A Systematic Solution to the (De-)Composition Problem in General Game Playing

Timothy Cerexhe and **David Rajaratnam** and **Abdallah Saffidine** and **Michael Thielscher** [1]

Abstract. General game players can drastically reduce the cost of search if they are able to solve smaller subproblems individually and synthesise the resulting solutions. To provide a systematic solution to this (de-)composition problem, we start off with generalising the standard decomposition problem in planning by allowing the composition of individual solutions to be further constrained by domain-dependent requirements of the global planning problem. We solve this generalised problem based on a systematic analysis of composition operators for transition systems, and we demonstrate how this solution can be further generalised to general game playing.

1 INTRODUCTION

General Game Playing (GGP) aims at creating AI systems that can understand the rules of new games and then learn to play them without human intervention. Fostered by the annual AAAI GGP competition since 2005, the field has emerged in direct response to specialised systems that use highly specific algorithms to play only a single type of game. In contrast, a GGP system must autonomously adapt to new and possibly radically different problems. Research into GGP can thus be viewed as part of a broader research agenda to build systems that exhibit forms of general intelligence [8].

A general game-playing system cannot be endowed with game-specific algorithms in advance. A key objective of research into GGP, therefore, is to develop methods for automatically analysing the rules of a game in order to find structures that help players to construct an efficient search strategy at runtime [8]. To emphasise this, the AAAI competition has recently focused on games with an internal structure that, if recognised, can be utilised to decompose, and hence drastically reduce, the search space [11].

Despite the recognition of the importance of decomposition in GGP, competition systems have so far had very limited success in dealing with such games [11]. Unfortunately, this is also reflected in the extremely sparse nature of the research coverage of this topic. Firstly, based on the encoding of games as *propositional automata*, Cox *et al.* [5] provide theoretical conditions under which a global game can be decomposed into a conjunction of multiple sub-games. Secondly, Günther *et al.* [12] provide an approach that is based on the construction of a *dependency graph* of action and fluent predicates for single player games, such that disconnected sub-graphs identify independent sub-games. Finally, Zhao *et al.* [18] extend the dependency graph approach to the multi-player case.

The apparent lack of effective application to GGP systems despite these advances is the result of one key failure, namely, the lack of a strong account of how local sub-game solutions can be combined into global game solutions. We refer to this as the *composition problem*. While previously identified [12, 18], nevertheless, current approaches have only been able to deal with it in an ad-hoc algorithmic manner without providing any theoretical foundations on which to understand the properties and behaviour of these algorithms.

The composition problem is particularly challenging in GGP due to the separation of goal and termination conditions, making the satisfaction of global goal conditions highly sensitive to the execution order of sub-game actions. For example, satisfying the goal of one sub-game before another may cause the premature termination of the global game. In fact, it is worth noting that the separation of goal and terminal conditions is one of the key features that distinguishes GGP from AI planning and makes GGP a more general and difficult problem. We shall return to this relationship in the concluding Section 5, where we discuss the potential application of our decomposition approach to the problem of factored planning [1].

In this paper we address the composition problem in GGP by developing a systematic approach based on model checking products of Transition Systems (TSs). Our main contributions are:

- The reduction of the model checking problem of global TSs to the model checking of their composed parts.
- The worst-case complexity analysis of standard model checking algorithms when applied to decomposed problems, establishing the theoretical advantages of our approach.
- An experimental evaluation with games from past GGP competitions highlighting potential (orders of magnitude) performance gains of the approach.

It is worth emphasising that the task of identifying and decomposing games is not within the scope of this paper. Rather we are concerned with the theoretical foundations of sub-game composition. Fortunately, existing techniques for sub-game identification [12, 17, 18] can be applied without jeopardising results about the soundness of the transition systems themselves, and these form the basis for our experimental results.

The remainder of this paper proceeds as follows. Section 2 provides the main theoretical contribution, whereby a set of TS composition operators are defined and the notion of a *stability condition* is developed. Section 3 presents the complexity analysis for solving decomposition problems using common algorithms, highlighting the advantage of the theory to common special cases. Section 4 provides an Answer Set Programming based implementation of the theory showing its application to GGP for solving single-player games and proving desirable game properties. Finally, in Section 5 we summarise and discuss our results in the broader context of related fields and outline possible directions for future research.

[1] School of Computer Science and Engineering, The University of New South Wales, Australia, email:{timothyc,daver,abdallahs,mit}@cse.unsw.edu.au

2 COMPOSITION OF TRANSITION SYSTEMS

Partial order reduction, a major breakthrough in the software verification community, allows efficient model checking of "*next-free LTL*" formulas on asynchronous products of Transition Systems [2]. We draw inspiration from the verification community and share the TS formalism, but our target application has different assumptions: in verification, systems typically do not terminate or consider timesteps, whereas in GGP, local games terminate and interactions between server and agents constitute timesteps. We thus focus on another class of specifications, called *stability conditions*. Our approach handles specific time steps as well as sequential and synchronous products, while partial order reduction allows for nested "*until*" operators.

First we recall the definition of TSs and the composition operators on them. Next we define *stability conditions* to formally express queries on TSs. Finally, we show how these conditions can be decomposed, by translating model checking problems on products of TSs to sets of model checking problems on the factors.

By way of motivation, and to more clearly illustrate the theory, we consider the game of *Incredible*. This game has been used as the key example in discussions of decomposition in GGP [12].

