Journal of Logic and Computation Advance Access published August 5, 2009

Verification of Games in the Game Description Language

JI RUAN, WIEBE VAN DER HOEK and MICHAEL WOOLDRIDGE,
Department of Computer Science, University of Liverpool, Liverpool L69 3BX, UK.
E-mail: j.ruan@csc.liv.ac.uk; wiebe@csc.liv.ac.uk; mjw@csc.liv.ac.uk

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 the ability to play a specific game. Participants in the competition are provided with a previously unknown game specified in GDL, and are required to dynamically and autonomously determine how best to play this game. Recently, there has been much interest in the use of strategic cooperation logics for reasoning about game-like scenarios—the Alternating-time Temporal Logic (ATL) of Alur, Henzinger, and Kupferman is perhaps the best known example. Such logics are specifically intended to support reasoning about game-theoretic properties of multi-agent systems. In short, the aim of this article is to make a concrete link between ATL and GDL, with the ultimate goal of using ATL to reason about GDL-specified games. We make the following contributions. First, we demonstrate that GDL can be understood as a specification language for ATL models, and prove that the problem of interpreting ATL formulae over propositional GDL descriptions is EXPTIME-complete. Second, we use ATL to characterize a class of 'fair playability' conditions, which might or might not hold of various games.

Keywords: Verification, general game playing, game description language, alternating-time temporal logic, 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 Artificial Intelligence (AI). However, a common objection to such specialized competitions and dedicated game playing systems is that they explore only one very narrow aspect of intelligence and rationality. To overcome these objections, the American Association for Artificial Intelligence (AAAI) introduced in 2005, a *general game playing competition*, intended to test *the ability to play games in general*, rather than just the ability to play a specific game [12, 23]. 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 fares best overall. Note that the participant programs are required to interpret the rules of the games *themselves*, without human intervention or interpretation. The *Game Description Language* (GDL) is a special purpose, computer processable language, which was developed in order to define the games to be played by participant programs. Thus, a participant program must be able to interpret game descriptions expressed in GDL, and then autonomously play the game defined by the GDL description.

Since GDL is a language for defining games, it seems very natural to investigate the problem of reasoning about games defined in GDL. Just as the designer of a computer communications protocol might want to use model checking tools to investigate the properties of the protocol (ensure

doi:10.1093/logcom/exp039

it is deadlock-free, etc. [6]), so also the GDL game designer will typically want to investigate the properties of games. In addition to checking protocol-like properties such as deadlock-freeness, the fact that GDL is used for describing *games* suggests a whole new class of properties to check: those relating to the *strategic properties* of the game being defined.

One formalism for reasoning about games that has attracted much interest is *Alternating-time Temporal Logic* (ATL) [3]. 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 multiagent systems in recent years, and has proved to be a powerful and expressive tool for this purpose [3, 14, 21, 22, 27, 28].

In this article, we make a concrete link between ATL and GDL. Specifically, we show that GDL descriptions can be interpreted as specifications of an ATL model, and that ATL can thus be interpreted over GDL descriptions. The main contribution of this work is that we show it is possible to translate propositional GDL descriptions into ATL formulae that are 'equivalent' in a sense to be precisely defined later, and which are only polynomially larger than the original GDL description. As a corollary, we are able to characterize the complexity of ATL reasoning about propositional GDL games: the problem is EXPTIME-complete.

Apart from its theoretical interest, we also want to explore this topic more practically. We try to answer the following questions:

- What are the conditions which characterize when a given GDL description defines a (meaningful) game? We refer to these properties as *playability* conditions. Although some important playability conditions have been discussed in the GDL literature [12, 17], these conditions are in fact very minimal.
- How can we check whether a game specified in GDL satisfies such playability conditions?

The main *application* of our work, we believe, is in having an approach to analysing the properties of games described using GDL: the GDL game designer can express desirable properties of games using ATL, and then automatically check whether these properties hold of their GDL descriptions.

The article is structured as follows. In the following two sections, we introduce first the GDL and then ATL. In Section 4, we establish a link between GDL descriptions and ATL specifications, and characterise the complexity of interpreting an ATL specification with respect to a GDL description. In Section 5, we give a systematic classification of GDL playability conditions, and show how these conditions may be characterized as ATL formulae. This classification extends considerably the discussion and formalization of playability conditions given in [12, 17, 23]. We conclude with a brief discussion, pointers to related work and some directions for future work.

