An Abstract Machine for Reasoning about Situations, Actions, and Causality

Kerstin Eder [1] , Steffen Hölldobler [2] , and Michael Thielscher [3]

[1] University of Bristol, Department of Computer Science, Queen's Building, University Walk, Bristol BS8 1TR, UK, eder@compsci.bristol.ac.uk
[2] Wissensverarbeitung, Informatik, TU Dresden, D-01062 Dresden, Germany, sh@inf.tu-dresden.de
[3] International Computer Science Institute, 1947 Center Street, Berkeley, CA 94704-1198, USA, michaelt@icsi.berkeley.edu

In: R. Dyckhoff, H. Herre, and P. Schroeder-Heister, ed.'s, *Proceedings of the International Workshop on Extensions of Logic Programming (ELP)*. Springer LNAI 1050, pp. 137–151, Mar. 1996.

Abstract. Over the last years several new approaches for modeling situations, actions, and causality within a deductive framework were proposed. These new approaches treat the facts about a situation as resources, which are consumed and produced by actions. In this paper we extend one of these approaches, viz. an equational logic approach, by reifying actions to become resources as well. Using the concept of a membrane we show how abstractions and hierarchical planning can be modeled in such an equational logic. Moreover, we rigorously prove that the extended equational logic program can be mapped onto the so-called chemical abstract machine [1]. As this machine is a model for parallel processes this may lead to a parallel computational model for reasoning about situations, actions, and causality.

1 Introduction

Over the last years several new approaches for modeling situations, actions, and causality within a deductive framework were proposed [2, 7, 12]. It turned out that these approaches are equivalent to some extent since they are all based on the same idea: Facts about a situation are treated as resources, which are consumed and produced by actions [5]. This treatment allows for solving the frame problem [9, 10] within a purely deductive framework without the need to state frame axioms, non–monotonic laws of inertia, or successor state axioms (cf. [13]).

In this paper we will concentrate on one of the just mentioned new approaches, viz. on the equational logic approach [7]. There, facts about a situation are reified and represented as multisets of terms. The multisets themselves are

represented with the help of a binary function $\circ$ which is associative, commutative, and admits a unit element. In other words, $\circ$ is an AC1–function. In [7] the respective equational axioms are built into the unification computation, the actions are represented as definite clauses, and SLDE–resolution is used to generate plans transforming a given initial situation into a goal situation. The approach was later refined to also incorporate specificity, and it was shown that SLDENF–resolution [15, 16] is a sound and complete inference rule for this extension [8].

In the previously mentioned deductive approaches planning is performed within the so–called space of situations, i.e., the situations are first–class objects which are transformed by means of actions. The actions themselves are given as axioms and, therefore, are not first–class objects. In recent years, however, the most advanced AI planning systems operate on the space of plans, where actions and (partial) plans are first–class objects and an abstract initial plan is refined until the actions of the refined plan are executable on some machine or by some agent. Besides the fact that actions should be first–class objects, such plan–based planners require to model hierarchical and partial–order plans.

In the first part of this paper we will concentrate on modeling actions as first–class objects and hierarchical plans within an equational logic framework on the basis of [7]. The key new idea is to treat actions as resources, which are consumed whenever they are applied, and which are generated by combining somehow "simpler" actions. Here, the notion of "simplicity" is defined with respect to a given hierarchy of actions. On the lowest level of such a hierarchy actions are assumed to be always available, which is modeled with the help of the exponential "!" borrowed from linear logic [6].

Let us illustrate this approach with the help of a "standard" hierarchical planning example adapted from [14]. Suppose an agent named Gisela comes home from work. All she wants is to relax and watch TV. The problem of watching TV can be described as a planning problem with the initial situation of Gisela being at the door (d) of the living room, the TV being unplugged (u), the actions *enter* and *turn–on–TV*, which enable Gisela to enter the living room (l) and to turn on the TV (on) respectively, and the goal situation where the TV is on. Obviously, an initial plan could consist of the sequential execution of the actions *enter* and *turn–on–TV*.

However, an action like *turn–on–TV* is rather abstract as it omits many details, and an agent may be unable to execute such an actions directly. Rather the actions may need to be refined to the three actions *go–to–TV* for moving next to the TV (n), *plug–in* for plugging in the TV (in), and *switch–on* for switching on the TV. One should observe that a further refinement of these operators is possible, but for the purpose of this paper the above is in sufficient detail. Thus, the initial plan given in the previous paragraph can be refined to contain the consecutive actions *go–to–TV*, *plug–in* and *switch–on* instead of the action *turn–on–TV*.

As a first main result we will show how such a planning problem can be solved within an equational logic programming framework. We will demonstrate

that the view of actions as resources is important if we want to model hierarchical planning. Thereafter, we will concentrate on how such a deductive, hierarchical planning system can be implemented. One could, of course, run these programs on a standard PROLOG–system if the AC1–theory is translated into a suitable predicate and this predicate is called whenever two expressions are to be AC1–unified. Unfortunately, due to the inherent features of AC1–unification, this is far from being efficient without additional program analysis and transformation techniques [3]. The obvious other choice is to map the equational logic programs onto a suitable abstract machine. Abstract machines, like the Turing Machine, are widely used in the classical theory of sequential and parallel computations. But what kind of abstract machine should we target for? While the Warren Abstract Machine (WAM) [17] and its variants are candidates, again the standard search process carried out by a WAM has to be interleaved with the AC1–unification computation, and this causes considerable difficulties.

We therefore opted for an abstracted machine which has some sort of AC1–unification already built in, viz. the Chemical Abstract Machine (or CHAM, for short) [1]. Originally, the CHAM is a machine model for parallel processes. Molecules are floating within a solution, which is stirred by some mechanism. Some of the molecules may interact according to certain interaction rules. More formally, the solution is a multiset of molecules and the reaction rules are multiset rewritings. In the terminology of situations, actions, and causality, facts about a situation as well as actions are floating in a solution, and the interaction rules specify the application of actions.

