Strategy Logics and the Game Description Language

WIEBE VAN DER HOEK, JI RUAN AND MICHAEL WOOLDRIDGE

ABSTRACT. The Game Description Language (GDL) is a special purpose declarative language for defining games. GDL is used in the AAAI General Game Playing Competition, which tests the ability of computer programs to play games in general, rather than just to play a specific game. Participants in the competition are provided with a game specified in GDL, and then required to play this game. Recently, there has been much interest in the use of strategic cooperation logics for reasoning about game-like scenarios – Alternating-time Temporal Logic (ATL) is perhaps the best known example. The aim of this paper is to make a link between ATL and GDL. We show that a GDL specification can be viewed as a specification of an ATL model, and that ATL can thus be interpreted over GDL specifications. Our main result is that it is possible to translate a propositional GDL specification into an "equivalent" ATL specification, which is only polynomially larger than the original GDL specification. As a corollary, we are able to characterise the complexity of reasoning about GDL-specified games using ATL: it is EXPTIME-complete.

Keywords: Game Description Language, ATL, Game Playing, model checking

1 Introduction

Game playing competitions, particularly between humans and computers, have long been part of the culture of artificial intelligence. Indeed, the victory of IBM's Deep Blue computer over then world champion chess player Gary Kasparov in 1997 is regarded as one of the most significant events in the history of AI. However, a common objection to such specialised competitions and dedicated game playing systems is that they explore only one very narrow aspect of intelligence and rationality. To overcome these objections, in 2005 AAAI introduced a *general game playing competition*, intended to test *the ability to play games in general*, rather than just the ability to play a single game [GL05]. Participants in the competition are computer programs, which are provided with the rules to previously unknown games during the competition itself; they are required to play these games, and the overall winner is the one that fared best against overall. Participant programs must interpret the rules of the games *themselves*, without human intervention. The

Game Description Language (GDL) is a special purpose computer processable language which was developed in order to define the games to be played. Thus, a participant had to be able to interpret game definitions expressed in GDL, and then to play the game defined by the GDL specification.

Since GDL is a language for defining games, it seems natural to investigate the problem of reasoning about games defined using GDL. One successful formalism for reasoning about games is *Alternating-time Temporal Logic* (ATL) [AHK02]. The basic construct of ATL is the *cooperation modality*, $\langle\langle C\rangle\rangle\varphi$, where C is a collection of agents, meaning that coalition C can cooperate to achieve φ; more precisely, that C have a winning strategy for φ. ATL has been widely applied to reasoning about game-like multi-agent systems in recent years, and has proved to be a powerful and expressive tool for this purpose. In this paper, we make a concrete link between ATL and GDL. Specifically, we show that GDL specifications can be interpreted as specifications of an ATL model, and that ATL can thus be interpreted over GDL specifications.

Our main result is that it is possible to translate propositional GDL specifications into ATL specifications that are equivalent up to alternating bisimulation, and which are only polynomially larger than the original GDL specification. As a corollary, we are able to characterise the complexity of ATL reasoning about propositional GDL games: the problem is EXPTIME-complete. Apart from its theoretical interest, the main *application* of our work, we believe, is in having an approach to formal verification of GDL specifications: the GDL game designer can express desirable properties of games using ATL, and then automatically check whether these properties hold of their GDL specifications. It also seems possible in principle to employ our results in the use of ATL as a knowledge representation formalism for game playing agents, using model checking approaches to implement strategic reasoning. This possibility is at present of largely theoretical interest, and we do not propose it as a practical game-playing approach!

2 Game Descriptions and Game Models

GDL is intended for specifying games [GL05]. A game specification must define the states of the game, a unique initial state, and the players in the game ("roles"). For every state and every player, it must also define the moves available to that player in that state. It must also define the state transition function of the game: how moves transform the state of play. Finally, it must define what constitutes a win, and when a game is over. The approach adopted by GDL to represent games succinctly is to use a *logical* definition of the game. We informally introduce GDL, by way of an example (Figure 1): a version of "Tic-Tac-Toe". In this game, two players take turns to mark a 3×3 grid, and the player who succeeds in placing three of its marks in a row, column, or diagonal wins. GDL uses a prefix rule notation based on LISP. The first two lines, `(role xplayer)` and `(role oplayer)`,

```
(role xplayer)                  (<= (legal ?w (mark ?x ?y))
(role oplayer)                      (true (cell ?x ?y b))
(init (cell 1 1 b))                 (true (control ?w)))
...                             (<= (legal oplayer noop)
(init (cell 3 3 b))                 (true (control xplayer)))
(init (control xplayer))
(<= (next (cell ?m ?n x))       ...
    (does xplayer (mark ?m ?n)) (<= (goal xplayer 100)
    (true (cell ?m ?n b)))          (line x))
...                             ...
(<= (next (control oplayer))    (<= terminal
    (true (control xplayer)))       (line x))
(<= (line ?m ?x)                (<= terminal
    (true (cell ?m 1 ?x))           (line o))
    (true (cell ?m 2 ?x))
    (true (cell ?m 3 ?x)))
...
```

Figure 1. A fragment of a game in the Game Description Language

define the two players in this game. The following `init` lines define facts true in the initial state of the game (all the cells are blank, and `xplayer` has the control of the game). The following rule defines the effect of performing an action: if cell (m,n) is blank (`cell ?m ?n b`), and `xplayer` marks cell (m,n), then in the `next` state, it will be true that cell (m,n) is marked by `x`: `cell ?m ?n x`. The next rule says that if the current state is in control of `xplayer`, then the next state will be in control of `oplayer`. There are rules that define what it means to have a `line` of symbols. The first rule in the second column defines when it is `legal` for a player `?w` to perform a mark action. The `goal` rule defines the aim of the game: it says that the `xplayer` will get a reward of 100 if it brings about a line marked by `x`. The final, `terminal` rules define when the game is over.

Overall, then, a GDL definition consists of a list of rules, similar to logic programming languages. Certain operators in a GDL definition have a special meaning: `role` (used to define the players of the game); `init` (defining initial facts); `legal` (defining pre-conditions for actions); and `goal` (defining rewards for agents). An additional operator, `true`, is sometimes used, to make explicit that a particular expression should be true in the current state of the game. In what follows, we will formally define the interpretation of GDL specifications with respect to game models. We begin by introducing Datalog programs.