2 GDL and game models

GDL is a specialized language, intended for defining games [12, 17]. A game description must define the states of the game, a unique initial state and the players in the game ('roles' in GDL parlance). For every state and every player, the game description must define the moves (a.k.a. actions) available to that player in that state, as well as 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 is to use a *logical* definition of the game. We introduce GDL by the 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.

```
01 (<= (role xplayer))
02 (<= (role oplayer))
03 (<= (init (cell 1 1 b)))
    ...
11 (<= (init (cell 3 3 b)))
12 (<= (init (control xplayer)))
13 (<= (next (cell ?m ?n x))
14     (does xplayer (mark ?m ?n))
15     (true (cell ?m ?n b)))
    ...
28 (<= (next (control oplayer))
29     (true (control xplayer)))
30 (<= (row ?m ?x)
31     (true (cell ?m 1 ?x))
32     (true (cell ?m 2 ?x))
33     (true (cell ?m 3 ?x)))
   ...
54  (<= (legal ?w (mark ?x ?y))
55      (true (cell ?x ?y b))
56      (true (control ?w)))
57  (<= (legal oplayer noop)
58      (true (control xplayer)))
    ...
61  (<= (goal xplayer 100)
62      (true (line x)))
    ...
77  (<= terminal
78      (line x))
79  (<= terminal
80      (line o))
81  (<= terminal
82      (not open))
```

FIGURE 1. A fragment of a game in the GDL.

GDL uses a prefix rule notation based on LISP. The Tic-Tac-Toe game in Figure 1 consists of 82 lines. The first two lines, (`role xplayer`) and (`role oplayer`), define the two players in this game. The following `init` lines (lines 03–12) 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 (lines 13–15) defines the effect of making a move: if $cell(m,n)$ is blank (`cell ?m ?n b`), and `xplayer` marks it, then in the `next` state, it will be true that $cell(m,n)$ is marked by `x`: (`cell ?m ?n x`). The next rule (lines 28–29) says that if the current state is under the control of `xplayer`, then the next state will be under the control of `oplayer`. Lines 30–33 define what it means to have a `row` of symbols (we omit a number of related rules). The `legal` rule (lines 54–56) defines when it is `legal` for a player `?w` to perform a mark action. The `goal` rule (lines 61–62) 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 (lines 77–82) define when the game has ended.

Overall, a GDL description consists of a list of such rules. As we will see below, the semantics of these rules are similar to the semantics of rules in logic programming languages. Certain operators in a GDL description have a special meaning: `role` (used to define the players of the game); `init` (defining initial facts); `legal` (defining preconditions 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.

While GDL in [12, 17] permits predicates such as (`cell ?m ?n b`), we simplify this by allowing only nullary predicates, i.e. propositions. We can do this via instantiation of the predicates, i.e. replacing variables with their values. For example, variables like `?m`, `?n` are replaced by elements in their domain $\{1,2,3\}$. Thus, (`cell ?m ?n b`) is instantiated as (`cell 1 1 b`), (`cell 1 2 b`), …, (`cell 3 3 b`). It is easy to see that the rule in (lines 13–15) is replaced by nine rules with no predicates, and in general, there will inevitably be an undesirable blow-up in the number of rules when translating from arbitrary predicate form; nevertheless, the translation is possible, a point that is implicitly used in what follows. We refer to an expression such as (`cell 1 1 b`) as a nullary predicate, or an atomic proposition. We will refer to our fragment of GDL as propositional GDL in the remainder of the article.

In what follows, we will formally define the interpretation of GDL descriptions with respect to game models. As GDL is based on DATALOG, a logic programming language, we begin by introducing DATALOG.

2.1 DATALOG program and its semantics

DATALOG is a query and rule language for deductive databases that, syntactically, is a subset of PROLOG [7]. GDL uses DATALOG as a basis to specify game rules. When we build GDL models (Section 2.3) and ATL-theories (Section 4.2) from game descriptions, DATALOG will be used in the background as follows. A game description (Section 2.2) specifies what is true in the initial state, what should hold throughout the game, and what should be the effect of performing certain actions. To compute the propositional content of a state, the DATALOG semantics will play a crucial role: it offers, as a side effect, a solution to the *frame problem* [20]: only what is explicitly specified to be true (initially, or following from a global constraint, or as the effect of an action) will become true.

As we mentioned above, we deal with propositional GDL. Accordingly, we only introduce the propositional fragment of DATALOG (see [5] for a more extended description).

DEFINITION 2.1 (DATALOG: language, rules and programs)
The basic unit of the DATALOG Language is a set of atomic propositions $\Pi = \{p, q, \ldots\}$. Let $\ell(\Pi)$ be the set of literals over Π: $\ell(\Pi) = \Pi \cup \{\neg p \mid p \in \Pi\}$.

A DATALOG rule is of the form $(\Leftarrow(p)(\ell_1)\ldots(\ell_n))$ 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 the set $\{\ell_1, \ldots, \ell_n\}$. Note that a body can be empty.

A DATALOG program is a set of DATALOG rules.

A model for a DATALOG program is a set of atomic propositions, with the intended meaning that these are the atoms that are true. We make this precise in the following definition, which shows that a rule $(\Leftarrow(p)(\ell_1)\ldots(\ell_n))$ can roughly be interpreted as an implication $(\ell_1 \wedge \cdots \wedge \ell_n) \rightarrow p$.

