary due to technology modification. We propose an auto-tuning of existing programs into their maximum parallel form. • FPGA technology at the element level and specially designed wafer with pre-fabricated configuration fabric of active processing element and passive elements enables monitoring architecture for performance, power consumption and reliability.

8. Conclusion and future work

I.Known drawbacks of computer architectures blocking efficient use of reconfigurability of hardware and software for performance, reliability or energy-wise functioning are analysed.
II.Several holistic principles were presented and pursued through the whole life cycle of computer systems: from the preparation of algorithms down to the execution of programs and hardware.
III.Proposed approach introduces system evolving features of mutual design of architecture and system software.
IV.From system software point of view centre of gravity for hardware reconfigurability is based at compiler level, leaving only essential reconfigurability handling at the run-time level.
V.Shown that consistent support of reconfigurability eases parallelisation, reduces concurrency, assists fault tolerance and implement power-awareness for applications when necessary.
VI.Two models: control-data-predicate dependency of the program and graph-logic represent concurrency and parallelism of a program and enable their explicit separation.
VII.A sequence proposed to find the parallel for program potentially reducing time overheads up to the level limited only by available hardware resources.
VIII.System reconfigurability pursued and supported by programming language, service operating system and run-time system results ultimately in a simple, yet scalable, reliable system, and providing performance, reliability and power saving options with linear trade between.
IX.Hardware elements and the prototype with maximum reconfigurability (till the last soldier stands) are described.
X.A comparison of the evolving reconfigurability approach with known approaches was presented.
XI.Philosophically speaking, a degree of freedom in any system defines its adaptability and evolvability. A system's ability to adapt, use or exploit fundamental limitations depends *externally* on properties of nature, while *internally* force us to design future systems giving us an option to exploit topologic features, hardware and system software making evolving achievable.