Datalog Programs and their Semantics Let $\Pi = \{p, q, \dots\}$ be a set of atomic propositions. Then with $\ell(\Pi)$ we denote the set of literals over Π: $\ell(\Pi) = \Pi \cup \{\neg p \mid p \in \Pi\}$. A Datalog rule is of the form $p \Leftarrow \ell_1 \wedge \cdots \wedge \ell_n$ where $p \in \Pi$ and $\ell_i \in \ell(\Pi)$ $(i \leq n)$. If the displayed rule is called r, we call p its head ($p = hd(r)$) and the body of r, $bd(r)$, is $\ell_1 \wedge \cdots \wedge \ell_n$. We will also write $\ell_i \in bd(r)$. A body may be empty. A Datalog Program Δ is a set of Datalog rules.

DEFINITION 1 (Stratified and Acyclic Datalog Programs). Let a Datalog program Δ be given. Its *Dependency Graph* $DGraph(\Delta)$ is a labeled graph $\langle \Pi, R, lab \rangle$, where Rqp iff there is a rule $r \in \Delta$ with $p = hd(r)$ and either $q \in bd(r)$ (in which case $+ \in lab(p,q)$) or else $\neg q \in bd(r)$ in which case $- \in lab(p,q)$. Note that we can have both $-$ and $+$ in $lab(p,q)$. A Datalog program Δ is called *stratified* if its dependency graph contains no cycles with a $-$ label. An atom p is said to be in stratum $i \in \mathbb{N}$ if the maximum number of arcs labeled $-$ on any path ending at p in $DGraph(\Delta)$ is i. A rule $r \in \Delta$ is of stratum i if $hd(r)$ is. A Datalog program Δ is called *acyclic* if $DGraph(\Delta)$ contains no cycles.

Stratification guarantees that, when computing a model for Δ, whenever we have a literal $\neg q$ in the body of a rule r, we will consider all rules r' with $hd(r') = q$ before considering r. Δ is acyclic iff there is a level mapping $f : \ell(\Pi) \to \mathbb{N}$ for which $f(p) = f(\neg p)$ and for every rule $p \Leftarrow \ell_1 \wedge \cdots \wedge \ell_n$ in Δ, $f(p) > f(\ell_i)$, for all $i \leq n$.

DEFINITION 2 (Stratified Model). Given a stratified Datalog program Δ, we define its model $s = DatlogPMod(\Delta)$ as follows. First of all, let $t_0 = \{p \mid p \Leftarrow\in \Delta\}$. Suppose t_i is defined, initialise s_i to t_i and, as long as there is a rule $h \Leftarrow bd$ in stratum i such that $s_i \models bd$, add h to s_i. After this, put $t_{i+1} = s_i$. If the maximum stratum of Δ is k, put $s = t_{k+1}$.

THEOREM 3 ([Apt90]). The model s defined in Definition 2 is a unique model for Δ, and it does not depend on the particular stratification.

DEFINITION 4 (Completion of Δ). Given an acyclic Datalog program Δ, the completion of Δ is a set of formulas $CP(\Delta)$ as follows. Let the *definition* $\mathcal{D}(\Delta, p)$ of p be the set of rules r in Δ for which $hd(r) = p$. Then let

$$cp(p) = (p \leftrightarrow \bigvee_{r \in \mathcal{D}(\Delta,p)} bd(r))$$

where every empty body bd is replaced by $\top$. Note that, if p does not occur as a head in any rule in Δ, we have $cp(p) = \neg p$. Finally, the Clark completion $CP(\Delta)$ of a Datalog program Δ over Π is simply $\{cp(p) \mid p \in \Pi\}$.

THEOREM 5. *Let Δ be a set of acyclic rules, and Π be the set of atoms in consideration. We have $\forall p \in \Pi, p \in DatlogPMod(\Delta)$ iff $CP(\Delta) \models_{cl} p$,where $DatlogPMod(\Delta)$ is the stratified model of Δ, and the set $CP(\Delta)$ is the Clark completion of Δ and $\models_{cl}$ denotes consequence in classical logic.*

Game Descriptions We now more formally define GDL game descriptions. We make the following simplifications. First, we use a variable-free version of GDL, so variables like `?m`, `?n` are replaced by their values. Thus `(cell ?m ?n b)` is replaced by `(cell 1 1 b)`, $\cdots$, `(cell 3 3 b)`. Next, while GDL permits predicates such as `(cell 1 1 b)`, we will assume that the state of a game is

characterised by a set of nullary predicates $\mathtt{A}, \mathtt{B}, \ldots$. Thus, we think of a GDL predicate `(cell 1 1 b)` as a nullary predicate `cell_1_1_b`.

DEFINITION 6. *The set of* atomic propositions *of GDL, denoted At_{GDL}, is defined as follows. It contains* $\mathtt{A}, \mathtt{B}, \ldots$ *(the set of nullary predicates); a special atom* `terminal`*; atoms of the form* `legal i a` *(with* `i` *an agent and* `a` *an action); and, finally, atoms* `reward i v` *(with* `i` *an agent and* `v` *a value at most 100). We will use* $\mathtt{p}, \mathtt{q}, \ldots$ *as variables over these atoms. The set of* atomic expressions *$AtExpr_{GDL}$ of GDL, is defined as follows:*

- *for* $\mathtt{p} \in At_{GDL}$, `init p, true p, next p` $\in AtExpr_{GDL}$*;*
- *for every agent* `i` *and action* `a`, `role i, does i a` $\in AtExpr_{GDL}$.

$LitExpr_{GDL}$ *is* $AtExpr_{GDL} \cup \{\mathtt{not\ true\ p} \mid \mathtt{true\ p} \in AtExpr_{GDL}\}$.

A *game description* specifies the atoms from At_{GDL} that are true, either in the initial state, or as a result of global constraints, or as the effect of performing some joint actions in a given state.

DEFINITION 7 (Game Description Γ). A game description is a set of programming rules Γ over $LitExpr_{GDL}$. A programming rule r is of the form $h \Leftarrow e_1 \wedge \cdots \wedge e_m$ where h, the head $hd(r)$ of the rule, is an element of $AtExpr_{GDL}$ and each e_i ($i \leq m$) in the body $bd(r)$ of r is a literal from $LitExpr_{GDL}$. However, there are some restrictions on $hd(r)$ and $bd(r)$, as explained below. If in the above format $m = 0$, we say that r has *an empty body*. We can partition every game description Γ as $\Gamma = \Gamma_{\mathtt{init}} \cup \Gamma_{\mathtt{role}} \cup \Gamma_{\mathtt{glob}} \cup \Gamma_{\mathtt{next}}$, where:

- $\Gamma_{\mathtt{init}}$ is a set of init-constraints, i.e., constraints of the form $(\mathtt{init\ p}) \Leftarrow$. They have an empty body and their head represents an initial constraint.
- $\Gamma_{\mathtt{role}}$ contains all claims of the form $(\mathtt{role\ x}) \Leftarrow$. They specify what the agents in the game are.
- $\Gamma_{\mathtt{glob}}$ is a set of global constraints, i.e., rules of the form $(\mathtt{true\ p} \Leftarrow e_1 \wedge \cdots \wedge e_m)$, where each body e_i ($i \leq m, m \geq 0$) is either of the form `true q` or `not true q`.
- $\Gamma_{\mathtt{next}}$ contains all rules with a $\mathtt{next(p)}$ in the head: $(\mathtt{next(p)} \Leftarrow e_1 \wedge \cdots \wedge e_m)$ where each e_i is of the form `true q` or `not true q` or $\mathtt{does\ i\ a_i}$.