DEFINITION 2.2 (Models for DATALOG programs)
Given a DATALOG program Δ, a set of atoms $\Sigma \subseteq \Pi$ is a model of Δ if and only if it satisfies the following conditions:

- if $(\Leftarrow(p)) \in \Delta$, then $p \in \Sigma$;
- if $(\Leftarrow(p)\ bd) \in \Delta$ and $pos(bd) \subseteq \Sigma$ and $neg(bd) \cap \Sigma = \emptyset$, then $p \in \Sigma$, where $pos(bd)$ is the set of positive literals in bd and $neg(bd)$ is the set of negative literals.

It is easy to verify that under these semantics (as under conventional logical semantics), $\Delta_1 = \{(\Leftarrow(p)(q))\}$ has three models: $\{p, q\}$, $\{p\}$ and $\{\}$. However, DATALOG rules are not simply classical implications: only if there is a 'reason' to accept an atom p, will it be made true. Under this approach, in Δ_1, since q is not true, there is no reason to accept p, so the only model for Δ_1 would be $\{\}$. As a slightly more involved example, if we have a rule $r_1 = (\Leftarrow(p)(q)(\neg s))$, then, in order to accept p, we should have accepted q, and we should have no reason to accept s. But, of course, this can lead to cycles in reasoning: suppose we also have $r_2 = (\Leftarrow(q))$. Then one would expect we conclude p, but *not* if acceptance of p would undermine one of its reasons to accept it, i.e. if we also have a rule $r_3 = (\Leftarrow(s)(p))$, accepting p would undercut one of the reasons (namely $\neg s$) for accepting it. The following definition characterizes conditions under which such circular reasoning is guaranteed not to occur, thereby guaranteeing unique models for programs.

DEFINITION 2.3 (Dependency graphs for DATALOG programs)
Let a DATALOG program Δ be given. Its *Dependency Graph* $\mathcal{DG}(\Delta)$ is a labeled directed graph $\langle \Pi, lab^+, lab^- \rangle$, where

- Π is the set of atoms in Δ; each atom serves as a node in the graph.

- $lab^+(p,h)$ (i.e. an edge from p to h labeled with $+$) iff there is a rule $r \in \Delta$ with $h = hd(r)$ and $p \in bd(r)$.
- $lab^-(p,h)$ iff there is a rule $r \in \Delta$ with $h = hd(r)$ and $\neg p \in bd(r)$.

A dependency graph helps us to keep track of what we need to know before deciding whether an atom $p \in hd(r)$ will be accepted in a model, or not. In the case of Δ_1, we have $lab^+(q,p)$, and in case of $\Delta_2 = \{r_1, r_2, r_3\}$, we have $lab^+(q,p), lab^-(s,p)$ and $lab^+(p,s)$.

DEFINITION 2.4 (Stratified DATALOG programs)
A DATALOG program Δ is called *stratified* if its dependency graph $\mathcal{DG}(\Delta)$ contains *no cycles with a '−' label*. An atom p is said to be in stratum $i \in \mathbb{N}$ if the maximum number of edges labelled '−' on any path ending at $p \in \mathcal{DG}(\Delta)$ is i. A rule $r \in \Delta$ is of stratum i if $hd(r)$ is in stratum i. A DATALOG program Δ is called *acyclic* if $\mathcal{DG}(\Delta)$ contains no cycles.

Note that in our example, Δ_1 is stratified, but Δ_2 is not: there is a cycle containing the vertices p and s.

DEFINITION 2.5 (DATALOG semantics for stratified DATALOG programs)
Given a stratified DATALOG program Δ, we construct a model $s = DatlogPMod(\Delta)$ as follows. First of all, let $t_0 = \{p \mid (\Leftarrow p) \in \Delta\}$. Suppose t_i is defined, initialise s_i to t_i and, as long as there is a rule $(\Leftarrow (p)(\ell_1)\ldots(\ell_n))$ in stratum i such that $s_i \models \ell_1 \wedge \cdots \wedge \ell_n$, add p to s_i. After this, put $t_{i+1} = s_i$. If the maximum stratum of Δ is k, put $s = t_{k+1}$. We will call $DatlogPMod(\Delta)$ the DATALOG semantics of Δ.

Going back to our examples, we indeed obtain a unique model for Δ_1, i.e. $DatlogPMod(\Delta_1) = \{\}$. This shows that not all models for a DATALOG program in the sense of Definition 2.2 qualify as a model under the DATALOG semantics $DatlogPMod(\Delta)$ for a program Δ. But it is easy to see that in general we have that a model under the DATALOG semantics for Δ is indeed a model for Δ. 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. This cannot be done for $\neg s$ in the cyclic program Δ_2, but consider $\Delta_3 = \{r_1, r_2\}$: this program is stratified: the atoms q and s are in stratum 0, and p is in stratum 1. Hence, rule r_2 is of stratum 0 and r_1 is of stratum 1. So the model construction first considers r_2: it adds q to the model. When we consider rule r_1, we have moved to stratum 1, which means that we can now assume that s (of stratum 0) will not become true anymore, and the procedure sets p to true. In summary, we have $DatlogPMod(\Delta_3) = \{q, p\}$.