Example Incredible *is a single-player game that combines three underlying sub-games: the well-known* blocks world *(or* `blocks`*) construction game, a* `maze` *game requiring the player to carry a piece of gold from an initial position to a home position, and a* `wait` *game consisting of a set of superfluous transitions. The player earns points for solving the blocks world and maze sub-games and has to perform these task within 20 steps. Importantly, the game is terminated immediately on the completion of the maze sub-game.*

There are a number of interesting aspects of decomposition that are highlighted by this game. In the first place, the `wait` sub-game is redundant if decomposed correctly, but dramatically increases the search space if this fact remains unrecognised. Secondly, while the sub-games can be solved independently there are subtleties of termination that impose restrictions on how their solutions should be combined. For example, the early termination condition of the `maze` sub-game means that in order to maximise the final score a player should make the last step of the `maze` only after the `blocks` sub-game has been completed.

Composition Operators

We now formalise the precise style of Transition Systems (TSs) on which our theory operates and introduce the composition operators that can be used to combine these TSs. For further details on the precise notion used to describe TSs we refer to Baier and Katoen [2].

Definition 1 *A* Transition System (TS) *is a tuple* $\mathcal{T} = \langle \Sigma, \rightarrow, P, \lambda \rangle$ *where:* Σ *is a set of* states*;* $\rightarrow \subseteq \Sigma \times \Sigma$ *is a transition relation;* P *is a set of atomic propositions; and* $\lambda : \Sigma \rightarrow 2^P$ *is a labelling function.*

We are now ready to introduce our three composition operators—synchronous, asynchronous, and sequential. The first, synchronous case represents two systems proceeding in lockstep, for example an array of coordinated traffic lights.

Definition 2 *The* synchronous composition *of two TSs* $\mathcal{T}_1$ *and* $\mathcal{T}_2$ *is a new TS* $\mathcal{T}_1 \parallel \mathcal{T}_2 = \langle \Sigma_1 \times \Sigma_2, \rightarrow, P_1 \amalg P_2, \lambda \rangle$ *with:*[2]

- $\langle s_1, s_2 \rangle \rightarrow \langle s_1', s_2' \rangle \iff s_1 \rightarrow_1 s_1' \wedge s_2 \rightarrow_2 s_2'$
- $\lambda(\langle s_1, s_2 \rangle) = \lambda_1(s_1) \amalg \lambda_2(s_2)$

[2] We use $\amalg$ to denote the disjoint union.

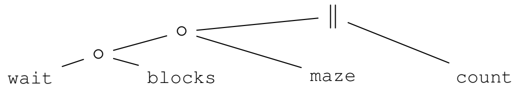

Figure 1. Composed Transition System for *Incredible*

With asynchronous composition the lockstep restriction is removed and the two systems progress completely independently. This interleaving can model multi-threaded programs on a uniprocessor.

Definition 3 *The* asynchronous composition *of two TSs* $\mathcal{T}_1$ *and* $\mathcal{T}_2$ *is a new TS* $\mathcal{T}_1 \circ \mathcal{T}_2 = \langle \Sigma_1 \times \Sigma_2, \rightarrow, P_1 \amalg P_2, \lambda \rangle$ *with:*

- $\langle s_1, s_2 \rangle \rightarrow \langle s_1', s_2 \rangle \iff s_1 \rightarrow_1 s_1'$
- $\langle s_1, s_2 \rangle \rightarrow \langle s_1, s_2' \rangle \iff s_2 \rightarrow_2 s_2'$
- $\lambda(\langle s_1, s_2 \rangle) = \lambda_1(s_1) \amalg \lambda_2(s_2)$

The third form of synchronisation is sequential. Transitions in the second system can only occur after the first system has reached a terminal state. This case is useful for modelling phase changes.

Definition 4 *The* sequential composition *of two TSs* $\mathcal{T}_1$ *and* $\mathcal{T}_2$ *is a new TS* $\mathcal{T}_1 ; \mathcal{T}_2 = \langle \Sigma_1 \times \Sigma_2, \rightarrow, P_1 \amalg P_2, \lambda \rangle$ *with:*

- $\langle s_1, s_2 \rangle \rightarrow \langle s_1', s_2 \rangle \iff s_1 \rightarrow_1 s_1'$
- $\langle s_1, s_2 \rangle \rightarrow \langle s_1, s_2' \rangle \iff \neg \exists s'. \, s_1 \rightarrow_1 s' \wedge s_2 \rightarrow_2 s_2'$
- $\lambda(\langle s_1, s_2 \rangle) = \lambda_1(s_1) \amalg \lambda_2(s_2)$

It is easy to prove that these operators are all associative, so they naturally generalise beyond the binary composition case. The synchronous and asynchronous operators are also commutative modulo isomorphism so the ordering of multiple similar compositions is unimportant. The sequential composition, however, is not commutative since the construction order introduces an implicit dependence.

Example *Using the defined composition operators we can now formally express the* Incredible *TS in terms of atomic TSs (Figure 1):* `incredible` $=$ $\big((\texttt{wait} \circ \texttt{blocks}) \circ \texttt{maze}\big) \parallel$ `count` *The* `wait` *TS corresponds to the sub-game containing the superfluous transitions,* `blocks` *encodes the blocks world puzzle, and* `maze` *encodes the gold delivery task. Additionally a* `count` *TS encodes the requirements of the game's step counter. This TS is combined synchronously to ensure that all sub-games adhere to the same counter.*

Stability Conditions

While TSs are a natural modelling analogue for many domains, the *application* of these systems can vary wildly. Planning systems typically want to find a path to a labeled goal state. Verification tasks are often the dual—the non-existence of a path with undesired effects. Game players seek a path to a labeled goal state that cannot be blocked by an opposing agent.