The second main result of our paper is that the equational logic approach for reasoning about situations, actions, and causality can be mapped onto an extended version of the CHAM. This opens the door not only for efficiently implementing the equational logic programs, but also for applying actions concurrently. As mentioned, the CHAM is an abstract machine for parallel processes in that molecules may interact concurrently as long as the various interactions do not interfere.

In Sect. 2 we formally define the kind of logic programs we are dealing with. Sect. 3 contains a brief description of the CHAM. In the main section, 4, we specify the extension of the CHAM as well as the transformation from SLDENF–resolution to multiset rewritings, and show that the transformation is correct. Finally, we discuss our result and point out possible future developments in Sect. 5.

2 Equational Logic Programming and Planning

Traditionally, reasoning about situations, actions, and causality was modeled within a conceptual framework where a situation is a snapshot of the world at a particular moment, and actions are the only means to transform one situation into another [9, 10]. In this paper, however, we adopt a more general framework based on a chemical metaphor first presented in [1]. There, a situation is represented as a *chemical solution* in which floating *molecules* can interact according

to *interaction rules*. The molecules are terms representing either facts or actions, situations are finite multisets of molecules, and the interaction rules are multiset rewritings. Among the interaction rules are those where actions are applied if their conditions are satisfied. But there are also more complex interactions like those where two actions are combined to yield a more general action. To deal with such combinations as well as with abstraction and hierarchical planning, a molecule is allowed to contain a subsituation enclosed in a membrane, which can be somewhat porous to allow communication between the encapsulated situation and its environment.

More formally, the reification of both actions and facts is modeled as follows.

- *Molecules* are either *agents* $\langle \mathcal{C}.\mathcal{E} \rangle$ [4] or *generators* $!\langle \mathcal{C}.\mathcal{E} \rangle$, where $\mathcal{C}$ and $\mathcal{E}$ are multisets of terms of the form $\emptyset$ or $\{\!|\, t_1, \ldots, t_n \,|\!\}$ [5], $n \geq 1$, representing *conditions* and *effects* of an action or sequence of actions, respectively. *Facts* about a situation are molecules of the form $\langle \emptyset.\mathcal{E} \rangle$, i.e. molecules, whose condition is the empty multiset. The notion of a generator is borrowed from [6] and denotes an unlimited source of molecules.
- A *situation* is denoted by a multiset of molecules and (sub-)situations of the form $\emptyset$ or $\{\!|\, m_1, \ldots, m_k \,|\!\}$, $k \geq 1$, enclosed in a membrane, such that no m_i and m_j, $i \neq j$, share any variables. Membranes are unary function symbols denoted by a surrounding box $\boxed{}$. For notational convenience we omit the parenthesis when writing membranes.
- The *airlock mechanism* is used to pass molecules from one level of abstraction to the next higher level. It extracts an agent from a situation, keeps the rest of the situation within the membrane, and isolates the extracted agent within an airlock attached to the membrane. The airlock construct is written as $m \lhd M$, where m is the isolated agent and M is a single molecule representing the remaining situation encapsulated within a membrane.

A (*hierarchical*) *planning problem* consists of two situations $\mathcal{S}$ and $\mathcal{T}$ and is the problem of whether there exists a substitution σ and a sequence of transformations such that this sequence transforms $\sigma\mathcal{S}$ into $\sigma\mathcal{T}$, where the set of transformations consists of the following rules.[6]

- Merging: Two agents $\langle \mathcal{C}_1, \mathcal{E}_1 \rangle$ and $\langle \mathcal{C}_2, \mathcal{E}_2 \rangle$ may interact iff there is a substitution σ such that $\sigma\mathcal{E}_1 \mathbin{\dot\cap} \sigma\mathcal{C}_2 \neq \emptyset$, and, if they interact then they are merged into the new agent $\langle \sigma\mathcal{C}_1 \mathbin{\dot\cup} (\sigma\mathcal{C}_2 \mathbin{\dot-} \sigma\mathcal{E}_1), \sigma\mathcal{E}_2 \mathbin{\dot\cup} (\sigma\mathcal{E}_1 \mathbin{\dot-} \sigma\mathcal{C}_2) \rangle$.
- Generator: A generator $!\langle \mathcal{C}, \mathcal{E} \rangle$ can be weakened to $\langle \mathcal{C}', \mathcal{E}' \rangle$, $!\langle \mathcal{C}, \mathcal{E} \rangle$, where the $'$-notation denotes that variables are standardized apart.

[4] Note that by representing agents in this manner we come very close to the semantics of the $\multimap$ operator in linear logic [6] in so far as an agent $\langle \mathcal{C}.\mathcal{E} \rangle$ can be interpreted as the material implication $\mathcal{C} \multimap \mathcal{E}$ between its conditions $\mathcal{C}$ and its effects $\mathcal{E}$.

[5] The terms t_i, $1 \leq i \leq n$, are "simple" terms in that they do not contain multisets as subterms nor are they multisets themselves.

[6] Throughout the paper we will use the brackets $\{\!|$ and $|\!\}$ to denote multisets, and the operators $\mathbin{\dot\cap}$, $\mathbin{\dot\cup}$, and $\mathbin{\dot-}$ denote the multiset operations corresponding to the set operations $\cap$, $\cup$, and $-$, respectively.

– Airlock: An agent m_i in a membrane $\boxed{m_1, \ldots, m_n}$ can be pushed through the membrane to yield the airlock $m_i \lhd \boxed{m_1, \ldots, m_{i-1}, m_{i+1}, \ldots, m_n}$.
– Release: An airlock $m \lhd M$ may release the agent m to become m, M.