The fact that a program Δ is acyclic is equivalent (see [5]) to requiring that there is a level mapping $f : \ell(\Pi) \to \mathbb{N}$ for which $f(p) = f(\neg p)$ and for every rule $(\Leftarrow (p)(\ell_1)\ldots(\ell_n))$ in Δ, $f(p) > f(\ell_i)$, for all $i \leq n$. This again says that once we consider a rule r, we may assume that all the atoms in its body have obtained a definite truth value.

THEOREM 2.1 ([5])
The procedure of Definition 2.5 yields a unique model for Δ, and it does not depend on the particular stratification.

Suppose we have a program Δ in which all the rules r_i with $hd(r_i) = p$, are $r_1, r_2, \ldots, r_n$. Interpreting rules as implications says that when one of the bodies is true, the head p should also be true. Under the more constructive interpretation, possible for stratified programs, it means: *the only way* to make p true, is to have one of the bodies of the rules $r_1, \ldots, r_n$ true. This is captured by the following definition and result.

DEFINITION 2.6 (Completion of Δ)
Given an acyclic DATALOG program Δ, the completion of Δ is a set of formulae $CP(\Delta)$ 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

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

where $bd(r)=\{\ell_1,\ldots,\ell_n\}$ and $\bigwedge bd(r)=\ell_1\wedge\cdots\wedge\ell_n$; for every empty body $bd(r)$, $\bigwedge bd(r)=\top$. 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 2.2 ([5])
Let Δ be an acyclic program, and Π be the set of atoms in Δ. We have

$$\forall p\in\Pi, p\in DatlogPMod(\Delta) \quad \text{iff} \quad CP(\Delta)\models_{cl} p,$$

where $DatlogPMod(\Delta)$ is the unique model of the stratified program Δ, the set $CP(\Delta)$ is the Clark completion of Δ and $\models_{cl}$ denotes the consequence in classical logic.

We already noted $DatlogPMod(\Delta_1)=\{\}$ and indeed, the Clark Completion of this program is $CP(\Delta_1)=\{q\leftrightarrow\bot, p\leftrightarrow q\}$. For Δ_2, we had $DatlogPMod(\Delta_2)=\{q,p\}$ and $CP(\Delta_2)=\{q\leftrightarrow\top, s\leftrightarrow\bot, p\leftrightarrow(q\wedge\neg s)\}$.

Theorem 2.2 links the DATALOG semantics of Definition 2.5 of a DATALOG program Δ to a classical interpretation of formulas obtained from Δ. We will use this as follows. A game description Γ will be a special kind of DATALOG program (Section 2.2). When building a model G_Γ for such a game description Γ, we will use the DATALOG semantics $DatlogPMod(\Gamma)$ (Section 2.3). However, when building an ATL-theory for Γ, (Section 4.2) we will use the completion $CP(\Gamma)$ of Γ. Theorem 2.2 helps us ensure that both constructions yield the same consequences.

The following observation is straightforward, and will be used later on.

OBSERVATION
Adding a fact $\Leftarrow(p)$ preserves the property of being stratified, i.e. if Δ is stratified, then so is $\Delta\cup\{(\Leftarrow(p))\}$.

2.2 Game descriptions

We now formally define GDL game descriptions.

DEFINITION 2.7 (GDL Syntax)
Let a primitive set of proposition symbols $Prim=\{\hat{\mathtt{p}},\hat{\mathtt{q}}\ldots\}$, a set of agents Ag, a set of actions Ac, a set of strings S and a set of integers $[0\ldots100]$[1] be given. The set of *atomic propositions* of GDL, denoted At_{GDL}, is defined as the smallest set that satisfies the following conditions:

- $Prim\subseteq At_{\text{GDL}}$;
- a special atom `terminal` $\in At_{\text{GDL}}$;
- for two strings $\mathtt{s}_1,\mathtt{s}_2\in S$, (`distinct` $\mathtt{s}_1$ $\mathtt{s}_2$) $\in At_{\text{GDL}}$;
- for every agent $\mathtt{i}\in Ag$ and action $\mathtt{a}\in Ac$, (`legal` i a) $\in At_{\text{GDL}}$;

[1] This set of integers is used to indicate the payoffs of agents in each state, following [17]. The range $0\ldots100$ is arbitrary.

- for every agent `i` and integer `v` in [0 ... 100]), (`goal i v`) $\in At_{\text{GDL}}$.

The set of *atomic expressions* $AtExpr_{\text{GDL}}$ of GDL, is defined as the smallest set that satisfies the following conditions:

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

$LitAt_{GDL}$ is {`p`,(`true p`),(`not p`),(`not (true p`)) | `p` $\in At_{GDL}$}. $LitExpr_{\text{GDL}}$ is $AtExpr_{\text{GDL}} \cup LitAt_{GDL}$.