9. References

[1] Hannas William C., "The writing on the wall," University of Pensilavaniya Press, 2003, ISBN 0-8122-3711-0.
[2] Schagaev I, Elisabeth Bacon E, Georg Hagel G , Michail Charnine M, Kirk B, Kravtsov G. Weduca: Web-enhanced design of university curricula, 07/2013; DOI:ISBN: 1-60132-235-6,In proc.: FECS'13.
[3] Goldstine A. and Goldstine, H. H. (1946) ENIAC, The Electronic Numerical Integrator. Math Tables and Other Aids to Computation, vol. 1, pp. 97-110
[4] Everett R, and Swain, F., Report R-127 (1947) Whirlwind I Computer Block Diagrams. MIT Servomechanisms Laboratory.
[5] Whirlwind I, Master Drawing List And General Rack Layout Of Computer. (1952), MIT, Dept. of Electrical Engineering, Cambridge.
[6] Smithsonian Institute (1990) Comp. history. http://americanhistory.si.edu/collections/comphist/
[7] Hofstra University, (1999), History in the Computing Curriculum. Appendix 4 1950–59. www.comphist.org/pdfs/CompHist_9812tla4.pdf
[8] Holland J.H. (1960) Iterative circuit computers, IRE-AIEE-ACM computer conf, May 3-5, 1960, San Francisco,pp. 259- 265,
[9] Newell, A. (1960) A On programming a highly parallel machine to be an intelligent technician,IRE-AIEE-ACM'60, May 3-5, pp. 267-282.
[10] Schwartz, E. (1961) An Automatic Sequencing Procedure with Application to Parallel Programming, Journal of the ACM (JACM), v.8 n.4, p.513-537, ACM, New York
[11] Slotnick D. L., et al. (1962) The SOLOMON Computer, AFIPS '62 (Fall) Proc., December 4-6, 1962, Philadelphia, pp. 97-107, ACM, New York.
[12] Squire, J. S. and Palais, S. M. (1963) Physical and Logical Design of a Highly Parallel Computer, Proc. SJCC
[13] Gosden J. Explicit parallel processing description and control in programs for multi- and uni-processor computers, AFIPS '66 (Fall): Proceedings of the November 7-10, 1966, pp. 651 – 660, ACM, NY
[14] Nair R (2002) Effect of increasing chip density on the evolution of computer architectures, IBM Journal of Research and Development, vol. 46, num. 2/3, IBM
[15] LaPedus M., EETimes (2003) Intel gears up 90-nm processor, chip set rollout http://www.eetimes.com/conf/idf/showArticle.jhtml?articleID=10800811&kc=3172
[16] IBM Power6 Microprocessor and IBM System p 570, 05.21.07
[17] Monkman S., Schagaev I. Redundancy + Repeatability = Recoverability, Electronics, 2013, 2, 212-233; doi:10.3390/electronics2030212,ISSN 2079-9292
[18] Kinnersley, Bill The language list. (November 2009) http://people.ku.edu/~nkinners/LangList/Extras/langlist.htm,
[19] Berka S. Operating System Documentation Project. (Nov 2009) http://www.operating-system.org/betriebssys tem/_english/index.htm,
[20] Ames B. (2007) Intel tests chip design with 80-core processor. http://www.macworld.com/news/2007 /02/12/intel/
[21] Schagaev I. Reliability of malfunction tolerance, Proc. IMCSIT, vol. 3, Wisla, Oct 20 – 22, 2008, pp. 733-737.
[22] Kaegi T., Schagaev I. System Software Support of Hardware Efficiency, ITACS 2013,ISBN 978-0-9575049-0-5
[23] Plyaskota S., Schagaev I. Economic Effectiveness of Fault Tolerance. Automation and Remore Control, 07, 1995, PP 1017-1026.
[24] Tursky W,Wasserman A. Computer programming methodology, SIGSOFT Softw. Eng. Notes, 1978, vol. 3, No. 2, pp. 20-21, ACM, NY
[25] Wirth N. Programming in Modula-2, 1989, Springer,Berlin.
[26] Wirth N. Gutknecht J. Project Oberon: The Design of an Operating System & Compiler, 1992, Addison-Wesley
[27] Wirth N. The Programming Language Oberon. 1988, Software – Practice and Experience, vol.18, num. 7, pp. 671-690, John Wiley & Sons Inc., NY
[28] Flynn M. (1972) Some Computer Organisations and Their Effectiveness, IEEE Transactions on Computers, vol. C-21, pp. 948-960, IEEE.
[29] Fisher A. (1983) Very Long Instruction Word architectures and the ELI- 512, Proc. of the 10th annual Int-l Symp on Comp. Architecture, Stockholm, Sweden, pp. 140-150, ACM, New York
[30] Dijkstra E. W. (1965) Solution of a problem in concurrent programming control, Comm of the ACM, vol. 8, num. 9, pp. 569, ACM New York
[31] Lamport L. (1983) The weak Byzantine Generals problem, Journal of the ACM, vol. 30, No ., pp.668–676, ACM New York.
[32] Lamport L. and Melliar-Smith, P. (1985) Synchronising clocks in the presence of faults. Journal of the ACM, vol 32, num.1, pp. 52–78, ACM, NY
[33] Gutknecht J. The Dining Philosophers Problem Revisited, JMLC 2006, Lecture Notes in Computer Science, vol. 4228, pp. 377 – 382, Springer-Verlag, Berlin
[34] Hennesy J. and Patterson, D. (2008) Computer organisation and design, 4th ed., Morgan Kaufmann
[35] Bryant R. and O'Hallaron, D. (2002) Computer Systems: A programmer's perspective. Prentice Hall
[36] Bläser L., A Component Language for Structured Parallel Programming. Proc JMLC, Oxford, UK, Sept. 2006.
[37] Bläser L. A High-Performance Operating System for Structured Concurrent Programs. Pr-ngs of the Workshop on Programming Languages and Operating Systems (PLOS), October 2007
[38] Asanovic et al.,(2006) The Landscape of Parallel Computing Research: A View from Berkeley. Technical Report Nr. UCB/EECS-2006-183, Dec.

Control operators vs. Graph Logic Model

Igor Schagaev
London Metropolitan University, London, UK
i.schagaev@londonmet.ac.uk, also info@it-acs.co.uk

Abstract - *Simple paradigm to unite control operators for programming languages into one scheme using graph-logic representation of relations between agents (or elements of interaction) assuming independence of behavior for each element is presented. Shown that power of this structure exceed known models of description of behavior for concurrence and parallelism. Proposed model explicitly separates concurrency and parallelism and indicates further steps to automatic reprocessing programs for making them better tuned to modern architectures.*

Keywords: Computer languages, Control structures, Concurrent structures, Logic, Graph Theory.