Game Models From game specifications, we can build game models. Our game models are based on the tree representation. For the description of Game Models G, our approach is equivalent to that of [GL05]. Instead of *roles* we will refer to a set $Ag = \{1, \ldots, n\}$ of *agents* or *players*. Given a set of atomic propositions At_{GDL}, a *Game Model* is then a structure:

$$G = \langle S, s_0, Ag, Ac_1, \cdots, Ac_n, \tau, \pi \rangle$$

where S is a set of game states; $s_0 \in S$ is the initial state of G; Ag denotes the set of agents; Ac_i is a set of actions for agent i; $\tau : Ac_1 \times \cdots \times Ac_n \times S \rightarrow S$ is such that $\tau(\langle a_1, \ldots, a_n \rangle, s) = u$, means that if in game state s, agent i chooses action a_i, ($i \leq n$), the system will transfer to its successor state u, and we require all states, except the initial state, have only one predecessor; and $\pi : S \rightarrow 2^{At_{GDL}}$ is an interpretation function that gives each state a set of atomic propositions in At_{GDL}. We will often abbreviate an action profile $\langle a_1, \ldots, a_n \rangle$ to $\vec{a}$. (Note that we do not include the subset $T \subseteq S$ included in the game models of [GL05]. This subset is supposed to denote the terminal states: we can obtain this set in G by simply collecting all the states that satisfy `terminal`.)

Now we must specify when a game model G is a model for a game description Γ; this makes precise the informal description of [GL05], and in fact represents the first formal semantics for GDL. We compute the game model $GMod(\Gamma)$ for a game description Γ as follows. The main idea is that every state $s \in S$ of $GMod(\Gamma)$ is associated with the unique model under the stratified semantics of some Datalog Program Δ that is derived from Γ. In particular, let $\delta(\Gamma_{\texttt{glob}})$ be derived from $\Gamma_{\texttt{glob}}$ by replacing every occurrence of `true p` by `p`. Since we assume that Δ does not contain `init` or `next` in any body of any rule, $\delta(\Gamma_{\texttt{glob}})$ is indeed a Datalog Program. Also, let $\delta(\Gamma_{\texttt{init}})$ be $\{\texttt{p} \Leftarrow\mid \texttt{init p} \Leftarrow\in \Gamma_{\texttt{init}}\}$. The set Ag of agents, and Ac_i of actions for agent i in $GMod(\Gamma)$ are immediately read off from Γ: $Ag = \{i \mid \texttt{role i} \Leftarrow \in \Gamma_{\texttt{role}}\}$ and $Ac_i = \{a_i \mid \texttt{does i a}_\texttt{i} \text{ occurs in } \Gamma\}$. In the following, we construct S, τ, and π step by step. First, we define s_0. Put

$$\pi(s_0) = DatlogPMod(\delta(\Gamma_{\texttt{init}}) \cup \delta(\Gamma_{\texttt{glob}}))$$

Next, suppose a game state $s \in S$ has already been defined. If this is not a terminal state, i.e., `terminal` $\notin \pi(s)$, each agent should have at least one legal action available. An action a_i is legal for agent i in state s, if and only if (`legal i a`$_i$) $\in \pi(s)$. If `terminal` $\notin \pi(s)$, we define, for every profile of legal actions $\langle a_1, \cdots, a_n \rangle$, a successor u of s by first computing the atoms that need to be true due to $\Gamma_{\texttt{next}}$.

$$F_\Gamma(\langle a_1, \ldots, a_n \rangle, s) = \quad \{\, \texttt{p} \Leftarrow \mid \quad \exists\, (\texttt{next(p)} \Leftarrow bd) \ \in \Gamma_{\texttt{next}} \ \& \ \pi(s) \cup \{\texttt{does i a}_\texttt{i} \mid i \leq n\} \models_{cl} bd\}$$

So, $F_\Gamma(\langle a_1, \ldots, a_n \rangle, s)$ computes those atoms that need to be true in the next state (the F is for 'forward'), given that each agent i performs a_i. Now we add:

$$u = \tau(\langle a_1, \ldots, a_n \rangle, s) \text{ and } \pi(u) = DatlogPMod(F_\Gamma(\langle a_1, \ldots, a_n \rangle, s) \cup \delta(\Gamma_{\texttt{glob}}))$$

We illustrate the main idea by the Tic-Tac-Toe example. As `control xplayer` $\in \delta(\Gamma_{\texttt{init}})$, using $\Gamma_{\texttt{glob}}$, we then get `(legal xplayer (mark 1 1))` $\in \pi(s_0)$, and `(legal oplayer noop)` $\in \pi(s_0)$. We also see that `terminal`$\notin$

$\pi(s_0)$, because the bodies of all the global rules with head `terminal` can not be satisfied. Thus we have an action profile $\vec{a} = \langle$`mark 1 1`, `noop`$\rangle$. It is easy to verify that `(cell 1 1 x)` and `(control oplayer)`$\in F_\Gamma(\vec{a}, s_0)$.

Atoms of the form `does i` $\mathtt{a_i}$ are not added to the game model $GMod(\Gamma)$ – they are only used to calculate different successors for a given game state s. So, they incorporate a kind of hypothetical reasoning of the form: "suppose player i were to do a_i, what would be the resulting next state?"