The merging rule is used to combine two actions and to satisfy the conditions of actions. The generator rule is a form of controlled weakening adapted from linear logic [6]. It is used to model the most primitive actions within a given scenario. We assume that there is an unlimited supply of these most primitive actions, or, in other words, an agent may execute such an action as often as he needs to without consuming any resources. The molecules enclosed in a membrane are encapsulated and the merging as well as the generator rule can be applied freely within the membrane. Occasionally, however, a membrane may want to communicate with its environment. Therefore, the airlock and the release rules may be used to push agents through the membrane. One should observe that the airlock rule can easily be made sensitive to allow only certain agents to pass through the membrane. In the CHAM these two rules are bidirectional, but for the purposes considered in this paper it suffices to move agents into one direction.

It is now straightforward to specify an equational logic program which solves hierarchical planning problems. Multisets are represented with the help of a binary function symbol $\circ$, written infix, which is assumed to be associative, commutative, and admits the unit element $\emptyset$, i.e., the constant $\emptyset$ itself corresponds to the empty multiset. The entire equational theory AC1 is built into the unification computation, and we will use the predicate $=_{AC1}$ to denote equality modulo this theory.

We can now turn to the specification of the program. First of all, there is the predicate[7] $\mathit{transform}(S, T)$ with intended meaning that the initial situation S can be transformed to the goal situation T by a (possibly empty) sequence of interactions.

$$\mathit{transform}(S, T) \leftarrow \mathit{interact}(S, S') \wedge \mathit{transform}(S', T). \tag{1}$$

$$\mathit{transform}(S, S') \leftarrow S =_{AC1} S'. \tag{2}$$

The predicate $\mathit{interact}(S, T)$ is intended to be true when the situation T can be obtained from the current situation S through an interaction.

$$\begin{aligned}\mathit{interact}(\langle C_1.E_1 \circ E_1' \rangle \circ \langle C_2 \circ C_2'.E_2 \rangle \circ Z, \langle C_1 \circ C_2.E_1 \circ E_2 \rangle \circ Z \rangle) \\ \leftarrow\ E_1' =_{AC1} C_2' \ \wedge\ E_1' \neq_{AC1} \emptyset.\end{aligned} \tag{3}$$

$$\mathit{interact}(!A \circ S, A' \circ !A \circ S) \leftarrow \mathit{rename}(A, A').^{8} \tag{4}$$

$$\mathit{interact}(Z \circ \boxed{S}, Z \circ \boxed{S'}) \leftarrow \mathit{interact}(S, S'). \tag{5}$$

[7] Throughout this paper, we use a PROLOG-like syntax, i.e., constants and predicates are in lower cases whereas variables are denoted by upper case letters. Moreover, all free variables are assumed to be universally quantified.

[8] The predicate *rename* is used to standardize variables apart.

$$interact(Z \circ X \lhd \boxed{S}, Z \circ X \lhd \boxed{S'}) \leftarrow interact(S, S'). \tag{6}$$

$$interact(Z \circ \boxed{\langle C.E\rangle \circ S}, Z \circ \langle C.E\rangle \lhd \boxed{S}). \tag{7}$$

$$interact(Z \circ \langle C.E\rangle \lhd \boxed{S}, Z \circ \langle C.E\rangle \circ \boxed{S}). \tag{8}$$

Clause (3) implements the merging rule, (4) the generator rule, (5) and (6) denote that interactions may be within membranes and airlocks, respectively, whereas clauses (7) and (8) denote the airlock and release rule.

The program requires SLDENF–resolution, i.e., where the equational theory AC1 is built into the unification procedure and negative literals are solved via negation–as–failure [15, 8, 16].[9] With the above clauses for *transform* and *interact* , the equational theory AC1, and SLDENF–resolution we can now solve the planning problem presented in the introduction. On the highest abstraction level, the problem is phrased as the following goal.

$$\leftarrow transform(\langle \emptyset.d\rangle \circ !\langle d.l\rangle \circ !\langle l.on \circ l\rangle, \langle \emptyset.on \circ Z\rangle \circ S). \tag{9}$$

The variables Z and S occurring in the goal specification represent the other effects resulting from executing the generated plan and other resources still available, respectively. By applying (4) repeatedly to the two generators occurring in (9) using (1) we obtain

$$\leftarrow transform(\langle \emptyset.d\rangle \circ \langle d.l\rangle \circ \langle l.on \circ l\rangle \circ !\langle d.l\rangle \circ !\langle l.on \circ l\rangle, \langle \emptyset.on \circ Z\rangle \circ S).$$

The actions *enter* ($\langle d.l\rangle$) and *turn–on–TV* ($\langle l.on\rangle$) can now be successively applied to the fact $\langle \emptyset.d\rangle$ using (3) and yielding

$$\leftarrow transform(\langle \emptyset.on \circ l\rangle \circ !\langle d.l\rangle \circ !\langle l.on\rangle, \langle \emptyset.on \circ Z\rangle \circ S).$$

From this goal the empty clause can be derived by applying (2) obtaining the answer substitution $\{Z \mapsto l,\ S \mapsto !\langle d.l\rangle \circ !\langle l.on \circ l\rangle\}$. The generated plan itself can be extracted from the SLDENF–refutation and consists of the action *enter* followed by *turn–on–TV*.