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. The meaning of these atomic expressions will be given in Definition 2.10.

DEFINITION 2.8 (Game Descriptions)
A GDL game description Γ is a set of DATALOG rules r of the form [2] ($\Leftarrow$(`h`)(`e`$_1$)...(`e`$_m$)) where `h`, the head $hd(r)$ of the rule, is an element of $AtExpr_{\text{GDL}}$ and each `e`$_i$ ($i \in [1...m]$) in the body $bd(r)$ of r is a literal expression from $LitExpr_{\text{GDL}}$. If $m=0$, we say that r has *an empty body*. We can split every game description Γ into four different types of rules where:

- $\Gamma_{\texttt{role}}$ contains all claims of the form ($\Leftarrow$(`role x`)). They specify the agents in the game.
- $\Gamma_{\texttt{init}}$ is a set of constraints of the form ($\Leftarrow$(`init p`)). They have an empty body and their heads represent constraints of the initial state of the game.
- $\Gamma_{\texttt{glob}}$ is a set of global constraints, i.e. rules of the form ($\Leftarrow$(`p`) (`e`$_1$) ... (`e`$_m$)), where `p` $\in At_{GDL}$ and each body `e`$_i$ ($i \in [1...m]$) is from $LitAt_{GDL}$.
- $\Gamma_{\texttt{next}}$ contains all rules with a (`next p`) in the head: ($\Leftarrow$(`next p`)(`e`$_1$)...(`e`$_m$)) where each `e`$_i$($i \in [1...m]$) is from $LitAt_{GDL}$ or of the form (`does i a`).

Finally, we will assume that game descriptions Γ are stratified in the sense of Definition 2.4.

2.3 Game models

In general, a game model can be seen as a game tree, where we have a set of nodes, representing states of the game, and a labelled edge from one state to another representing a transition between those states, caused by the performance of actions/moves by players. We will shortly consider how to compute such game states from game descriptions.

For the description of game models G, our approach is equivalent to that of [12]. Instead of *roles* we will refer to a set $Ag = \{1, \ldots, n\}$ of *agents* or *players*.

DEFINITION 2.9 (GDL game model)
Given the set of atomic propositions At_{GDL}, a *GDL Game Model* is a structure:

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

where S is a finite, non-empty set of game states; $s_0 \in S$ is the initial state of G; Ag is a finite, non-empty set of agents, or players in the game; Ac_i is a finite non-empty set of possible actions or moves

[2] We do not allow disjunctions in the body of a GDL rule, but note that this is not a restriction: a rule such as ($\Leftarrow$(`h`)(`e`$_1 \vee$ `e`$_2$)) can be captured by two rules ($\Leftarrow$(`h`)(`e`$_1$)) and ($\Leftarrow$(`h`)(`e`$_2$)).

for agent i; $\tau : Ac_1 \times \cdots \times Ac_n \times S \to S$ is a partial function 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 change to its successor state u—we require all states, except the initial state, to have only one predecessor; and finally, $\pi : S \to 2^{At_{\text{GDL}}}$ is an interpretation function, which associates with each state the set of atomic propositions in At_{GDL} that are true in that state. We will often abbreviate an action profile $\langle a_1, \ldots, a_n \rangle$ to $\vec{a}$.

Our game models are essentially those of [12]. However, we encode notions like being a terminal state, or a legal action, through π, rather than through separate relations in the model.

Now we specify when a game model G is a model for a game description Γ; this makes precise the informal description of [12], and in fact represents a formal semantics for GDL.

We compute the game model *GMod* for a game description Γ as follows. The main idea is that every state $s \in S$ of *GMod* 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 $\Gamma_{\texttt{glob}}$ does not contain `init` or `next` in any body of any rule, $\delta(\Gamma_{\texttt{glob}})$ is indeed a DATALOG program. Moreover, $\delta(\Gamma_{\texttt{glob}})$ is stratified whenever $\Gamma_{\texttt{glob}}$ is. Also, let $\delta(\Gamma_{\texttt{init}})$ be $\{\Leftarrow \texttt{p} \mid (\Leftarrow \texttt{init p}) \in \Gamma_{\texttt{init}}\}$. The set Ag of agents, and Ac_i of actions for agent i in *GMod* are immediately read off from Γ: $Ag = \{i \mid (\Leftarrow \texttt{(role i)}) \in \Gamma_{\texttt{role}}\}$ and $Ac_i = \{\texttt{a} \mid \texttt{(legal i a)}$ occurs in $\Gamma\}$.

In the following, we construct S, τ and π step by step.