It is now easy to verify that $GMod(\Gamma)$ is indeed a game model for Γ, if we give the following truth definitions. Let $G = \langle S, s_0, Ag, Ac_1, \cdots, Ac_n, \tau, \pi\rangle$ be a game model. Let $\{i_1, \ldots, i_k\} = Ag'$ be a set of agents $\subseteq Ag$, each i_x with an action a_x ($x \leq k$). Then we say that t is an $i_1 : a_1, \ldots i_k : a_k$ successor of s if there is a choice for the agents j in $Ag \setminus Ag'$ for an action b_j from Ac_j such that $\tau(\langle c_1, \ldots, c_n\rangle, s) = t$, where $c_v = a_x$ if $v = i_x \in Ag'$, and $c_v = b_j$ if $v = j \in Ag \setminus Ag'$. For a game state s we define $s \models_{GDL}$ `p` iff `p` $\in \pi(s)$: the other connectives are straightforward.

- $G \models_{GDL}$ `init p` iff $s_0 \models_{GDL}$ `p`
- $G \models_{GDL}$ `true p` $\Leftarrow bd$ iff $\forall s, s \models_{GDL} bd \Rightarrow s \models_{GDL}$ `p`
- $G \models_{GDL}$ `next(p)` $\Leftarrow$ `true` $\mathtt{p_1} \wedge \ldots$ `true` $\mathtt{p_n} \wedge$ `does` $\mathtt{i_1}\ \mathtt{a_1} \wedge \cdots \wedge$ `does` $\mathtt{i_k}\ \mathtt{a_k}$ iff $\forall s, t:$ ($s \models_{GDL}$ $\mathtt{p_1} \wedge \cdots \wedge \mathtt{p_n}$ and t is an $i_1 : a_1, \ldots i_k : a_k$ successor of s $\Rightarrow t \models_{GDL}$ `p`).

It is straightforward to verify that $GMod(\Gamma) \models_{GDL} \Gamma$.

Alternating-time Temporal Logic (ATL) We now want to introduce a logic for reasoning about games defined using GDL. For this, we believe *Alternating-time Temporal Logic* is ideally suited [AHK02]. The key construct in ATL is $\langle\langle C\rangle\rangle T\varphi$, where C is a coalition, (a set of agents), and $T\varphi$ a temporal formula, meaning "coalition C can act in such a way that $T\varphi$ is guaranteed to be true". Temporal formulas are built using the unary operators $\bigcirc$, $\Box$, $\diamondsuit$, and $\mathcal{U}$, where $\bigcirc$ means "in the next state", $\Box$ means "always", $\diamondsuit$ means "eventually", and the binary operator $\mathcal{U}$ means "until". The syntax of ATL formulas, (with respect to a set of agents Ag, and a set of atomic propositions Φ), is given by the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\langle C\rangle\rangle \bigcirc \varphi \mid \langle\langle C\rangle\rangle \Box \varphi \mid \langle\langle C\rangle\rangle \varphi \mathcal{U} \varphi$$

where $p \in \Phi$ is a propositional variable and $C \subseteq Ag$ is a set of agents.

ATL has a number of equivalent semantics; since *moves*, or *actions*, play such a prominent role in game playing, we use *Action-based Alternating Transition Systems* . An *Action-based Alternating Transition System* (AATS) is a tuple

$$\mathcal{A} = \langle Q, q_0, Ag, Ac_1, \ldots, Ac_n, \rho, \tau, \Phi, \pi\rangle$$

where: Q is a finite, non-empty set of *states*; $q_0 \in Q$ is the *initial state*; $Ag = \{1, \ldots, n\}$ is a finite, non-empty set of *agents*; Ac_i is a finite, non-empty set of *actions*, for each $i \in Ag$, where $Ac_i \cap Ac_j = \emptyset$ for all $i \neq j \in Ag$; $\rho : Ac_{Ag} \rightarrow 2^Q$ is an *action precondition function*, which for each action $a \in Ac_{Ag}$ defines the set of states $\rho(a)$ from which a may be executed; $\tau : Ac_1 \times \cdots \times Ac_n \times Q \rightarrow Q$ is a partial *system transition function*, which defines the state $\tau(\vec{a}, q)$ that would result by the performance of $\vec{a}$ from state q – note that, as this function is partial, not all joint actions are possible in all states (cf. the precondition function above); Φ is a finite, non-empty set of *atomic propositions*; and $\pi : Q \rightarrow 2^\Phi$ is an interpretation function, which gives the set of atomic propositions satisfied in each state: if $p \in \pi(q)$, then this means that the propositional variable p is satisfied in state q.

It is required that AATSs satisfy the following coherence constraints: (*Non-triviality*) agents always have at least one legal action – $\forall q \in Q, \forall i \in Ag, \exists a \in Ac_i$ s.t. $q \in \rho(a)$; and (*Consistency*) the ρ and τ functions agree on actions that may be performed: $\forall q, \forall \vec{a} = \langle a_1, \cdots, a_n \rangle, (\vec{a}, q) \in \text{dom } \tau$ iff $\forall i \in Ag, q \in \rho(a_i)$.

Given an agent $i \in Ag$ and a state $q \in Q$, we denote the *options* available to i in q – the actions that i may perform in q – by $options(i, q) = \{a \mid a \in Ac_i \text{ and } q \in \rho(a)\}$. For a coalition C, we define $options(C, q) = \bigcup\{options(i, q) \mid i \in C\}$. We then say that a *strategy* for an agent $i \in Ag$ is a function $\sigma_i : Q \rightarrow Ac_i$ which must satisfy the *legality* constraint that $\sigma_i(q) \in options(i, q)$ for all $q \in Q$. A *strategy profile* for a coalition $C = \{i_1, \ldots, i_k\} \subseteq Ag$ is a tuple of strategies $\langle \sigma_1, \ldots, \sigma_k \rangle$, one for each agent $i \in C$. We denote by Σ_C the set of all strategy profiles for coalition $C \subseteq Ag$; if $\sigma_C \in \Sigma_C$ and $i \in C$, then we denote i's component of σ_C by σ_C^i. Given a strategy profile $\sigma_C \in \Sigma_C$ and state $q \in Q$, let $out(\sigma_C, q)$ denote the set of possible states that may result by the members of the coalition C acting as defined by their components of σ_C for one step from q:

$$out(\sigma_C, q) = \{q' \mid \tau(\vec{a}, q) = q' \text{ where } (\vec{a}, q) \in \text{dom } \tau \text{ and } \sigma_C^i(q) = a_i \text{ for } i \in C\}$$

Notice that the set $out(\sigma_{Ag}, q)$ is a singleton. Also, $out(\cdot, \cdot)$ only deals with one-step successors, and we interchangeably write $out(\sigma_C, q)$ and $out(Ac_C, q)$: for the one step future, a strategy carries the same information as an action. A q_0-*computation* is an infinite sequence of states $\lambda = q_0, q_1, \ldots$. If $u \in \mathbb{N}$, then we denote by $\lambda[u]$ the component indexed by u in λ.