So far we have solved the problem as if the actions *enter* and *turn–on–TV* were always available and could be executed by our agent Gisela. However, Gisela may be a robot and the actions that Gisela can perform are simpler than *turn–on–TV*. Hence, such a complex action must be synthesized from those simpler ones. Now assume that the simpler actions are the ones mentioned in the introduction, i.e. the actions *go–to–TV* ($\langle l.n \circ l\rangle$), *plug–in* ($\langle u \circ n.in \circ n\rangle$), and *switch–on* ($\langle in \circ n.on \circ in \circ n\rangle$). It has to be shown, of course, that these simpler actions can indeed be combined to achieve the complex action. To do so, we replace the generator $!\langle l.on\rangle$ occurring in (9) by the membrane

$$M = \boxed{\langle \emptyset.u\rangle \circ !\langle l.n \circ l\rangle \circ !\langle u \circ n.in \circ n\rangle \circ !\langle in \circ n.on \circ in \circ n\rangle}\ .$$

[9] One should observe that the only need for negation–as–failure is the test whether $E_1' \neq_{AC1} \emptyset$ in (3).

In other words, we now assume that there is an unlimited supply of the simpler actions. We have also given some additional fact about the initial situation, viz. that the TV is unplugged ($\langle \emptyset.u \rangle$). Thus, we obtain the new goal

$$\leftarrow \mathit{transform}(\langle \emptyset.d \rangle \circ !\langle d.l \rangle \circ M, \langle \emptyset.on \circ Z \rangle \circ S). \tag{10}$$

We could solve this goal, if the complex action *turn–on–TV* could be generated by applying interaction rules within the membrane M. Once generated, the complex action can be pushed through the membrane with the help of (7) and (8) and, finally, the refutation can proceed as in the case of the refutation of (9). It is easy to see that applications of (4) followed by applications of (3) transforms the membrane M into

$$\boxed{\langle l.on \circ in \circ l \circ n \rangle \circ !\langle l.n \circ l \rangle \circ !\langle u \circ n.in \circ n \rangle \circ !\langle in \circ n.on \circ in \circ n \rangle} \ .$$

One should observe that the generated complex action $\langle l.on \circ in \circ l \circ n \rangle$ has additional effects compared to the action $\langle l.on \circ l \rangle$ used in the refutation of (9). This, of course, is no surprise as we have developed a more refined plan and in doing so more information about the conditions and effects of the involved actions becomes available.

Surely, one need not stop at this level. The action *go–to–TV*, for example, may itself be synthesized from even simpler actions which may require battery fuel for moving from one place of the living room to another one, etc. In this case, the generator $!\langle l.n \circ l \rangle$ has to be replaced by an appropriate membrane. In general, only the actions on the lowest level of abstraction are modeled as generators, whereas the actions on the other levels of abstraction are generated within membranes and then pushed through the membrane to the next higher level. The following result shows that hierarchical planning can be achieved purely deductively.

Theorem 1. *The planning problem consisting of the situations $\mathcal{S}$ and $\mathcal{T}$ can be solved with substitution σ iff there is an SLDENF–refutation of*

$$\leftarrow \mathit{transform}(\sigma S, \sigma T) \ .$$

wrt. P, where P denotes the equational logic program specified in this section and S and T are representations of $\mathcal{S}$ and $\mathcal{T}$, respectively.

This theorem can be obtained by induction on the length of interaction sequence. Moreover, with the help of an appropriate lifting lemma, the result can be lifted such that SLDENF–resolution may be used to compute σ or an even more general (modulo AC1) answer substitution (see [4]).

3 The CHAM

The chemical metaphor used for hierarchical planning in Sect. 2 was adapted from the Chemical Abstract Machine (CHAM). The states of this machine are

chemical solutions where floating molecules interact according to transformation rules. It was first introduced by G. Berry and G. Boudol to model asynchronous concurrent computations [1].

Formally a CHAM is specified by molecules $m_1, m_2, \ldots$, solutions, i.e. multisets of molecules, $S_1, S_2, \ldots$ and a set of interaction rules. The molecules are terms of an algebra and the solutions are finite multisets of molecules, written $\{\ m_1, m_2, \ldots, m_k\ \}$. In each CHAM any solution S can itself be considered as a single molecule and can thus appear as a subsolution. The corresponding operator $\{\ \}$ is called the membrane operator.

The interaction rules of a CHAM are multiset rewritings of the form

$$m_1, m_2, \ldots, m_k \rightarrow m'_1, m'_2, \ldots, m'_l\,,$$

where m_i and m'_j are molecules; they are presented as *rule schemata*, and the actual rules are instances of these schemata. The rules determine an interaction relation $S \rightarrow S'$ between solutions. All interactions obey the following four laws:

- The Reaction Law. An instance of the right-hand-side of a rule can replace the corresponding instance of its left-hand-side. If

$$m_1, m_2, \ldots, m_k \rightarrow m'_1, m'_2, \ldots, m'_l \tag{11}$$

  is a rule and $M_1, M_2, \ldots, M_k, M'_1, M'_2, \ldots, M'_l$, are instances of the m_i's and the m'_j's, then

$$\{\ M_1, M_2, \ldots, M_k\ \} \rightarrow \{\ M'_1, M'_2, \ldots, M'_l\ \}. \tag{12}$$

- The Chemical Law. Reactions can be performed freely within any solution:

$$\frac{S \rightarrow S'}{S \,\dot\cup\, S'' \rightarrow S' \,\dot\cup\, S''} \tag{13}$$

- The Membrane Law. A subsolution can evolve freely in any context[10]:

$$\frac{S \rightarrow S'}{\{\ C[S]\ \} \rightarrow \{\ C[S']\ \}} \tag{14}$$

In addition some CHAMs use the airlock-construct. An airlock is a molecule of the form $m \lhd S$ where m is a molecule and S is a solution. Airlocks are build and suppressed by the following law:

- The Airlock Law:

$$\{\ m\ \} \,\dot\cup\, S \leftrightarrow \{\ m \lhd S\ \} \tag{15}$$

[10] We use the context notation $C[]$ — as in λ-calculus — to denote a molecule with a hole [] in which to place another molecule.