- *First*, we define the initial state s_0. Put

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

  Since $\delta(\Gamma_{\texttt{glob}})$ is stratified, by our observation ending in Section 2.1, we apply *DatlogPMod*() to a stratified program.
- *Next*, suppose a game state $s \in S$ has already been defined. If this is not a terminal state, i.e. $\texttt{terminal} \notin \pi(s)$, 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 $(\texttt{legal i a}_\texttt{i}) \in \pi(s)$. If $\texttt{terminal} \notin \pi(s)$, we define, for every profile of legal actions $\langle a_1, \ldots, a_n \rangle$, a successor u of s by first computing the atoms that need to be true due to $\Gamma_{\texttt{next}}$.

$$\begin{aligned} F_\Gamma(\langle a_1, \ldots, a_n \rangle, s) = \{ \Leftarrow \texttt{p} \mid\ & \exists\, (\Leftarrow (\texttt{next p})\, (\texttt{e}_1) \cdots (\texttt{e}_m)) \in \Gamma_{\texttt{next}} \\ & \& \ \pi(s) \cup \{\neg \texttt{q} \mid \texttt{q} \notin \pi(s)\} \\ & \cup \{(\texttt{does i a}_i) \mid i \in [1 \ldots n]\} \models_{cl} \texttt{e}_1 \wedge \cdots \wedge \texttt{e}_m \} \end{aligned}$$

  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 a new state u to G and stipulate:

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

  Again, using the observation at the end of Section 2.1, u is well-defined.
- *Iteration:* we repeat the above procedure to all the descendents of the initial state, until we reach all the terminal states.

We illustrate the main idea with the Tic-Tac-Toe example. Suppose that we already have a propositional version of the GDL description presented in Figure 1, i.e. all the variables have been instantiated. As $(\texttt{control xplayer}) \in \delta(\Gamma_{\texttt{init}})$, we use $\Gamma_{\texttt{glob}}$ and 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` are not 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`) $\in F_\Gamma(\vec{a}, s_0)$, due to the `next` rules.

Atoms of the form (`does i` a_i) are not added to the game model *GMod*—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* is indeed a game model for Γ, if we give the following truth definitions.

DEFINITION 2.10 (GDL semantics)
Let $G = \langle S, s_0, Ag, Ac_1, \ldots, Ac_n, \tau, \pi \rangle$ 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 any agent 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 model G and state s, we define:

- $G, s \models_{\text{GDL}}$ `p` iff `p` $\in \pi(s)$, for any `p` $\in At_{\text{GDL}}$;
- $G, s \models_{\text{GDL}}$ `not p` iff $G, s \not\models_{\text{GDL}}$ `p`;
- $G, s \models_{\text{GDL}}$ `true p` iff $G, s \models_{\text{GDL}}$ `p`;
- $G, s \models_{\text{GDL}}$ `not (true p)` iff $G, s \not\models_{\text{GDL}}$ `true p`;
- $G \models_{\text{GDL}}$ `init p` iff $G, s_0 \models_{\text{GDL}}$ `p`;
- $G \models_{\text{GDL}}$ (`<=` (`p`) (e_1)…(e_m)) iff $\forall s : (\forall i \in [1 \ldots m] : G, s \models_{\text{GDL}} e_i) \Rightarrow G, s \models_{\text{GDL}}$ `p`;
- $G \models_{\text{GDL}}$ (`<=` (`next p`)(e_1)…(e_m)(`does` i_1 a_1)…(`does` i_k a_k)) iff $\forall s, t$: $(\forall i \in [1 \ldots m]$: $G, s \models_{\text{GDL}} e_i$ and t is an $i_1 : a_1, \ldots i_k : a_k$ successor of $s) \Rightarrow G, t \models_{\text{GDL}}$ `p`.

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

3 Alternating-time temporal logic

We now introduce a logic for reasoning about games defined using GDL. For this, we believe ATL is ideally suited [3]. 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 formulae are built using the unary operators $\bigcirc$, $\Box$, $\Diamond$ and $\mathcal{U}$, where $\bigcirc$ means 'in the next state', $\Box$ means 'always', $\Diamond$ means 'eventually' and the binary operator $\mathcal{U}$ means 'until'.

DEFINITION 3.1 (Language of ATL)
The language of ATL (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.

3.1 Semantics

ATL has been provided with a number of semantics, the main three of them are proven to be equivalent in [15]. Since *moves*, or *actions*, play such a prominent role in game playing, we use *Action-based Alternating Transition Systems* (AATSs) (which, for the language of ATL, are equivalent to, for instance, Concurrent Game Structures).

DEFINITION 3.2
An 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 $i \neq j \in Ag$; $\rho : Ac_{Ag} \to 2^Q$ is an *action precondition function*, which for each action $a \in Ac_{Ag} (= \bigcup_{i \in Ag} Ac_i)$ defines the set of states $\rho(a)$ from which a may be executed; $\tau : Ac_1 \times \cdots \times Ac_n \times Q \to Q$ 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 $\pi : Q \to 2^\Phi$ 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.