Given a strategy profile σ_C for some coalition C, and a state $q \in Q$, we define $comp(\sigma_C, q)$ to be the set of possible runs that may occur if every agent $i \in C$ follows the corresponding strategy σ_i, starting when the system is in state $q \in Q$. That is, the set $comp(\sigma_C, q)$ will contain all possible q-computations that the coalition C can "enforce" by cooperating and following the strategies in σ_C.

$$comp(\sigma_C, q) = \{\lambda \mid \lambda[0] = q \text{ and } \forall u \in \mathbb{N} : \lambda[u+1] \in out(\sigma_C, \lambda[u])\}.$$

Again, note that for any state $q \in Q$ and any grand coalition strategy σ_{Ag}, the set

$comp(\sigma_{Ag}, q)$ will be a singleton, consisting of exactly one infinite computation.

We can now give the rules defining the satisfaction relation "$\models$" for ATL, which holds between pairs of the form $\mathcal{A}, q$ (where $\mathcal{A}$ is an AATS and q is a state in $\mathcal{A}$), and formulas of ATL. We only give the coalitional cases, and we will assume standard abbreviations for logical connectives:

$\mathcal{A}, q \models \langle\langle C\rangle\rangle \bigcirc \varphi$ iff $\exists \sigma_C \in \Sigma_C$, such that $\forall \lambda \in comp(\sigma_C, q)$, we have $\mathcal{A}, \lambda[1] \models \varphi$;

$\mathcal{A}, q \models \langle\langle C\rangle\rangle \Box \varphi$ iff $\exists \sigma_C \in \Sigma_C$, such that $\forall \lambda \in comp(\sigma_C, q)$, we have $\mathcal{A}, \lambda[u] \models \varphi$ for all $u \in \mathbb{N}$;

$\mathcal{A}, q \models \langle\langle C\rangle\rangle \varphi \mathcal{U} \psi$ iff $\exists \sigma_C \in \Sigma_C$, such that $\forall \lambda \in comp(\sigma_C, q)$, there exists some $u \in \mathbb{N}$ such that $\mathcal{A}, \lambda[u] \models \psi$, and for all $0 \leq v < u$, we have $\mathcal{A}, \lambda[v] \models \varphi$.

The following is from [AHKV98].

DEFINITION 8 (Alternating Bisimulation). Let $\mathcal{A}_1 = \langle Q_1, q_1, Ag, Ac_1^1, \ldots, Ac_n^1, \rho_1, \tau_1, \Phi, \pi_1\rangle$ and $\mathcal{A}_2 = \langle Q_2, q_2, Ag, Ac_1^2, \ldots, Ac_n^2, \rho_2, \tau_2, \Phi, \pi_2\rangle$ be two AATS's. Then a relation $\mathcal{R} \subseteq Q_1 \times Q_2$ is called an *alternating bisimulation* if $\mathcal{R}q_1q_2$ and, for every two states t_1 and t_2 for which $\mathcal{R}t_1t_2$,we have:

1. For all $p \in \Phi$, $p \in \pi(t_1)$ iff $p \in \pi(t_2)$.
2. For every coalition $C \subseteq Ag$, and every $ac_C^1 \in options(C, t_1)$, there exists $ac_C^2 \in options(C, t_2)$ such that for every $t_2' \in out(ac_C^2, t_2)$, there is a $t_1' \in out(ac_C^1, t_1)$ so that $\mathcal{R}t_1't_2'$.
3. For every coalition $C \subseteq Ag$, and every $ac_C^2 \in options(C, t_2)$, there exists $ac_C^1 \in options(C, t_1)$ such that for every $t_1' \in out(ac_C^1, t_1)$, there is a $t_2' \in out(ac_C^2, t_2)$ so that $\mathcal{R}t_1't_2'$.

Note that the set of agents in both structures are the same, while the actions in both structures do not have to be the same, since in ATL, one cannot directly refer to actions in the object language. We have:

THEOREM 9 ([AHKV98]). *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be such that there is an alternating bisimulation $\mathcal{R}$ between them, with $\mathcal{R}q_1q_2$. Then, for all ATL-formulas φ:*

$$\mathcal{A}_1, q_1 \models \varphi \qquad \Leftrightarrow \qquad \mathcal{A}_2, q_2 \models \varphi$$

Transformation from Game Models to AATSs Given a game model $G = \langle S, s_0, Ag, Ac_1, \cdots, Ac_n, \tau, \pi\rangle$ and a set of atomic propositions At_{GDL}, we can define an associated AATS $\mathcal{A}_G = \langle Q, q_0, Ag, Ac_1 \cup \{fin_1\}, \ldots, Ac_n \cup \{fin_n\}, \rho, \tau', \Phi, \pi'\rangle$ with the same sets of agents Ag and such that Φ is constructed from At_{GDL} in the following manner.

DEFINITION 10. Define a translation $t : At_{GDL} \rightarrow At_{ATL}$, where we assume that for every nullary predicate $\mathtt{A}$ in At_{GDL} there is an atom p_A in At_{ATL}.

$t(\mathtt{A}) = p_A$ $\quad$ $t(\mathtt{reward\ i\ v}) = \mathit{reward}(i,v)$

$t(\mathtt{legal\ i\ ai}) = \mathit{legal}(i,a_i)$ $\quad$ $t(\mathtt{terminal}) = \mathit{terminal}$

For every $\mathtt{p}$ in At_{GDL}, add $t(\mathtt{p})$ to Φ. To keep track of the actions and state transitions, we add $\mathit{done}(i,a_i)$ to Φ for each $\mathtt{does(i,a}_i\mathtt{)}$ in consideration, and p_{old} for each $\mathtt{p} \in At_{GDL}$. We also add *init* to Φ to indicate the inital state of a game. Finally, we add a special atom $z_\perp$ to Φ. This atom denotes a special 'zink state', z, which we add to $\mathcal{A}_G$ in order to make it a proper AATS. The idea is that z is the only sucessor of every terminal state and itself. The other elements of $\mathcal{A}_G$ are:

- $Q = S \cup \{z\}$, where z is a *sink state*, and $q_0 = s_0$;
- $\rho : Ac_{Ag} \rightarrow 2^Q$ is the *action precondition function*, which agrees, for each agent, with $\mathit{legal}(i,a_i)$, i.e. $\rho(a_i) = \{q \mid q \models \mathit{legal}(i,a_i) \wedge \neg \mathit{terminal}\}$. Moreover, $\rho(\mathit{fin}_i) = \{z\} \cup \{q \mid q \models \mathit{terminal}\}$, for every agent i.
- $\tau' : Ac_1 \times \cdots \times Ac_n \times Q \rightarrow Q$ is exactly as τ, but we have some additional transitions: $\tau'(\langle \mathit{fin}_1, \ldots, \mathit{fin}_n \rangle, q) = z$, for all $q \in \{z\} \cup \{q \mid q \models \mathit{terminal}\}$;
- $\pi' : Q \rightarrow 2^\Phi$ is such that $\pi'(q)$ is the minmal set satisfying the following conditions:
  - $\mathit{init} \in \pi(q_0)$, and $z_\perp \in \pi(z)$,
  - $\pi'(q) \supseteq \{t(\mathtt{p}) \mid \mathtt{p} \in \pi(q)\}$ for all $q \in Q \setminus \{z\}$,
  - $\forall q, q' \in Q \setminus \{z\}$ and an action profile $\vec{a} = \langle a_1, \cdots, a_n \rangle$ such that $q' = \tau'(\vec{a}, q)$, we require $\forall i \in Ag, \mathit{done}(i,a_i) \in \pi'(q')$, and $\{t(\mathtt{p})_{old} \mid \mathtt{p} \in \pi(q)\} \subseteq \pi'(q')$. Moreover $\forall i \in Ag, \mathit{done}(i,\mathit{fin}_i) \in \pi'(z)$.

Our intuition behind π' is that each state, except q_0 and z , has exactly one *done*-proposition for each agent to record the action made in its unique predecessor, and a set of p_{old} to record the atomic propositions that is true in the same predecessor.

Define the translation $\mathcal{T} : \mathit{GDL} \rightarrow \mathit{ATL}$ as follows:

$$\mathcal{T}(\mathtt{init\ p} \Leftarrow) = \mathit{init} \rightarrow t(\mathtt{p})$$

$$\mathcal{T}(\mathtt{true}\ p \Leftarrow e_1 \wedge \cdots \wedge e_m) = \langle\langle\rangle\rangle \Box(\neg z_\perp \wedge \mathcal{T}(e_1) \wedge \cdots \wedge \mathcal{T}(e_m) \rightarrow t(\mathtt{p}))$$

$$\mathcal{T}(\mathtt{next\ p} \Leftarrow e_1 \wedge \cdots \wedge e_m \wedge \mathtt{does\ i_1\ a_1} \wedge \cdots \wedge \mathtt{does\ i_k\ a_k}) = \langle\langle\rangle\rangle \Box(\neg z_\perp \wedge \mathcal{T}(e_1) \wedge \cdots \wedge \mathcal{T}(e_m) \rightarrow \langle\langle\{i_1, \ldots i_k\}\rangle\rangle \bigcirc (\mathcal{T}(p) \wedge \mathit{done}(i_1,a_1) \wedge \cdots \wedge \mathit{done}(i_i,a_k)))$$

where $t : At_{GDL} \to At_{ATL}$ is as in Definition 10, and for an GDL expression e_i, we stipulate: $\mathcal{T}(e_i) = t(\mathtt{p})$ if $e_i =$ `true p`, and $\neg t(\mathtt{p})$ if $e_i =$ `not true p`.

THEOREM 11. *Let Γ be a GDL game description, and $G = GMod(\Gamma)$ its game model. Then, for each rule r, for all $s \in S$ and all $e \in AtExpr_{GDL}$,*

$$G \models_{GDL} r \text{ iff } \mathcal{A}_G \models \mathcal{T}(r) \text{ and } G, s \models_{GDL} e \text{ iff } \mathcal{A}_G, s \models \mathcal{T}(e)$$

3 Game Descriptions and ATL Specifications

We now have a link between GDL and ATL: for every GDL game description Γ, we can compute the game model $GMod(\Gamma)$; and we can associate such game models with ATL models. In this section, we take this link one stage further. We show how, from a game description Γ, we can systematically construct an ATL formula Γ_{ATL} that *completely and correctly characterises the game described by* Γ. We define the formula Γ_{ATL} as the conjunction of several component formulas:

$$\Gamma_{ATL} = INIT \wedge MEM \wedge STRAT \wedge DONE \wedge ONE\_ACT \wedge LEGAL \wedge TERM$$

We start with *INIT*. Let $S_0 = DatlogPMod(\delta(\Gamma_{\mathtt{init}}) \cup \delta(\Gamma_{\mathtt{glob}}))$, which gives the minimal set of atomic consequences (using the global rules) of all `init p` formulas. Consider:

$$INIT = init \wedge \langle\langle\rangle\rangle \bigcirc \langle\langle\rangle\rangle \Box \neg init \wedge P_{at} \wedge \bigwedge_{p_{old} \in At_{ATL}(\Gamma)} \neg p_{old} \wedge \neg terminal \wedge \neg z_{\bot} \wedge N_{done}$$

where $P_{at} = \bigwedge_{\mathtt{p} \in S_0} p \wedge \bigwedge_{\mathtt{p} \notin S_0} \neg p$, and $N_{done} = \bigwedge_{i \in Ag} \bigwedge_{a \in Ac_i \cup \{fin_i\}} \neg done(i, a)$.

The intended use of an atom p_{old} is that it records the old, i.e. previous, truth-value of p. This is captured by the principle *MEM*:

$$MEM = \langle\langle\rangle\rangle \Box \bigwedge_{\mathtt{p} \in At_{ATL}(\Gamma)} \bigwedge_{x = t(\mathtt{p}), \neg t(\mathtt{p})} (x \wedge \neg terminal \to \langle\langle\rangle\rangle \bigcirc x_{old})$$

Let $bd_1, bd_2, \ldots$ be variables over possible bodies of rules, that is, conjunctions of literals, but not including any `does i a`. We assume that atoms `does i a` only occurs in rules which have a head of the form `next p`. This is their intended use: to enable players to compute the next state, given the moves of all players. Let $\mathtt{x} \in At_{GDL}(\Gamma)$. Now suppose that *all* the rules r in Γ with $hd(r) \in \{\mathtt{x}, \mathtt{next\ x}\}$ are the following:

$$\begin{array}{lllllll}
r_1: & \mathtt{x} & \Leftarrow & bd_1 & & & \\
\ldots & \mathtt{x} & \Leftarrow & \ldots & & & \\
r_h: & \mathtt{x} & \Leftarrow & bd_h & & & \\
s_1: & \mathtt{next\ x} & \Leftarrow & bd'_1 & \wedge & \mathtt{does\ i_{1_1}\ a_{1_1}} \wedge \cdots \wedge \mathtt{does\ i_{m_1}\ a_{m_1}} \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \\
s_k: & \mathtt{next\ x} & \Leftarrow & bd'_k & \wedge & \mathtt{does\ i_{k_1}\ a_{k_1}} \wedge \cdots \wedge \mathtt{does\ i_{m_k}\ a_{m_k}}
\end{array}$$