Since we want to follow as closely as possible the specific CHAM given in [1], we are next going to introduce the syntax of this machine together with its set of transformation rules. Let $\mathcal{N} = \{a, b, \ldots\}$ be a set of names and $\mathcal{L} = \{a, \overline{a} \mid a \in \mathcal{N}\}$ be a set of labels. The symbols $\alpha, \beta, \ldots$ are used to range over labels. There are four basic operators, which are ' 0 ' (inaction), ' . ' (prefixing), ' | ' (parallel) and ' \ ' (restriction). Molecules are of the following form:

$$\begin{aligned} m &::= p \mid \alpha.m \mid m\backslash a \mid S \mid m \lhd S \\ \text{where } p &::= 0 \mid \alpha.p \mid (p_1|p_2) \mid p\backslash a. \end{aligned}$$

The set of transformation rules contains the following:

$$p_1|p_2 \rightleftharpoons p_1, p_2 \tag{16}$$

$$a^+.p_1, a^-.p_2 \rightarrow p_1, p_2 \tag{17}$$

$$0 \rightharpoonup \tag{18}$$

$$(\alpha.p)\backslash a \rightleftharpoons \alpha.(p\backslash a) \text{ if } \alpha \notin \{a, \overline{a}\} \tag{19}$$

$$p\backslash a \rightleftharpoons \{\!|\ p\ |\!\}\backslash a \tag{20}$$

$$\{\!|\ m, m_1, m_2, \ldots, m_n\ |\!\} \rightleftharpoons \{\!|\ m \lhd \ \{\!|m_1, m_2, \ldots, m_n\ |\!\}\} \tag{21}$$

$$(\alpha.p) \lhd S \rightleftharpoons \alpha.(p \lhd S) \tag{22}$$

The *parallel* rule (16) says, that by heating ($\rightharpoonup$) a molecule of the form $p_1|p_2$ it breaks up into its components p_1 and p_2 . Conversely, by cooling down ($\leftharpoondown$) a pair of molecules p_1, p_2 the compound molecule $p_1|p_2$ can be rebuild. Only *ions*, which are molecules of the form $\alpha.p$, where α represents the valence of the ion, are allowed to perform reactions according to the *reaction* rule (17). A reaction is an irreversible transformation in which two complementary ions, floating in a solution, react with each other, whereby they release their bodies and the valences vanish. The *inaction cleanup* rule (18) simply states that 0 evaporates when heated. The *restriction ion* rule (19) and the *restriction membrane* rule (20) are special heating and cooling rules for restricted molecules. In the *airlock* rule (21) the airlock mechanism is specified and the *heavy ion* rule (22) denotes how to build an ion out of a molecule in an airlock.

4 The PCHAM — A CHAM for Deductive Planning

In this section we are going to transfer the functionality of the equational logic program in Sect. 2 (hereafter referred to as program P) to the machine model of the previously introduced CHAM. Before doing so we need to modify the CHAM slightly. Thereafter, we will show that the modified machine, which we will call PCHAM, can handle the planning problems presented in Sect. 2. In other words, we show that the PCHAM is an appropriate abstract machine model for reasoning about situations, actions, and causality.

4.1 PCHAM

Let $\mathcal{N}$ be the set of terms denoting the facts as well as the conditions and effects of actions in P, and let α range over the set $\mathcal{V} = \{v^+, v^- : v \in \mathcal{N}\}$. Adding the symbol "!" (exponential) to the above set of operators, PCHAM molecules have the following form, where S denotes a multiset of molecules.

$$m ::= \{\!|p|\!\} \mid !\{\!|p|\!\} \mid \{\!|p|\!\}\backslash^+ \mid \{\!|p|\!\}\backslash^- \mid \alpha.0 \lhd \{\!|p|\!\}\backslash^+ \mid \alpha.0 \lhd \{\!|p|\!\}\backslash^- \mid S \mid \{\!|p|\!\} \lhd S$$
$$\text{where } p ::= \alpha.0 \mid (p_1|p_2) \mid 0$$

We can adopt (16) directly; the reaction rule used by the PCHAM (23) is slightly different from (17) in that only ions[11] attached to a restricted airlock may perform reactions and furthermore two such ions may react if they are unifiable. The positive restriction operator ($^+$) ensures, that no positive valence (v^+) can be extracted from the encapsulated solution, likewise the negative restriction operator ($^-$) does not allow negative valences (v^-) to be extracted from the encapsulated solution.

$$\{\!|v^- \lhd \{\!|p|\!\}\backslash^+, v^+ \lhd \{\!|p|\!\}\backslash^-|\!\} \dot\cup S \;\rightarrow\; \sigma(\{\!|\{\!|p|\!\}\backslash^+, \{\!|p|\!\}\backslash^-|\!\} \dot\cup S) \tag{23}$$

where σ is the most general unifier (mgu) of v^- and v^+. One should observe that the mgu of two terms itself can be computed by multiset rewritings (cf. [11]).