A natural mechanism for generalising these different use cases is to consult a domain-specific "stability condition". This condition is a (possibly infinite) sequence of formulas that constrains acceptable trajectories through the corresponding TS. We now provide a precise formalisation of these intuitive concepts.

Definition 5 *A* stability condition Φ *is a sequence of propositional formulas:* $\Phi = (\phi_n)_{0 \leq n < N}$ *with* $N \in \mathbb{N} \cup \{\infty\}$*. It is* conjunctive *if every formula is conjunctive. The* length *of* Φ*,* N*, is also written* $|\Phi|$*.*

Before considering stability conditions in a TS $\langle \Sigma, \rightarrow, P, \lambda \rangle$, we recall that a state $s \in \Sigma$ *satisfies* a propositional formula ϕ ranging over P, written $s \models \phi$, if $\lambda(s) \models \phi$ where the satisfaction of ϕ by a set of atomic propositions is defined as usual.

Definition 6 *A state s_0 in a system $\mathcal{T}$* satisfies *a stability condition $\Phi = (\phi_n)_{0 \le n < N}$, written $\mathcal{T}, s \models \Phi$, if there is a sequence of N states $s_0 \rightarrow s_1 \rightarrow s_2 \dots$, such that $\forall 0 \le n < N$, $\mathcal{T}, s_n \models \phi_n$.*

We may omit the TS or the initial state when it is obvious from the context and simply write $s \models \Phi$ or $\mathcal{T} \models \Phi$.

The GGP context helps to clarify the concept of stability conditions. In particular, a stability condition to solve a game constrains the trajectories through that game's TS such that all intermediate states in the trajectory are non-terminal while the final state is both terminal and goal satisfying.

Example *The termination condition of Incredible is satisfied when the gold is dropped at the home destination or after a timeout of twenty steps. The goal condition is to have constructed two specific towers and retrieved the gold. Hence, the stability condition Γ is a finite sequence $\langle \varphi, \dots, \varphi, \psi \rangle$ where $\varphi = \neg(\texttt{gold} \vee \texttt{timeout})$ and $\psi = (\texttt{gold} \vee \texttt{timeout}) \wedge (\texttt{gold} \wedge \texttt{towers})$*

We now establish properties of how the stability conditions of a composed TS relate to the stability conditions of its components. This is the crucial element if sub-game solutions are to be combined to provide global game solutions.

Theorem 1 *Let Φ be a stability condition. There exists a set $\mathcal{D}(\Phi)$ of conjunctive stability conditions, such that for any TS and state s, $s \models \Phi$ if and only if $\exists \Psi \in \mathcal{D}(\Phi)$ such that $s \models \Psi$.*

If Φ is of finite length, then we have the bound $|\mathcal{D}(\Phi)| \le K^{|\Phi|}$, where K is the maximum size of a disjunctive normal form in Φ, and for any $\Psi \in \mathcal{D}(\Phi)$, $|\Psi| = |\Phi|$.

Proof: Assume that $\Phi = (\phi_n)_{0 \le n < N}$. For every n, construct the Disjunctive Normal Form (DNF) of ϕ_n: $\phi_n \equiv \bigvee_{0 \le i_n < K_n} \phi_{n,i_n}$ where ϕ_{n,i_n} is a conjunctive formula, for all i_n. Let $\mathbb{K} = \mathbb{Z}_{K_0} \times \mathbb{Z}_{K_1} \times \dots \mathbb{Z}_{K_n} \times \dots$ where $\mathbb{Z}_k$ is the set $\{0, 1, \dots, k-1\}$. For any $i = \langle i_0, \dots, i_n, \dots \rangle \in \mathbb{K}$, we define $\Phi(i) = (\phi_{n,i_n})_{0 \le n < N}$. Let $\mathcal{D}(\Phi)$ be defined as $\{\Phi(i), i \in \mathbb{K}\}$.

Let s_0 be a state of a TS $\mathcal{T}$. Then $s_0 \models \Phi$ iff $(\exists s_0 \rightarrow s_1 \rightarrow \dots$ s.t. $\forall 0 \le n < N, s_n \models \phi_n)$ iff $(\exists s_0 \rightarrow s_1 \rightarrow \dots$ s.t. $\forall 0 \le n < N$, $\exists i_n, s_n \models \phi_{n,i_n})$ iff $(\exists i \in \mathbb{K}$ s.t. $s_0 \models \Phi(i))$. □

Example *Putting the formulas in our running example into DNF, we obtain that* $\texttt{incredible} \models \langle \varphi, \dots, \varphi, \psi \rangle$ *iff* $\texttt{incredible} \models \langle \phi_1, \dots, \phi_1, \phi_2 \rangle$ *or* $\texttt{incredible} \models \langle \phi_1, \dots, \phi_1, \phi_3 \rangle$*, where φ, ψ are as above and $\phi_1 = \neg\texttt{gold} \wedge \neg\texttt{timeout}$, $\phi_2 = \texttt{gold} \wedge \texttt{towers}$, and $\phi_3 = \texttt{timeout} \wedge \texttt{gold} \wedge \texttt{towers}$. Note that we* did not *simplify ψ or prune ϕ_3 for the sake of exposition.*

We can now focus on solving the model checking problem for conjunctive stability conditions. Unless mentioned otherwise, the stability conditions in the rest of this section will always be assumed to be conjunctive.