We map all these rules for $\mathtt{x}$ to an ATL formula $\varphi(x)$. For this, we first translate the symbols from GDL to those of ATL using the function t defined in Definition 10. Then we let function t_{old} be exactly like t, except for the predicate atoms: $t_{old}(\mathtt{p}) = t(\mathtt{p})_{old} = p_{old}$. For each such atom x, we can now define an ATL constraint $MIN(x)$, as follows:

$$MIN(\mathtt{x}) = x \leftrightarrow \left(\bigvee_{i \leq h} t(bd_i) \vee \bigvee_{j \leq k} (t_{old}(bd'_j) \wedge done(i_{j_1}, a_{j_1}) \wedge \cdots \wedge done(i_{j_m}, a_{j_m})) \right)$$

And,, if $\mathtt{x}$ does not occur in a head of any rule in Γ, we define $MIN(\mathtt{x}) = \neg x$.

The stratification semantics of Γ is now captured by the following constraint:

$$STRAT = \langle\langle\rangle\rangle \Box \bigwedge_{\mathtt{p} \in At_{GDL}(\Gamma)} (\neg init \wedge \neg z_{\perp} \rightarrow MIN(\mathtt{p}))$$

The following constraint makes sure that for all non-initial states, some action is done by each agent:

$$DONE = \langle\langle\rangle\rangle \Box (\neg init \rightarrow \bigwedge_{i \in Ag} \bigvee_{a_i \in Ac_i \cup \{fin_i\}} done(i, a_i))$$

and only one action is done, as each state only has one predecessor:

$$ONE\_ACT = \langle\langle\rangle\rangle \Box \bigwedge_{i \in Ag, a_i \neq b_i \in Ac_i \cup \{fin_i\}} (done(i, a_i) \rightarrow \neg done(i, b_i))$$

One assumption for playing GDL games is that each agent must play legal moves. This is captured by the following principal:

$$LEGAL = \langle\langle\rangle\rangle \Box \bigwedge_{i \in Ag, a_i \in Ac_i} \bigwedge_{x = i, Ag} (legal(i, a_i) \wedge \neg terminal \leftrightarrow \langle\langle x \rangle\rangle \bigcirc done(i, a_i))$$

This principle says that, when an action a_i is legal for agent i, and the current state is not a terminal state, then agent i should have a strategy to enforce it, and vice versa.

When a terminal state is reached, no further 'real' moves are played:

$$TERM = \langle\langle\rangle\rangle \Box \bigwedge_{i \in Ag} (terminal \vee z_{\perp}) \leftrightarrow \langle\langle\rangle\rangle \bigcirc (z_{\perp} \wedge done(i, fin_i)))$$

It is straightforward to conceive a Game Model as an ATL model. The following is essentially a soundness result for our transformation. Let Γ be a game description, and $GMod(\Gamma)$ its game model, with initial state s_0; recall that $\mathcal{A}_{GMod(\Gamma)}$ is

the AATS system that is obtained from $GMod(\Gamma)$ by interpreting it as an AATS structure. Then:

$$\mathcal{A}_{GMod(\Gamma)}, s_0 \models \Gamma_{ATL}$$

Given this construction, we can state a main result of this paper. In the following, we add a requirement resulting in *uniform* AATS structures:

$$(uni)\ \ \forall s \in Q \forall C \subseteq Ag \forall \sigma_C \quad \forall s', s'' \in out(\sigma_C, s) \forall i \in C \forall a_i \in Ac_i : \\ (done(i, a_i) \in \pi(s') \Leftrightarrow done(i, a_i) \in \pi(s''))$$

This requirement says that, in the outcome states of a coalition C executing a strategy, for the agents in C, the related *done* propositions are uniformally true or false. Easy to see that $\mathcal{A}_{GMod(\Gamma)}$ satisfies this requirement.

LEMMA 12. *Satisfability checking with respect to uniform AATSs is in EXPTIME.*

Suppose we are given a game description Γ. On the one hand, this gives rise to a game model $GMod(\Gamma)$. This model can also be interpreted as an ATL model $\mathcal{A}_{GMod(\Gamma)}$. Now, the result is this: *every model for* Γ_{ATL} *is bisimilar with* $\mathcal{A}_{GMod(\Gamma)}$. Moreover, the transformation of the GDL description Γ into the ATL specification Γ_{ATL} can be done in polynomial time.

THEOREM 13. Let $G = GMod(\Gamma)$ be the model for a game description Γ, and let $\mathcal{A}_1 = \langle Q_1, q_1, Ag, \{Ac_i | i \in Ag\}, \rho_1, \tau_1, \Phi, \pi_1, \rangle$ be its associated AATS structure. Let $\mathcal{A}_2 = \langle Q_2, q_2, Ag, \{Ac_i | i \in Ag\}, \rho_2, \tau_2, \Phi, \pi_2, \rangle$ be an arbitrary uniform AATS for Γ_{ATL}. There exists an alternating bisimulation $\mathcal{R}$ between $\mathcal{A}_1$ and $\mathcal{A}_2$, with $\mathcal{R}q_1q_2$.

Proof. (Sketch) We define a relation $\mathcal{R} \subseteq Q_1 \times Q_2$ as follows:

$$\mathcal{R}z_1z_2 \ \text{ iff } \ \pi_1(z_1) = \pi_2(z_2)$$

Using *INIT*, one readily sees that $\mathcal{R}q_1q_2$. Suppose we have established $\mathcal{R}z_1z_2$ for some $z_1 \in Q_1$ and $z_2 \in Q_2$ (cf Figure 2). Suppose $\mathcal{A}_1, z_1 \not\models terminal$. By assumption, we then also have $\mathcal{A}_2, z_2 \not\models terminal$. Take an arbitrary coalition C with a joint action $ac_C^1 \in options(C, z_1)$, and consider $U_1 = out(ac_C^1, z_1) \subseteq Q_1$ in $\mathcal{A}_1$. We need to find $ac_C^2 \in options(C, z_2)$ such that for every $u_2 \in out(ac_C^2, z_2)$, there is a $u_1 \in U_1$ so that $\mathcal{R}u_1u_2$.