In order to be consistent with GDL, in this article, we require 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: $\forall q, \forall \vec{a} = \langle a_1, \ldots, a_n \rangle, (\vec{a}, q) \in \text{dom } \tau$ 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 define a *strategy* for an agent $i \in Ag$ as a function $\sigma_i : Q \to Ac_i$ which must satisfy the *legality* constraint that $\sigma_i(q) \in options(i, q)$ for all $q \in Q$. In this definition, a strategy is memoryless in the sense that an action is chosen only for states, not for a history of states. 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.

In a nutshell, it should be clear that game models (Definition 2.9) and AATSs (Definition 3.2) are very similar: the basic difference is that a game model has terminal states without any outgoing transitions.

We can now give the rules defining the satisfaction relation '$\models_{\text{ATL}}$' 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 formulae of ATL.

DEFINITION 3.3 (Semantics of ATL)
Given an $\mathcal{A}$, a state q, the semantics of ATL is defined as follows:

$\mathcal{A}, q \models_{\text{ATL}} p$ iff $p \in \pi(q)$ (where $p \in \Phi$);
$\mathcal{A}, q \models_{\text{ATL}} \neg\varphi$ iff $\mathcal{A}, q \not\models_{\text{ATL}} \varphi$;
$\mathcal{A}, q \models_{\text{ATL}} \varphi \vee \psi$ iff $\mathcal{A}, q \models_{\text{ATL}} \varphi$ or $\mathcal{A}, q \models_{\text{ATL}} \psi$;
$\mathcal{A}, q \models_{\text{ATL}} \langle\langle C\rangle\rangle \bigcirc\varphi$ iff $\exists \sigma_C \in \Sigma_C$, such that $\forall \lambda \in comp(\sigma_C, q)$, we have $\mathcal{A}, \lambda[1] \models_{\text{ATL}} \varphi$;
$\mathcal{A}, q \models_{\text{ATL}} \langle\langle C\rangle\rangle \Box\varphi$ iff $\exists \sigma_C \in \Sigma_C$, such that $\forall \lambda \in comp(\sigma_C, q)$, we have $\mathcal{A}, \lambda[u] \models_{\text{ATL}} \varphi$ for all $u \in \mathbb{N}$;
$\mathcal{A}, q \models_{\text{ATL}} \langle\langle C\rangle\rangle \varphi \mathcal{U} \psi$ 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 $\mathcal{A}, \lambda[u] \models_{\text{ATL}} \psi$, and for all $0 \leq v < u$, we have $\mathcal{A}, \lambda[v] \models_{\text{ATL}} \varphi$.

The remaining classical logic connectives ('$\wedge$', '$\rightarrow$', '$\leftrightarrow$') are assumed to be defined as abbreviations in terms of $\neg, \vee$, in the conventional manner, and $\langle\langle C\rangle\rangle \Diamond\varphi$ is defined as $\langle\langle C\rangle\rangle \top \mathcal{U} \varphi$. For readability, we omit set brackets in cooperation modalities, for example, writing $\langle\langle 1\rangle\rangle$ instead of $\langle\langle\{1\}\rangle\rangle$ and writing $\langle\langle\rangle\rangle$ instead of $\langle\langle\{\}\rangle\rangle$. Finally, we write $\mathcal{A} \models_{\text{ATL}} \varphi$ for $\mathcal{A}, q_0 \models_{\text{ATL}} \varphi$.

We now give an equivalence relation between two AATSs, called *Alternating Bisimulation*. The purpose is to characterize the AATS structures that cannot be distinguished by ATL formulas. Here by 'distinguish', we mean there exists an ATL formula such that it is true in one structure but false in another structure. The following definition is based on [2].

DEFINITION 3.4 (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 AATSs. Then, a relation $\mathcal{R} \subseteq Q_1 \times Q_2$ is called an *alternating bisimulation* if $\mathcal{R} q_1 q_2$ and, for every two states t_1 and t_2 for which $\mathcal{R} t_1 t_2$,we have:

- **Invariance:** for all $p \in \Phi$, $p \in \pi(t_1)$ iff $p \in \pi(t_2)$.
- **Zig:** 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 $\mathcal{R} t_1' t_2'$.
- **Zag:** 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 3.1 ([2])
Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be such that there is an alternating bisimulation $\mathcal{R}$ between them, with $\mathcal{R} q_1 q_2$. Then, for all ATL formulae φ:

$$\mathcal{A}_1, q_1 \models_{\text{ATL}} \varphi \quad \Leftrightarrow \quad \mathcal{A}_2, q_2 \models_{\text{ATL}} \varphi.$$

4 Linking GDL and ATL

From previous sections, we can see that GDL and ATL are intimately related: GDL is a language for defining games, while ATL is a language for expressing properties of such games. The difference between the two languages is that GDL takes a relatively *constructive*, *internal* approach to a game

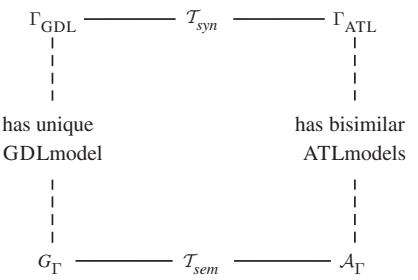

FIGURE 2. The relation between a GDL description of a game Γ_{GDL} and its related ATL theory Γ_{ATL}.

description, essentially defining how states of the game are constructed and related by possible moves. In contrast, ATL takes an *external*, *strategic* view: while it seems an appropriate language with which to express potential strategic properties of games, it is perhaps not very appropriate for defining games.