Theorem 2 *Let Φ be a stability condition over $P_1 \amalg P_2$. There exists a pair of stability conditions, $\langle \Phi_1, \Phi_2 \rangle$, such that for any* synchronous *composition $\mathcal{T}_1 \parallel \mathcal{T}_2 = \mathcal{T}$, and for any state $\langle s_1, s_2 \rangle$ of $\mathcal{T}$, we have that $\langle s_1, s_2 \rangle \models \Phi$ iff $s_1 \models \Phi_1$ and $s_2 \models \Phi_2$.*

If Φ is of finite length, then $|\Phi_1| = |\Phi_2| = |\Phi|$.

Proof: For all n, let $\varphi_{j,n}$ be the projection of ϕ_n over the atoms in P_j: $\phi_n = \varphi_{1,n} \wedge \varphi_{2,n}$. We define $\Phi_j = (\varphi_{j,n})_{0 \le n < N}$, for $j \in \{1, 2\}$.

$\langle s_1, s_2 \rangle \models \Phi$ iff $(\exists \langle s_1^0, s_2^0 \rangle \rightarrow \langle s_1^1, s_2^1 \rangle \rightarrow \dots$ s.t. $\langle s_1, s_2 \rangle = \langle s_1^0, s_2^0 \rangle$ and $\forall 0 \le n < N, \langle s_1^n, s_2^n \rangle \models \phi_n)$ iff $(\exists s_1^0 \rightarrow s_1^1 \rightarrow \dots$ s.t. $\forall 0 \le n < N$, $s_1^n \models \varphi_{1,n}$ and $\exists s_2^0 \rightarrow s_2^1 \rightarrow \dots$ s.t. $\forall 0 \le n < N$, $s_2^n \models \varphi_{2,n})$ iff $(s_1 \models \Phi_1$ and $s_2 \models \Phi_2)$. □

Example *Applying Theorem 2 to the $\langle \phi_1, \dots, \phi_1, \phi_2 \rangle$ condition in* Incredible *gives that* $\texttt{incredible} \models \langle \phi_1, \dots, \phi_1, \phi_2 \rangle$ *iff* $\texttt{wait} \circ \texttt{blocks} \circ \texttt{maze} \models \langle \neg\texttt{gold}, \dots, \neg\texttt{gold}, \texttt{gold} \wedge \texttt{towers} \rangle$ *and* $\texttt{count} \models \langle \neg\texttt{timeout}, \dots, \neg\texttt{timeout}, \top \rangle$.

Theorem 3 *Let Φ be a stability condition over $P_1 \amalg P_2$ then there exists a set $\mathcal{A}(\Phi)$ of pairs of stability conditions, such that for any* asynchronous *composition $\mathcal{T}_1 \circ \mathcal{T}_2 = \mathcal{T}$, and for any state $\langle s_1, s_2 \rangle$ of $\mathcal{T}$, we have that $\mathcal{T}, \langle s_1, s_2 \rangle \models \Phi$ iff $\exists \langle \Phi_1, \Phi_2 \rangle \in \mathcal{A}(\Phi)$ such that $\mathcal{T}_1, s_1 \models \Phi_1$ and $\mathcal{T}_2, s_2 \models \Phi_2$.*

If Φ is of positive finite length, then we have the bound $|\mathcal{A}(\Phi)| \le 2^{|\Phi|-1}$, and for any $\langle \Phi_1, \Phi_2 \rangle \in \mathcal{A}(\Phi)$, $|\Phi_1| + |\Phi_2| = |\Phi| + 1$.

Proof: For all n, let $\varphi_{j,n}$ be the projection of ϕ_n over P_j. Let $\mathbb{Z}_2^{N-1}$ be the set of sequences of integers in $\{1, 2\}$ of length $N-1$. Let $(u_n)_{0 \le n < N-1}$ be such a sequence. For $j \in \{1, 2\}$, we construct the sequence $(\tau_n^j)_{0 \le n \le N}$ where $\tau_n^j = \sum_{0 \le i < n} [u_i]_j$, that is, τ_n^j is the number of occurrences of j in the sequence u up to index n.

For $u \in \mathbb{Z}_2^{N-1}$ and $j \in \{1, 2\}$, and for all n, we define $\phi_{j,n}(u) = \bigwedge_{z : \tau_z^j = n} \varphi_{j,z}$ and $\Phi_j(u) = (\phi_{j,n}(u))_{0 \le n < \tau_N^j}$. Finally, we define $\mathcal{A}(\Phi) = \{\langle \Phi_1(u), \Phi_2(u) \rangle, u \in \mathbb{Z}_2^{N-1}\}$.

The rest of the proof is similar to that of Theorem 1 and 2. □

Theorem 4 *Let Φ be a stability condition over $P_1 \amalg P_2$ then there exists a set $\mathcal{S}(\Phi)$ of pairs of stability conditions, such that for any* sequential *composition $\mathcal{T}_1 ; \mathcal{T}_2 = \mathcal{T}$, and for any state $\langle s_1, s_2 \rangle$ of $\mathcal{T}$, we have that $\mathcal{T}, \langle s_1, s_2 \rangle \models \Phi$ iff $\exists \langle \Phi_1, \Phi_2 \rangle \in \mathcal{S}(\Phi)$ such that $\mathcal{T}_1, s_1 \models \Phi_1$ and $\mathcal{T}_2, s_2 \models \Phi_2$.*

If Φ is of finite length, then we have the bound $|\mathcal{S}(\Phi)| \le |\Phi| + 1$, and if $0 < |\Phi|$ then for any $\langle \Phi_1, \Phi_2 \rangle \in \mathcal{S}(\Phi)$, $|\Phi_1| + |\Phi_2| = |\Phi| + 1$.

Proof: For all n, let $\varphi_{j,n}$ be the projection of ϕ_n over P_j.