1 Introduction

Every algorithmic language describes decision actions using logic statement of selection:

- "if" to choose one of two options
- "case of" to choose options from more than two
- "while" when our decision depends on conditions with uncertain time trigger or other independent parameter change

This is well supported by classification of relations introduced by E. Kant [1], whom I consider as a first theoretical programmer, opposing to a sentimental story of Ada, lady-lover of lord and part-time poet Byron. (Frankly, Nabokov's ADA makes much more sense to me).

E.Kant classified statements in terms of relationships and possible interactions between elements involved. Accordingly E. Kant an object might correspond, relate, and interact with others using the following relationships:

- *One-to-one*,
- *one-to-many*,
- *many-to-one*,
- *many-to-many*

And it seems to me nice and easy, provided we make decisions where to go, what to choose and our decisions are mutually exclusive. That is why, by the way, each processor instruction set has operator XOR and set-and-wait.

2 New Control Scheme

Unfortunately or fortunately our decisions are not always that simple as presented above: we can be friendly with different groups of people, make not mutually exclusive decisions, starts selective actions with various taboo: ""you can go your party but you do not drink and back before 11pm!" - remember? ;), etc., etc.

And this is all executes at the same time... Thus E.Kant diagram should be extended, one option of extension, called GLM is shown below on Figure 1. GLM stands for graph logic model to describe mutual dependency of various kind, was successfully applied in real world applications, including active conditional control systems, active safety monitoring systems, overpowers descriptive power of Markov and Bellman models and similar. Accordingly GLM, leaving one state, say "a" we might describe our leaving conditions using logic basic operators {AND, OR, XOR} attached to a leaving end of the links between "a" and neighbors.

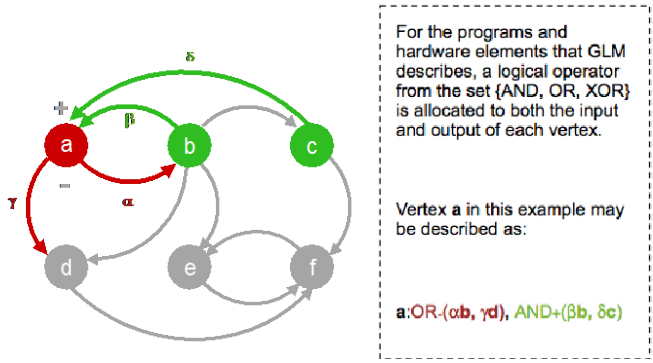


Figure 1 Graph Logic Model of interactions between nodes

Note that *at the same time* the various options are possible: selection of leaving conditions to several neighbors using {OR} on each out-coming link, or broadcasting to all neighbors through all out-coming links using {AND}, or picking just one and only neighbor using {XOR}.

Still, we think that we are the important ones and make impeccable decisions. This claim stands when we act and all others just follow…

But how about civil disobedience (for more on the subject I recommend to read Henry David Thoreau "On civil disobedience" or Gandhi passive resistance, when no matter what and which government (in first example US in the second UK) instructing how to obey and we do not follow and note – we act differently?

How to describe Vichy's collaborationism and De Gaulle resistance at the same time existing in France during WW2? How to describe Italian type strikes when people sit in office and do nothing? (Sound like EC...)

Did anybody spot - we are talking distributed computing now, as we have introduced various modes of reaction of opposite side of link and have to accept it's own will to act without our "instructions".

From now on the interactions between nodes with logic based decision rules applied to "own socket" for both sides of link are assumed!

All these examples of other nodes involvement in interaction force us to attribute the same link twice - at the leaving end and at the incoming end with different logic operators if necessary, Figure 2.

Figure 2 Logic operators for incoming and out-coming links

Using GLM we are able to describe mentioned above political phenomena and much wider and wilder conditions appeared in really of complex models without difficulties.

As another extreme example, when we assume that all logic operators within graph are operators are XOR we converge GLM into Markov model. In turn, adding weights (Greek letters on the graph of Figure 1) on links (cost, time other independent variables required) and assuming, again XOR as only operator allowed for out-coming and in-coming links we are able to describe Bellman optimization model using GLM notation.

Thus nodes and links between them with attributed logical operators attached to each leaving and coming ends form, in fact, new basis for control operators for next generation of programming languages.