We slightly changed the airlock mechanism of the CHAM to fit our purposes. In the PCHAM it applies only to unrestricted membranes. The following two rules are called *P–airlock* and *disconnect* rule.

$$\{\!|\{\!|p|\!\}, m_1, m_2, \ldots, m_k|\!\} \rightleftharpoons \{\!|p|\!\} \lhd \{\!|m_1, m_2, \ldots, m_k|\!\} \tag{24}$$
$$\{\!|p|\!\} \lhd \{\!|m_1, m_2, \ldots, m_k|\!\} \rightleftharpoons \{\!|p|\!\}, \{\!|m_1, m_2, \ldots, m_k|\!\} \tag{25}$$

Instead of (22) there is a special mechanism necessary, called the *P–ion* rule, which extracts ions from a solution encapsulated in a restricted membrane.

$$\{\!|v^-, \alpha_1, \alpha_2, \ldots, \alpha_l|\!\}\backslash^+ \rightleftharpoons v^- \lhd \{\!|\alpha_1, \alpha_2, \ldots, \alpha_l|\!\}\backslash^+ \tag{26}$$
$$\{\!|v^+, \alpha_1, \alpha_2, \ldots, \alpha_l|\!\}\backslash^- \rightleftharpoons v^+ \lhd \{\!|\alpha_1, \alpha_2, \ldots, \alpha_l|\!\}\backslash^-$$

Next we need two rules, the first for encapsulating solutions into restricted membranes, called the *add restrict* rule (27), and the second for removing restricted membranes again, called the *remove restrict* rule (28). A restricted membrane is a prerequisite for a reaction. Only pairs of molecules can be encapsulated in restricted membranes. Furthermore, to force a reaction to occur locally the two molecules become encapsulated in an additional membrane.

$$\{\!|p|\!\}, \{\!|q|\!\} \rightharpoonup \{\!|\{\!|p|\!\}\backslash^+, \{\!|q|\!\}\backslash^-|\!\} \tag{27}$$
$$\{\!|\{\!|p|\!\}\backslash^+, \{\!|q|\!\}\backslash^-|\!\} \rightharpoonup \{\!|p, q|\!\} \tag{28}$$

[11] Because an PCHAM ion always has ' 0 ' in its body, we will denote it by α for notational convenience.

Finally, we need an additional rule, called the *generator* rule which handles the exponential "!".

$$!\{\mathcal{C}|\mathcal{E}\} \rightarrow \{\mathcal{C}'|\mathcal{E}'\}, !\{\mathcal{C}|\mathcal{E}\} \tag{29}$$

where $\mathcal{C}'$ and $\mathcal{E}'$ are standardized apart.

4.2 Mapping the Equational Logic Program onto the PCHAM

In this subsection we want to show how SLDENF–refutations can be transformed into PCHAM–rewritings. In particular, we will show that if there is an SLDENF–refutation from the goal $\leftarrow \mathit{transform}(I, G)$ wrt. the program P with answer substitution θ, then there is a sequence of PCHAM rewriting steps from $\mathcal{T}(I)$ to some G' such that G' and $\mathcal{T}(\theta G)$ are AC1–unifiable, where $\mathcal{T}$ transforms ELP–situations into PCHAM–situations.

$$\begin{array}{rlrl}
\mathcal{T}(\emptyset) &= \{\} & \mathcal{T}^+(\emptyset) &= 0 \\
\mathcal{T}(S_1 \circ S_2) &= \mathcal{T}(S_1) \dot{\cup} \mathcal{T}(S_2) & \mathcal{T}^+(v) &= v^+ \\
\mathcal{T}(\langle \mathcal{C}.\mathcal{E} \rangle) &= \{\mathcal{T}^+(\mathcal{C})|\mathcal{T}^-(\mathcal{E})\} & \mathcal{T}^+(v_1 \circ v_2) &= \mathcal{T}^+(v_1)|\mathcal{T}^+(v_2) \\
\mathcal{T}(!\langle \mathcal{C}.\mathcal{E} \rangle) &= !\ \mathcal{T}(\langle \mathcal{C}.\mathcal{E} \rangle) & \mathcal{T}^-(\emptyset) &= 0 \\
\mathcal{T}(\boxed{S}) &= \{\mathcal{T}(S)\} & \mathcal{T}^-(v) &= v^- \\
\mathcal{T}(M \lhd S) &= \mathcal{T}(M) \lhd \mathcal{T}(S) & \mathcal{T}^-(v_1 \circ v_2) &= \mathcal{T}^-(v_1)|\mathcal{T}^-(v_2)
\end{array}$$

Having specified molecules and solutions of our PCHAM we can now turn our attention to the mapping of SLDENF–derivations onto PCHAM–rewritings. Our goal is to give a sequence of PCHAM–rewriting steps for each application of SLDENF–resolution using one of the clauses (3)–(8). As we will demonstrate, the PCHAM–rewriting steps will consider only that part of a situation which changes, but not the context. Hence, PCHAM–rewritings solve the frame problem in a quite natural way.

The first clause (3) defines how a reaction between two agents in a solution takes place. The corresponding sequence of PCHAM-transformation rules[12] is stated below, where we have abbreviated $c^+_{1,1}, \ldots, c^+_{1,k}$ by $\overline{c}$ and $e^-_{2,1}, \ldots, e^-_{2,n}$ by $\overline{e}$, and have also indicated the corresponding rule number in brackets.