Let $\phi_{1,n}(i)$ be defined for $0 \le n < i \le N$ as $\phi_{1,n}(i) = \varphi_{1,n}$ and for $0 \le n = i < N$ as $\phi_{1,n}(i) = \bigwedge_{i \le m < N} \varphi_{1,m}$.

We define $\phi_{2,n}(i)$ for $0 \le i \le N$ and $0 \le n \le N - i$, with the constraint $n < N$.

If $0 = n$ then $\phi_{2,n}(i) = \bigwedge_{0 \le m \le i} \varphi_{2,m}$ else $\phi_{2,n}(i) = \varphi_{2,n+i}$.

For $0 \le i \le N$ and $n < N$, we define $\Phi_1(i) = (\phi_{1,n}(i))_{0 \le n \le i}$ and $\Phi_2(i) = (\phi_{2,n}(i))_{0 \le n \le N-i}$. Finally, we define $\mathcal{S}(\Phi) = \{\langle \Phi_1(i), \Phi_2(i) \rangle, 0 \le i \le N\}$.[3]

The rest of the proof is similar to that of Theorem 1 and 2. □

Theorems 1, 2, 3, and 4 provide a policy for solving composed problems based on solutions for the local problems.

3 SOLVING FINITE COMPOSED MODEL CHECKING PROBLEMS

In this section we consider the application of two simple model checking algorithms to checking composed finite TSs, and show how decomposition changes the worst-case complexity of these algorithms. Both algorithms are based on the simple observation that a state s satisfies a stability condition $(\phi_n)_{0 \le n < N}$, with $0 < N$, if and only if s satisfies ϕ_0 and a successor of s satisfies $(\phi_n)_{1 \le n < N}$.

[3] We assume $\infty - \infty = 0$ and $\infty - \infty + 1 = 1$ for notational convenience.

Dynamic Programming The most popular CTL model checking algorithm is a form of Dynamic Programming (DP) which can be naturally adapted to our setting when the condition to be checked is finite [2]. Algorithm 1 shows pseudo-code for the DP approach to model checking stability conditions. Given a stability condition $(\phi_n)_{0 \leq n < N}$, DP computes for each depth k and each state s whether $s \models (\phi_n)_{k \leq n < N}$. This allows us in particular to answer whether a particular query state s_0 satisfies the full condition. If b is the branching factor of the system, that is the maximum number of outgoing transitions in any given state, then the worst-case complexity of this algorithm is $O(Nb|\Sigma|)$.

Algorithm 1: Dynamic Programming model checking.

```
dp(s_0, (φ_n)_{0≤n<N})
    Let M and M' be two maps from states to booleans
    foreach s ∈ Σ do M(s) ← true
    for k = N − 1 down to k = 0 do
        foreach s ∈ Σ do M'(s) ← M(s) ; M(s) ← false
        foreach s ∈ Σ do
            if s ⊨ φ_k then
                foreach s → s' do
                    if M'(s') then M(s) ← true
    return M(s_0)
```

Depth-First Search A Depth-First Search (DFS) traversal of the state space looking for a path that satisfies the stability condition is a simple alternative to DP. The basic idea here is to check at each level of a DFS that the state satisfies the corresponding formula as described in Algorithm 2. For a DFS, if the formula has length N and the system has branching factor b, then we need b^{N-1} time in the worst case.

Algorithm 2: Depth-First Search model checking.

```
dfs(s, (φ_n)_{k≤n<N})
    if k = N then return true
    else if s ⊨ φ_k then
        foreach s → s' do
            if dfs(s', (φ_n)_{k+1≤n<N}) then return true
    return false
```

Worst-case Analysis Given a TS and a stability condition, we can use DP or DFS to directly solve the associated model checking problem. Alternatively, we can try to decompose the system into local ones and use the theorems of the previous section to obtain an equivalent set of local model checking problems. Assuming a conjunctive stability condition, Table 1 compares the worst-case complexity of both approaches for the three types of composition, in terms of the branching factor and the number of states of the system, and the size of the decomposition of the condition, $|\mathcal{A}|$ and $|\mathcal{S}|$.

Table 1. Worst-case complexity of the model checking of a stability condition of length $N+1$ for composed TSs. We assume that for $j \in \{1, 2\}$, $\mathcal{T}_j$ has $\sigma_j = |\Sigma_j|$ states and a branching factor of b_j.

	Dynamic programming		Depth-First Search	
	Original	Composed	Original	Composed
$\|\|$		$N(\sigma_1 + \sigma_2)$	$(b_1 b_2)^N$	$b_1^N + b_2^N$
$\circ$	$N\sigma_1\sigma_2$	$\|\mathcal{A}\|N(\sigma_1 + \sigma_2)$	$(b_1 + b_2)^N$	$\|\mathcal{A}\|(b_1^N + b_2^N)$
;		$\|\mathcal{S}\|N(\sigma_1 + \sigma_2)$	$\max\{b_1, b_2\}^N$	$\|\mathcal{S}\|(b_1^N + b_2^N)$

The worst-case benefits of decomposition are clear for synchronous systems for both algorithms. For asynchronous and sequential systems using the DP algorithm, the 2^N and $N+1$ bounds on $|\mathcal{A}|$ and $|\mathcal{S}|$ from Theorem 3 and 4, already provides better complexity when the state space is large and the condition is relatively short. This analysis, however, does not show any improvement for DFS in those types of systems.