It follows from $ac_C^1 \in options(C, z_1)$ that $\mathcal{A}_1, z_1 \models legal(i, a_i^1)$ for all $i \in C, a_i^1 \in ac_C^1$. Therefore, $\mathcal{A}_2, z_2 \models legal(i, a_i^1)$ for all $i \in C, a_i^1 \in ac_C^1$. And by *LEGAL*, we have $\mathcal{A}_2, z_2 \models \langle\langle i \rangle\rangle \bigcirc done(i, a_i^1)$ for all $i \in C, a_i^1 \in ac_C^1$. So, for all $i \in C, a_i^1 \in ac_C^1$, there is $ac_i^2 \in options(i, z_2)$ such that for all $x \in out(ac_i^2, z_2) \subseteq Q_2$, $\mathcal{A}_2, x \models done(i, a_i)$. Let ac_C^2 be an action profile that consisits of a_i^2 for all $i \in C$ and $U_2 = out(ac_C^2, z_2) \subseteq Q_2$. It is easy to see that for all $x \in U_2$, we have

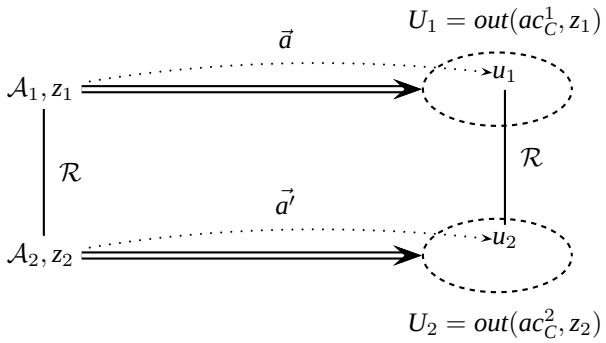

Figure 2. Alternating bisimulation between $\mathcal{A}_1$ and $\mathcal{A}_2$

$x \models done(i, a_i)$ for $i \in C$. We pick an arbitrary $u_2 \in U_2$. We are done if we can show that there is a $u_1 \in U_1$ for which $\mathcal{R}u_1u_2$.

By *DONE* and *ONE_ACT*, there is one and only one $done(i, a)$ true in u_2 for each $i \in Ag$. We already know $u_2 \models done(i, a_i^1)$ for $i \in C$, and we assume $u_2 \models done(j, b_j^1)$ for all $j \in Ag \setminus C$. It follows that $z_2 \models \langle\langle Ag \rangle\rangle \bigcirc done(j, b_j^1)$ for all $j \in Ag \setminus C$, and by *LEGAL*, we have $z_2 \models legal(j, b_j^1)$ for all $j \in Ag \setminus C$, hence $z_1 \models legal(j, b_j^1)$ for all $j \in Ag \setminus C$. Now go back to $\mathcal{A}_1$ and consider $u_1 = out(\vec{a}, z_1)$, where $\vec{a}$ consists of the actions a_i such that $z_1 \models legal(i, a_i)$. We claim that this u_1 is the state we are looking for: it satisfies $\mathcal{A}_1, u_1 \models p$ iff $\mathcal{A}_2, u_2 \models p$, for all $p \in \Phi$. The other direction is proven in a similar way. ■

One way of interpreting the result above is as follows: GDL can be viewed as a *model specification language*, suitable for use in a *model checker* [CGP00]. This gives rise to the formal decision problem of *ATL model checking problem over GDL game descriptions*, which can be described as follows: *Given an ATL formula φ and a GDL game description Γ, is it the case that $\mathcal{A}_{GMod(\Gamma)} \models \varphi$?*

THEOREM 14. *ATL model checking over propositional GDL game descriptions is EXPTIME-complete.*

Proof. Membership in EXPTIME follows from the Theorem 13 and Lemma 12 (cf. [WLWW06]). Given game description Γ, and ATL formula φ, construct Γ_{ATL}, and then check whether $\Gamma_{ATL} \wedge \neg\varphi$ is unsatisfiable; the correctness of this procedure follows from Theorem 13. EXPTIME-hardness may be proved by reduction from the problem of determining whether a given player has a winning strategy in the two-player game PEEK-G_4 [SC79, p.158]. Enocding PEEK-G_4 in GDL is a straightforward exercise in GDL programming, and the question of whether there exists a winning strategy is directly encoded in an ATL formula to model check (cf. [HLW05]). ■

Note that, although this seems a negative result, it means that interpreting ATL over GDL specifications is no more complex than interpreting ATL over apparently simpler model specification languages such as the Simple Reactive Systems Language [HLW05]. We refer to [HKV97] for related complexity results for model checking symbolically-represented systems.

4 Conclusion

There is much interest in the connections between logic and games, and in particular in the use of ATL-like logics for reasoning about multi-agent systems. Here, we investigated the connections between ATL and the Game Description Language, a declarative language specifically intended for defining games. We first demonstrated that GDL can be understood as a specification language for ATL models, and subsequently that it is possible to succinctly characterise GDL game descriptions directly as ATL formulas, and that, as a corollary, the problem of interpreting ATL formulas over GDL descriptions is EXPTIME-complete. Our work, we believe, could be applied to formal verification of GDL specifications: the GDL game designer can express properties of games using ATL, and then automatically check whether these properties hold of their GDL specifications.

Acknowledgment We would like to thank Dirk Walther and two anonymous reviewers for comments and helpful suggestions.

BIBLIOGRAPHY

[AHK02] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.

[AHKV98] R. Alur, T. Henzinger, O. Kupferman, and M. Vardi. Alternating refinement relations. In *International Conference on Concurrency Theory*, pages 163–178, 1998.

[Apt90] K. Apt. Introduction to logic programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 494–574. Elsevier, 1990.

[CGP00] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 2000.

[GL05] M. Genereseth and N. Love. General game playing: Overview of the AAAI competition. Technical report, Stanford University, Stanford, 2005.

[HKV97] David Harel, Orna Kupferman, and Moshe Y. Vardi. On the complexity of verifying concurrent transition systems. In *International Conference on Concurrency Theory*, pages 258–272, 1997.

[HLW05] W. van der Hoek, A. Lomuscio, and M. Wooldridge. On the complexity of practical ATL model checking. In *Proceedings AAMAS*, 2006.

[SC79] L. J. Stockmeyer and A. K. Chandra. Provably difficult combinatorial games. *SIAM Journal of Computing*, 8(2):151–174, 1979.

[WLWW06] D. Walther, C. Lutz, F. Wolter, and M. Wooldridge. ATL satisfiability is indeed ExpTime-complete. *Journal of Logic and Computation*, 16:765–787, 2006.