$$\begin{array}{lll}
 & \{c^+_{1,1}|\ldots|c^+_{1,k}|e^-_{1,1}|\ldots|e^-_{1,l}\}, \{c^+_{2,1}|\ldots|c^+_{2,m}|e^-_{2,1}|\ldots|e^-_{2,n}\} & \text{see (16)} \\
\stackrel{*}{\rightharpoonup} & \{\overline{c}, e^-_{1,1}, \ldots, e^-_{1,l}\}, \{c^+_{2,1}, \ldots, c^+_{2,m}, \overline{e}\} & \text{see (27)} \\
\rightharpoonup & \{\{\overline{c}, e^-_{1,1}, \ldots, e^-_{1,l}\}\backslash^+, \{c^+_{2,1}, \ldots, c^+_{2,m}, \overline{e}\}\backslash^-\} & \text{see (26)} \\
\stackrel{2}{\rightharpoonup} & \{e^-_{1,1} \lhd \{\overline{c}, e^-_{1,2}, \ldots, e^-_{1,l}\}\backslash^+, c^+_{2,1} \lhd \{c^+_{2,2}, \ldots, c^+_{2,m}, \overline{e}\}\backslash^-\} & \text{see (23)} \\
\rightarrow & \sigma_1(\{\{\overline{c}, e^-_{1,2}, \ldots, e^-_{1,l}\}\backslash^+, \{c^+_{2,2}, \ldots, c^+_{2,m}, \overline{e}\}\backslash^-\}) & \\
\stackrel{*}{\rightarrow} & \sigma_d \ldots \sigma_1(\{\{\overline{c}, e^-_{1,d+1}, \ldots, e^-_{1,l}\}\backslash^+, \{c^+_{2,d+1}, \ldots, c^+_{2,m}, \overline{e}\}\backslash^-\}) & \text{see (28)} \\
\rightharpoonup & \sigma_d \ldots \sigma_1(\{\overline{c}, e^-_{1,d+1}, \ldots, e^-_{1,l}, c^+_{2,d+1}, \ldots, c^+_{2,m}, \overline{e}\}) & \text{see (16)} \\
\stackrel{*}{\rightharpoonup} & \sigma_d \ldots \sigma_1(\{c^+_{1,1}|\ldots|c^+_{1,k}|e^-_{1,d+1}|\ldots|e^-_{1,l}|c^+_{2,d+1}|\ldots|c^+_{2,m}|e^-_{2,1}|\ldots|e^-_{2,n}\}) &
\end{array}$$

[12] By $\stackrel{n}{\rightarrow}$ we denote that the corresponding interaction rule is applied n times.

One should observe that in the step where (26) is used, any molecule with a positive or negative valence can be pushed through the membrane with negative or positive restriction, respectively.

In the sequel we want to abbreviate the above sequence of rules by the following macro–rule.

$$\begin{aligned} &\{c^+_{1,1}|\ldots|c^+_{1,k}|e^-_{1,1}|\ldots|e^-_{1,l}\}, \{c^+_{2,1}|\ldots|c^+_{2,m}|e^-_{2,1}|\ldots|e^-_{2,n}\} \xrightarrow{M} \\ &\sigma(\{c^+_{1,1}|\ldots|c^+_{1,k}|e^-_{1,d+1}|\ldots|e^-_{1,l}|c^+_{2,d+1}|\ldots|c^+_{2,m}|e^-_{2,1}|\ldots|e^-_{2,n}\}) \end{aligned} \tag{30}$$

where $1 \leq d \leq \min(l, m)$ denotes the number of valences vanishing in the reaction, and σ is the iteratively produced mgu during the performance of these reactions. It should be noted that the only need of the negation as failure part within an SLDENF–refutation corresponds to the fact that within the macro sequence at least one application of (23) has to take place.

The clause (4) is used to generate new agents. The corresponding interaction rule, viz. (29) mimics this clause. As the chemical law (13) and the membrane law (14) require that interactions can be performed freely within any solution and context, these laws take care of the clauses (5) and (6). Finally, the remaining clauses (7) and (8) are handled by the modified airlock rule (24) and the disconnect rule (25) of the PCHAM.

Theorem 2. *If there is an SLDENF–refutation of* transform(I, G) *wrt.* P *with computed answer substitution* θ *then there is PCHAM–rewriting from* $\mathcal{T}(I)$ *to some* G' *such that* G' *and* $\mathcal{T}(\theta G)$ *are AC1–unifiable.*

The theorem is proved by induction on the length of the SLDENF–refutation using the correspondence between SLDENF–resolution steps and PCHAM–rewriting steps mentioned above. Due to lack of space we will not give this proof in detail, they can be found in [4]. Rather we depict the PCHAM–rewriting sequences for the example discussed in Sect. 2. The first sequence corresponds to the SLDENF-derivation of the goal (9). We have underlined the interacting molecules.

$$\begin{aligned} &\{\ \{0|d^-\},\ !\underline{\{d^+|l^-\}},\ !\underline{\{l^+|on^-|l^-\}}\ \} && \text{see (29)} \\ \xrightarrow{2}\ &\{\ \underline{\{0|d^-\}},\ \underline{\{d^+|l^-\}},\ \{l^+|on^-|l^-\},\ !\{d^+|l^-\},\ !\{l^+|on^-|l^-\}\ \} \\ \xrightarrow{M}\ &\{\ \underline{\{0|l^-\}},\ \underline{\{l^+|on^-|l^-\}},\ !\{d^+|l^-\},\ !\{l^+|on^-|l^-\}\ \} \\ \xrightarrow{M}\ &\{\ \{0|on^-|l^-\},\ !\{d^+|l^-\},\ !\{l^+|on^-|l^-\}\ \} \end{aligned}$$

It is easy to see that the final state of the PCHAM corresponds to the instantiated goal state of (9) modulo AC1.

The second sequence corresponds to the SLDENF–refutation of (10), except that we have modeled the action *enter* as an agent and have applied it first. It shows how membranes can be used to model the lower levels of the planning hierarchy. Let T denote the unlimited source of actions at the lower level, i.e.