In the asynchronous case, the computation of the bound on the size of $\mathcal{A}$ made no restrictions on the number of distinct formulas that could appear in Φ. A more refined approach looks at the blocks of consecutive formulas that constitute Φ. If $\Phi = \langle \psi_0, \ldots, \psi_0, \ldots, \psi_{m-1}, \ldots, \psi_{m-1} \rangle$, where formula ψ_i appears N_i consecutive times, then we obtain the tighter bound $|\mathcal{A}| \leq N_{m-1} \prod_{0 \leq i < m-1} (N_i + 1)$. In particular, in planning as well as in GGP we have $m = 2$, $N_0 = N-1$, and $N_1 = 1$, leading to $|\mathcal{A}| \leq N$.

In the sequential composition case, the reason for the unfavourable worst-case complexity is that we may have to search for very asymmetric subplan lengths. In practice, though, the problem may have the following favourable property: "if there exists a solution, then there exists a solution making at least N_0 moves in both subproblems". In that case the resulting complexity is $2(b_1^{N-N_0} + b_2^{N-N_0})$.

4 ANSWER SET PROGRAMMING

We now consider a practical implementation of our theory as a general Answer Set Program (ASP). ASPs are compact logical descriptions used for efficiently generating models. They have a logic programming-like syntax, but can have more exotic elements in the head of their horn clauses. An empty head indicates a *constraint*: the body of the rule should not hold. If the head has one or more atoms inside curly braces then it is a "*generator*"—it indicates that the solver can make arbitrarily many of the atoms true, provided the body holds. If numbers appear to the left or right of the curly braces, then these are additional *cardinality constraints* that require a minimum and maximum of these head atoms to hold, respectively. For further details we refer to Gebser *et al.* [10].

The principle TSs from planning and GGP are routinely represented in ASP [15, 17]. The target domain must be temporalised to allow for a state space exploration. This essentially just adds a time parameter to time-dependent operators, particularly for state-update.

For the following we assume that Φ is a stability condition on a composed Transition System $\mathcal{T}$ with a given initial state s. We provide a generic ASP module based on the theory described in Section 2 to decide whether $\mathcal{T}, s \models \Phi$. We further assume an ASP representation of Φ where every formula is in DNF (predicate `phi`), and an ASP representation of the atomic systems composing $\mathcal{T}$.

Our module searches for a global plan satisfying Φ by solving local model checking problems as per Theorems 1–4. It does so by non-deterministically generating instances of predicates `require` and `pickT`. They respectively represent conditions on and transitions to be taken in the local systems at given timepoints. If the ASP solver finds a model, it outputs a sequence of global transitions (predicate `plan`) satisfying Φ. If the program is unsatisfiable then $\mathcal{T}, s \not\models \Phi$.

Finally, note that our encoding is compatible with infinite stability conditions which may be checked with an incremental ASP solver [9].

The module The generic module is given in Fig. 2. It should be noted that the module described here is not GGP specific and references to "games" are purely a notational convenience. This module applies equally to planning and other TS encoded problems. The variables T, U, and V stand for timepoints, G and C denote games. Variable A represents a transition label and I represents a formula label.

```
1 1{ pickD(I,T) : phi(I,T) } :- time(T).
2  { pickA(G,T) }1 :- act(G,T),asyn(G).
3 1{ pickS(G,T) : act(G,T) }1 :- sequ(G).
4 :- pickS(G,T), legals(C,T,_), child(G,C,_).
5 1{ pickT(G,T,A):legals(G,T,A) }1 :- act(G,T),leaf(G).
6
7 map(G,0,0,0) :- child(G,_,_).
8 map(G,T+1,U+1,V+1):-map(G,T,U,V),act(G,T),sync(G).
9 map(G,T+1,U+1,V):-
10       map(G,T,U,V),act(G,T),sequ(G),pickS(G,I),T≤I.
11 map(G,T+1,U,V+1):-
12       map(G,T,U,V),act(G,T),sequ(G),pickS(G,I),I<T.
13 map(G,T+1,U+1,V):-
14       map(G,T,U,V),act(G,T),asyn(G),pickA(G,T).
15 map(G,T+1,U,V+1):-
16       map(G,T,U,V),act(G,T),asyn(G),not pickA(G,T).
17
18 gmap(G,T,T)  :- root(G),time(T).
19 gmap(C,T,V)  :- gmap(G,T,U),child(G,C,_),map(G,U,V,_).
20 gmap(C,T,V)  :- gmap(G,T,U),child(G,_,C),map(G,U,_,V).
21
22 act(G,0..N-1) :- gmap(G,T,N).
23 require(I,G,U) :- pickD(I,T),gmap(G,T,U).
24 plan(A,T) :- pickT(G,U,A), gmap(G,T,U), not gmap(G,T+1,U).
```

Figure 2. The decomposition framework expressed as ASP code.

The predicates root, child, and leaf describe the structure of the composed game, represented as a binary tree. Each internal node is labelled sync, asyn, or sequ. The legals predicate indicates legal transitions in a given leaf subgame at a given local time. The act predicate indicates the set of timepoints for which the subgame is *active*, that is, the timepoints at which a transition may be applied.

Line 1 to 5 are non-deterministic choice points. pickD corresponds to the existential quantifier in Theorem 1 where, for every timepoint, at least one *minterm* has to be satisfied (note: a *minterm* is a conjunction of literals). An upper bound is not needed since multiple individual minterms may legitimately hold simultaneously. pickA corresponds to Theorem 3, such that for every active timepoint T in an asynchronous product G, we may act in the first subgame pickA(G,T) or in the second subgame not pickA(G,T). pickS corresponds to Theorem 4, for every sequential product, we choose a local timepoint to switch from the first to the second subgame. pickT corresponds to Definition 6, we select exactly one transition for every leaf game and every non-final (active) timepoint.