3 Concurrency and Parallelizm

Almost everything that starts together, or at the same time or using the same data sooner or later will face a conflict of interests - parallel branches of program will require final aggregation of it into few numbers or functions; access to hardware, or informational, or time resources will be limited and conflict arises.

There are very few pure parallel program and systems - to name one known Sony PlayStation or any digital TV set - where incoming data flow splits and distributed in parallel to display visual elements.

For all the rest existing descriptive schemes of parallel program are not actually correct or useful. Use of GLM might help here:

What we start in parallel (leaving condition is AND for each link from a chosen node) might be completed in mutually exclusive mode (incoming condition XOR).

Using Figure 1 example traces **a-d-f** and **a-b-e-f** can be activated in parallel and eventually end up with conflict of interests, thus each of incoming links to node **f** should be attributed with XOR operators.

Tracing of branches of a program with attaching operators is becoming interesting area of research as we are able using GLM to separate really independent segments and allocate them properly on existing or next generation [2] hardware.

4 Conclusions

Graph logic model provides exceptional flexibility for expressing of control in various environments of interacting agents.

Attributed logical operators attached to each leaving and incoming ends of edge form new scheme of control operators for next generation of programming, when number of co-existing active agents will interact voluntarily.

5 Acknowledgements

Figure 1 was suggested and drawn by Simon Monkman, - my former student and good friend. Value your support, Simon.

6 References

[1] E. Kant "Critics of pure reason", Everyman ISBN 0-40-87357-X, 1993

[2] Schagaev et al. ERA – Evolving Reconfigurable Architecture. ERA Proc. of Conf: Software Engineering, Artificial Intelligence, Networking. and Parallel/Distributed Computing – SNPD 2010, PP 215-220.

Concrete State Machine Language (CSML)

Nelson Rushton, Texas Tech Dept. of Computer Science
June, 2014

---

1. Introduction

Software programs do essentially two things: compute and communicate. Mainstream languages, such as C++ and Java, are based on a core set of general purpose constructs that can be used for either task. The hypothesis of CSML is that these two subtasks place different and independent demands on the language, and thus that things are simplified by combining a computing-only language with a communication-only language.

CSML is named after ASML (Abstract State Machine Language), invented by Yuri Gurevich in the mid 1980's as a high level (or, more accurately *adjustable level*) algorithm specification language. It has since become an industry standard, though in systems engineering more so than in software. ASML was conceived as a procedural pseudocode, using a set of functional primitives that may vary from program to program. The only constraint on the primitive functions of an abstract state machine (ASM) is that the human reader of the pseudocode understand their meaning. The name *concrete* state machine comes from the fact that, rather than leave the primitive functions of the machine as pseudocode, or implementing them as lower level state machines (a common procedure known in the ASM community as *refinement*), we plan to implement the primitive functions of the machine in a functional programming language.

2. Syntax

A ***variable declaration*** is either an empty string or an expression of the form `vars` $x_1 : S_1$ … $x_n : S_n$, where each x is an identifier and each S is a SequenceL sequence.

An ***action*** is one of the following:

- $x := E$ (where x is a state variable and E is a term)
- `playSound(`*s*`)` (where *s* is a sound)
- `blit(`*M,k*`)` -- (where *M* is a set of images and *k* is an integer). *k* is the priority of the blit, where blits with higher priority cover up blits of lower priority.

A *rule* is one of the following:

- an action
- `if` $p : R$ (where p is a Boolean expression and R is a rule)
- a string of two or more of actions

An *init statement* is written `init:`*R* where *R* is a rule

The sequential and nested structure of rules and declarations is represented by spacing and indentation, as in Python.

A *concrete state machine* consists of a variable declaration, an init statement, one or more rules, and a SequenceL program.

3. Semantics

An *input port* is one of the following:

- `mousePosition`
- `leftButtonDown`
- `rightButtonDown`
- `keyPressed(`*c*`)`, where *c* is the ascii code of a character

A *state variable* of a CSM is an identifier appearing on the left hand side of an assignment in its initialization. The functional program that accompanies a CSM can be written treating any state variables and input ports as constant symbols. A definedfunction that references one of these symbols in its definition is called a *fluent,* since it may return different values when called with the same arguments when the machine is in different states.