$$T = !\{l^+|n^-|l^-\},\ !\{u^+|n^+|in^-|n^-\},\ !\{in^+|n^+|on^-|in^-|n^-\}.$$

$$
\begin{array}{ll}
 & \{\ \underline{\{0|d^-\}},\ \underline{\{d^+|l^-\}},\ \{\ \{0|u^-\},\ T\}\ \} \\
\xrightarrow{M} & \{\ \{0|l^-\},\ \{\ \{0|u^-\},\ \underline{T}\}\ \} \qquad \text{see (29)} \\
\xrightarrow{3} & \{\ \{0|l^-\},\ \{\ \underline{\{0|u^-\}},\ \{l^+|n^-|l^-\},\ \underline{\{u^+|n^+|in^-|n^-\}}, \\
 & \{in^+|n^+|on^-|in^-|n^-\},\ T\}\ \} \\
\xrightarrow{M} & \{\ \{0|l^-\},\ \{\ \underline{\{l^+|n^-|l^-\}},\ \underline{\{n^+|in^-|n^-\}},\ \{in^+|n^+|on^-|in^-|n^-\},\ T\}\ \} \\
\xrightarrow{M} & \{\ \{0|l^-\},\ \{\ \underline{\{l^+|in^-|l^-|n^-\}},\ \underline{\{in^+|n^+|on^-|in^-|n^-\}},\ T\}\ \} \\
\xrightarrow{M} & \{\ \{0|l^-\},\ \{\ \underline{\{l^+|on^-|in^-|n^-|l^-\}},\ T\}\ \} \qquad \text{see (24)} \\
\rightarrow & \{\ \{0|l^-\},\ \underline{\{l^+|on^-|in^-|n^-|l^-\}} \lhd \{\ T\}\ \} \qquad \text{see (25)} \\
\rightarrow & \{\ \underline{\{0|l^-\}},\ \underline{\{l^+|on^-|in^-|n^-|l^-\}},\ \{\ T\}\ \} \\
\xrightarrow{M} & \{\ \{\ 0|on^-|in^-|n^-|l^-\},\ \{\ T\}\ \}
\end{array}
$$

5 Conclusion and Future Work

During the last years we have seen several new approaches for reasoning about situations, actions, and causality which solve the frame problem without the need to state frame axioms or similar laws of inertia. In contrast to conventional approaches like the situation calculus, where facts about a situation are modeled as properties, these new approaches model facts about a situation as resources.

In this paper we achieved the following results. First of all we demonstrated that hierarchical planning can be achieved within the mentioned new approaches if actions are modeled as resources. This was achieved by encapsulating simpler actions into a membrane in order to let these actions react to form more complex actions and to push the complex actions through the membrane to the next higher level. We have also introduced generators to model actions at the lowest level of the hierarchy.

As the concept of a membrane was borrowed from the CHAM, the next question to answer was how the hierarchical planning approach was related to the CHAM. In answering this question we developed a new abstract machine, called PCHAM, which extends the CHAM. The main feature of this extension is the incorporation of logical variables into the CHAM by means of the P–reaction rule. This is a quite natural extension of the CHAM as the most general unifier of two expressions can be computed by multiset rewritings. Finally, we showed that the PCHAM can serve as an abstract machine for the kind of reasoning problems considered in this paper.

The abstract machine model developed in this paper opens a new line of research. In the rewriting sequences presented in this paper only two molecules were allowed to react at a given time. But there is absolutely no reason why not more than a single pair of molecules may interact at a given time as long as the various interactions do not interfere. For example, in the final rewriting sequence of the previous section the first four rewriting steps ($\xrightarrow{M}, \xrightarrow{3}$) could have been

performed in parallel. In fact, the CHAM was devised as a machine model for asynchronous concurrent computations. We envision that in a larger world model many local interactions may take place at a time and that the overall global goal is achieved by means of these local interactions. Thus, a future planning system based on the PCHAM may very much look like a connectionist network.

Acknowledgement: We like to thank Sven-Erik Bornscheuer, who in many discussion forced us to straighten out our ideas. Thanks also to Wolfgang Bibel.

References

1. G. Berry and G. Boudol. The chemical abstract machine. In *Proceedings of the ACM Symposium on Principles of Programming Languages*, pages 81–94, 1990.
2. W. Bibel. A deductive solution for plan generation. *New Generation Computing*, 4:115–132, 1986.
3. A. de Waal and M. Thielscher. Solving deductive planning problems using program analysis and transformation. In M. Proietti, editor, *Proceedings of the International Workshop on Logic Program Synthesis and Transformation (LOPSTR)*. Springer, September 1995.
4. K. Eder. A resource-oriented deductive approach towards hierarchical planning. Diplomarbeit, Technische Universität Dresden, Fakultät Informatik, May 1995.
5. G. Große, S. Hölldobler, and J. Schneeberger. Linear deductive planning. *Journal of Logic and Computation*, 1996. (To appear).
6. J. Y. Girard. Linear logic. *Journal of Theoretical Computer Science*, 50(1):1 – 102, 1987.
7. S. Hölldobler and J. Schneeberger. A new deductive approach to planning. *New Generation Computing*, 8:225–244, 1990. A short version appeared in the Proceedings of the German Workshop on Artificial Intelligence, Informatik Fachberichte *216*, pages 63-73, 1989.
8. S. Hölldobler and M. Thielscher. Computing change and specificity with equational logic programs. *Annals of Mathematics and Artificial Intelligence*, 14(1):99–133, 1995.
9. J. McCarthy. Situations and actions and causal laws. Stanford Artificial Intelligence Project: Memo 2, 1963.
10. J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of Artificial Intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463 – 502. Edinburgh University Press, 1969.
11. A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4:258–282, 1982.
12. M. Masseron, C. Tollu, and J. Vauzielles. Generating plans in linear logic. In *Foundations of Software Technology and Theoretical Computer Science*, pages 63–75. Springer, LNCS *472*, 1990.
13. R. Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation — Papers in Honor of John McCarthy*, pages 359–380. Academic Press, 1991.
14. E. D. Sacerdoti. Planning in a hierarchy of abstraction spaces. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 412–422, 1973.

15. J. C. Shepherdson. SLDNF-resolution with equality. *Journal of Automated Reasoning*, 8:297–306, 1992.
16. M. Thielscher. On the completeness of SLDENF-resolution. *Journal of Automated Reasoning*, 1996. (To appear Fall '96).
17. D. H. D. Warren. An abstract Prolog instruction set. Technical Report 306, SRI International, 1983.