The map predicate records the time correspondence between an internal node and its children (Line 7 to 16). map(G,T,U,V) means that the timepoint T in game G maps to the timepoint U in the first subgame of G and to V in its second subgame.

For each game, the gmap predicate maps between global and local times (Line 18 to 20). gmap(G,T,U) means that the global time T corresponds to the local time U in game G. This predicate can be based on map and the tree structure of the composition. For the root game, the global and local time are identical. If a game is not root, then the correspondence can be derived via the correspondence of its parent.

require(I,G,U) means that the (projection of) minterm I needs to hold at local time U in game G (Line 23).

Finally, plan(A,T) outputs the global solution found by the ASP solver, if any. Action A needs to happen at global time T (Line 24).

Example *The domain-specific model checking code for* Incredible *is presented in Fig. 3. The factorization of the global game in terms of local ones is given by the composition tree (Line 1 to 4). Recall that the stability condition for* Incredible *is of the form* $\Phi = \langle \phi_1, \ldots, \phi_1, \phi_2 \vee \phi_3 \rangle$*. Lines 6–11 specify* Φ *where* ends(T) *is such that* $|\Phi| =$T*. Lines 12–18 ensure that the local conditions set by the Fig. 2 module are met.*

```
1 root(incredible).            sync(incredible).
2 child(incredible,a0,count). asyn(a0).   leaf(count).
3 child(a0,a1,maze).           asyn(a1).   leaf(maze).
4 child(a1,wait,blocks).       leaf(wait). leaf(blocks).
5
6 1 { ends(0..max_time) } 1.
7 time(0..T) :- ends(T).
8
9 phi(1,T) :- time(T), not ends(T).
10 phi(2,T) :- ends(T).
11 phi(3,T) :- ends(T).
12 :- require(1,maze, T),       gold(T).
13 :- require(1,count,T),       timeout(T).
14 :- require(2,maze, T),   not gold(T).
15 :- require(2,blocks,T),  not tower(T).
16 :- require(3,maze, T),   not gold(T).
17 :- require(3,blocks,T),  not tower(T).
18 :- require(3,count,T),   not timeout(T).
```

Figure 3. ASP code for model checking *Incredible*.

Experimental results To test the practicality of our ASP encoding we considered a number of past GGP competition games that have been designed to be decomposed by suitably sophisticated players. Existing techniques [12, 17] were used to detect and decompose each game and we therefore do not present this code here. The experiments were run on a laptop with an Intel Core i5 2.6GHz processor with 8GB RAM using version 4.2.1[4] of an off-the-shelf ASP solver [10]. No specific solver configuration options were enabled.

Firstly, we considered the problem of finding winning solutions for single-player decomposable games: *Incredible* (the example throughout this paper) and *Multiplehunter* (from the 2013 AAAI GGP competition). *Multiplehunter* is a pawn capturing board game played over nine boards where only one board matters. Due to its own time constraints the original game is in fact unsatisfiable. Consequently, we created two satisfiable variants: one that captures 13 pawns within the required time limit and the other that increases the time limit to capture all 14 pawns.

Secondly, we considered the multi-player context where model checking techniques have traditionally been used to prove game properties such as *playability* [16]. A playable game is one where every player can make at least one legal move in every non-terminal game state. This can be encoded using an alternative stability condition. In particular, assuming a state is playable iff each player has a legal move in that state, then construct a stability condition $\langle \varphi, \ldots, \varphi, \psi \rangle$, such that φ encodes that the state is not terminal and playable, and ψ encodes that the state is not terminal and not playable. This stability condition guarantees unsatisfiability of a playable game and any solution will represent a counter-example. Here we considered the game of *Dualrainbow*, a two-player graph colouring game where each player races to colour their own graph.[5].

The results of our experiments are presented in Table 2. They show timing results for the games solved with and without decomposition. To serve as a benchmark we also considered hand optimised versions: for *Incredible* the superfluous *contemplate* moves were removed and the *blocks world* and *maze* were serialised, for *Multiplehunter* the eight superfluous boards were removed, and for *Dualrainbow* one of the players was similarly removed.

The results are dramatic—queries on decomposed systems can be orders of magnitude faster than their original versions, even ap-

[4] http://potassco.sourceforge.net/
[5] http://gamemaster.stanford.edu

proaching benchmark performance. Note that ASP solvers ground domains first, then solve the propositional translation. Our composed times indicate that this grounding process is the new bottleneck. The benchmarks have an advantage here since they have irrelevant information physically removed from their descriptions. However, it is worth observing that as problems become harder the grounding time typically becomes insignificant in comparison to solving. Consequently, the comparison of solving times can be viewed as a more accurate indicator of performance, further highlighting the benefits of our decomposition technique.

Finally, it should be noted that as we are concerned only with solving composed subgames, Table 2 does not present the decomposition times. However, for completeness we can report that the decomposition of *Incredible* and *Dualrainbow* took 1.9 and 3.2 seconds respectively, while the *Multiplehunter* variants took 38 seconds. Despite these times being based on a naive and unoptimised implementation, the combined times show that there is still a distinct advantage to decomposing games so that they can be solved more efficiently.

Table 2. Results of ASP experiments, expressed in seconds. The number in parentheses is the solving component.