The semantics of rules is as follows:

- The semantics of actions is assumed to be clear.
- An *instance* of a rule *if p:R* is obtained by replacing each of its logical variables with a value from the sequence which is the range of that variable. When a rule is fired, each of its instances fires in parallel. if *p* is true then *R* is executed.
- The sequence $R_1,...R_n$ of rules is executed by executing each of its rules in random order (possibly in parallel)

The semantics of the entire CSM is as follows: First the initialization rule is executed, then all other rules are executed once per time step, in any order (including, possibly, in parallel), until the program is interrupted by an outside signal (such as the user closing the program window). The following is an example of a CSM for a tic-tac-toe application (the SequenceL implementations of the machine's primitive functions of the machine are not shown). This can be thought of as the procedural component of the program.

```
// This is a CSML program for a tic tac toe game. To run, it would require
// functions --- emptyBoard:Seqence<Int>, turn: Player -> Bool,
// empty:Cell->Bool, clicked:Cell->Bool, restartClicked:Bool, occupies:
// Player*Cell->Bool, gameOver:Bool, resetButton:Sprite, hashMarks:Sprite,
// resultsMessage:Sprite
```

```
// A *player* is either “x” or “o”, and a cell is an integer in [1...9].

vars:
  player:["x","o"]
  cell:[1...9]


// The board starts out empty

init:  board:= emptyBoard


// If it is a player’s turn and an empty cell is clicked, that player
// now occupies that cell.

if turn(player) &  empty(cell) & clicked(cell): board[cell]:=p


// If the restart button is clicked, the board returns to its
// original empty state.

if restartClicked: board := emptyBoard


// Video output
////////////////////////////////////////////////////////////

// Always display the hashmarks and the reset button.
blit (hashmarks ++ resetButton)

// If a player occupies a cell, show his mark there.
if occupies(cell, player): blit(playerMark(cell,player))

// If the game is over, show a message reporting the result
if gameOver:  blit(resultMessage)
```

Conclusions and overview

CSML is a simple (though outside-of-the-box) procedural language which, when combined with a functional language, can be used to write fairly complex interactive applications. The basic idea has been vetted through its use as a standard tool in systems engineering.

We estimate that in a typical CSML program, 90% of the code would be functional and 10%

would be CSML. This is based on programming commerical-scale applications using SequenceL is a pure functional language that is used with C++ to create applications.

CSML is inherently event driven rather than sequential/looping/branching, so that the main loop and “event driving architecture” that typically forms the boilerplate of large application fades into the background semantics of the language. With functional computation also encapsulated into a separate component, we can view CSML simply a language for nothing but event handling. Handling events is a pretty simple concept, intuitively (*any time this happens, do this*), and the fact that CSML does this and nothing else allows it to conform to the simplicity of the intuition.

The use of logical variables (as in Prolog) rather than loops (as in C, etc.) makes the syntax closer to natural language and common sense. For example, compare

If a bird lands in the tree and eats a berry, blow the whistle

with

For all b in Birds, for all T in trees, for all r in berries, if B lands in T and eats r then blow the whistle.

This difference can be seen in the tic-tac-toe program in a couple of places, when compared with what the corresponding code would look like in, say, java.

There is a lot of nondeterminism in the control flow of CSM’s. Therefore, they are amenable to artificially intelligent automatic parallelization. However, the resulting parallel executable cannot be guaranteed, in general, to be race free. Generating warnings about race conditions for CSML is a toic for future research.

To be used as a full-scale application development tool, CSML would have to somehow incorporate a wrapper for whatever read and write operations were needed. This would be a big undertaking to do for general purposes. The proposed solution to this is to enable the programmer to write their own actions and input ports in the target compilation language (say, C++). That way they just use CSML for control flow architecture. The message, then, is “You can use whatever read/write commands you want to interface with hardware, and we will give you an easier way to describe the logic of your program.”

References:

[1] Cooke, Daniel; Rushton, Nelson; Nemanich, Brad; Watson, Robert G.; Andersen, Per (March 2008), "Normalize, Transpose, and Distribute: An Automatic Approach

for Handling Nonscalars", *ACM Transactions on Programming Languages and Systems (TOPLAS) TOPLAS Homepage archive* (New York, NY, USA: ACM) **30** (2)

[2] Y. Gurevich, Evolving Algebras 1993: Lipari Guide, E. Börger (ed.), Specification and Validation Methods, Oxford University Press, 1995, 9-36. (ISBN 0-19-853854-5)