Domain	Original	Composed	Benchmark
Incredible	6.11 (5.87)	1.94 (1.60)	0.63 (0.46)
M.hunter 13	49.17 (43.66)	1.32 (0.38)	0.05 (0.01)
M.hunter 14	>1 hr	1.96 (1.03)	0.33 (0.28)
M.hunter (unsat)	2201 (2196)	1.31 (0.37)	0.19 (0.14)
Dualrainbow	19.04 (18.46)	3.52 (2.78)	10.50 (10.19)

5 CONCLUSION

In this paper we have provided a theoretically sound and comprehensive basis to reduce the model checking of products of TSs to the model checking of the factors. As well as providing a strong foundational theory and complexity results, we further showed how this theory can be applied in a practical GGP setting. In particular we provided concrete experimental results showing that an ASP encoding of decomposable games using our approach can provide for dramatic performance gains for solving and proving properties of games.

Our theoretical results provide avenues for future research both within the GGP domain and further afield. Within the GGP domain, while we have shown how to solve decomposable single-player games, the path is less clear for multi-player games. Indeed, how to generalize the stability conditions and their decomposition to express common multi-player solution concepts remains an open problem.

Beyond the GGP domain, an important direction for future research would be to consider the applications of our approach to the field of AI planning, and in particular, factored planning [1]. Factored planning involves the decomposition of a domain into multiple factors (sub-domains) as a means of reducing the global search space of plans. However, there are key differences between factored planning and our composed TS based approach.

Most obviously the general setting of AI planning has no correspondence to multi-player games where each player is competing to maximise its own goal. Furthermore, planning does not consider the separation of goals from termination conditions, which has been a key challenge for GGP decomposition to ensure that games are not prematurely terminated due to sub-optimal interleaving of sub-game actions. Consequently, when considered with respect to these two differences GGP problems represent a more general class with planning being one particular specialisation.

On the other hand there is also a sense in which factored planning is the more general approach. In particular, we require that fluents and actions be associated with only one sub-game, while factored planning typically allows for overlapping sub-domains where both fluents and actions can be shared [1, 3, 14].

These differences raise a number of avenues for future research. Firstly, it would be useful to consider the extent to which our approach can be directly applied to factored planning problems with non-overlapping sub-domains. Secondly, further work is required to determine whether factored problems, or a sub-class such as stratified decompositions [4], with overlapping sub-domains can be mapped into equivalent problems with non-overlapping sub-domains. It would then be necessary to establish the theoretical and practical consequences of such a transformation.

Finally, there is further scope to explore the broader relationship between the theory of composed transition systems developed here to other areas of AI research that similarly employ some form of decomposition. For example, the decomposition of a Markov Decision Process (MDP) into hierarchies of smaller MDPs is an important attribute of hierarchical reinforcement learning [6, 13].

ACKNOWLEDGEMENTS

This research was supported by the Australian Research Council (project numbers DP 120102144 and DP 120102023). The fourth author is also affiliated with the University of Western Sydney.

REFERENCES

[1] E. Amir and B. Engelhardt, 'Factored planning', in *Proc. of IJCAI*, pp. 929–935, (2003).
[2] C. Baier and J. Katoen, *Principles of model checking*, The MIT Press, April 2008.
[3] R. I. Brafman and C. Domshlak, 'Factored planning: How, when, and when not', in *Proc. of AAAI*, pp. 809–814, (2006).
[4] Y. Chen, Y. Xu, and G. Yao, 'Stratified planning', in *Proc. of IJCAI*, pp. 1665–1670, (2009).
[5] E. Cox, E. Schkufza, R. Madsen, and M. Genesereth, 'Factoring general games using propositional automata', in *Proc. of IJCAI Workshop on General Game Playing (GIGA)*, pp. 13–20, (2009).
[6] T. G. Dietterich, 'Hierarchical reinforcement learning with the MAXQ value function decomposition', *CoRR*, **cs.LG/9905014**, (1999).
[7] H. Finnsson and Y. Björnsson, 'Simulation-based approach to general game playing', in *Proc. of AAAI*, pp. 259–264, (2008).
[8] Michael G., N. Love, and B. Pell, 'General game playing: Overview of the AAAI competition', *AI Magazine*, **26**(2), 62–72, (2005).
[9] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele, 'Engineering an incremental ASP solver', in *Proc. of ICLP*, pp. 190–205, (2008).
[10] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub, 'Conflict-driven answer set solving', in *Proc. of IJCAI*, pp. 386–392, (2007).
[11] M. Genesereth and Y. Björnsson, 'The international general game playing competition', *AI Magazine*, **34**(2), 107–111, (2013).
[12] M. Günther, S. Schiffel, and M. Thielscher, 'Factoring general games', in *Proc. of IJCAI Workshop on General Game Playing (GIGA)*, pp. 27–34, (2009).
[13] B. Hengst, 'Discovering hierarchy in reinforcement learning with HEXQ', in *Proc. of ICML*, pp. 243–250, (2002).
[14] E. Kelareva, O. Buffet, J. Huang, and S. Thiébaux, 'Factored planning using decomposition trees', in *Proc. of IJCAI*, pp. 1942–1947, (2007).
[15] V. Lifschitz, 'Answer set programming and plan generation', *Artif. Intell.*, **138**(1-2), 39–54, (2002).
[16] J. Ruan and M. Thielscher, 'Strategic and epistemic reasoning for the game description language GDL-II', in *Proc. of ECAI*, pp. 696–701, (2012).
[17] M. Thielscher, 'Answer set programming for single-player games in general game playing', in *Proc. of ICLP*, pp. 327–341, (2009).
[18] D. Zhao, S. Schiffel, and M. Thielscher, 'Decomposition of multi-player games', in *Australasian Conf. on AI*, pp. 475–484, (2009).