In this section, we will answer the following question: how complex is it to interpret a property, represented by an ATL formula, over a game represented by a GDL description? We do this by building two links between GDL and ATL:

- On the semantic level, every GDL description Γ has an ATL *model* associated with it.
- On the syntactic level, every GDL description Γ has an ATL *theory* associated with it.

Let us now be more precise about the links between GDL and ATL (cf. Figure 2). We start from any game G with GDL description Γ_{GDL}. On the semantic level, we use our procedure from Section 2.3 to construct a unique model $G_\Gamma = GMod(\Gamma)$ from Γ_{GDL}. This model has an associated ATL model $\mathcal{A}_\Gamma = \mathcal{T}_{sem}(G_\Gamma)$. In Section 4.1, we introduce and explain $\mathcal{T}_{sem}$. On the syntactic level, we provide a translation $\mathcal{T}_{syn}$ that transforms the GDL specification Γ_{GDL} into an ATL theory $\Gamma_{ATL} = \mathcal{T}_{syn}(\Gamma_{GDL})$. We further show that this transformation is correct, in the following sense: all ATL-models satisfying Γ_{ATL} are bisimilar to $\mathcal{A}_\Gamma$. So Γ_{ATL} can be said to characterize the ATL theory of the game G. And, one has, for any GDL formula γ, that $G_\Gamma \models_{GDL} \gamma$ iff $\mathcal{T}_{sem}(G_\Gamma) \models_{ATL} \mathcal{T}_{syn}(\gamma)$, where $\models_{GDL}$ denotes the semantics of GDL and $\models_{ATL}$ denotes the semantics of ATL.

We now explore these two links in more detail.

4.1 From GDL Game models to ATL Game models: $\mathcal{T}_{sem}$

Suppose that we have already constructed a game model *GMod* from a GDL description Γ, using the methods in Section 2.3. It is not yet possible to interpret an ATL formula on this model *GMod*. In this subsection, we transform *GMod* (or indeed, any game model G) into an AATS, the ATL game structure on which we can interpret ATL formulae.

Given a GDL game model $G = \langle S, s_0, Ag, Ac_1, \ldots, Ac_n, \tau, \pi \rangle$ and a set of atomic propositions At_{GDL}, we can define an associated AATS $\mathcal{T}_{sem}(G) = \mathcal{A}_G = \langle Q, q_0, Ag, Ac_1 \cup \{fin_1\}, \ldots, Ac_n \cup \{fin_n\}, \rho, \tau', \Phi, \pi' \rangle$ with the same sets of agents Ag and such that Φ is constructed from At_{GDL} in the following manner. If the game model G is a model for a description Γ, we will also write $\mathcal{A}_\Gamma$ for $\mathcal{A}_{GMod}$.

DEFINITION 4.1 (Translation t and t_{old})
Define a translation $t : At_{\text{GDL}} \to At_{\text{ATL}}$, where we associate every atom in At_{GDL} with an atom in At_{ATL}.

$$\begin{array}{lll} t(\hat{\mathtt{p}}) & = \hat{p} \ (\hat{p} \in Prim) & t(\mathtt{goal\ i\ v}) \ = goal(i,v) \\ t(\mathtt{legal\ i\ a}) & = legal(i,a) & t(\mathtt{terminal}) = terminal \\ t(\mathtt{distinct\ s_1\ s_2}) & = distinct(s_1,s_2) & \end{array}$$

Let t_{old} be as follows: $t_{old}(\mathtt{p}) = t(\mathtt{p})_{old} = p_{old}$, for any $\mathtt{p} \in At_{\text{ATL}}$.

We add four types of atomic propositions to Φ:

(1) Atoms representing the current state of the game: for every $\mathtt{p}$ in At_{GDL}, add $t(\mathtt{p})$ to Φ.
(2) Atoms representing the previous state of the game: for every $\mathtt{p}$ in At_{GDL}, add $t(\mathtt{p})_{old}$ to Φ.
(3) Atoms representing actions that are done in the transition from previous state to current state: add atom $done(i,a)$ to Φ for each $(\mathtt{does\ i\ a})$.
(4) Atoms distinguishing the initial and end states of the game: add *init* for initial state and a special atom $\mathtt{s}_\perp$. The atom $\mathtt{s}_\perp$ denotes a special kind of states, called 'sink states', which we add to $\mathcal{A}_G$ in order to make it a proper AATS. The idea is that in game model G, a terminal state does not have any successors, but in $\mathcal{A}_G$, a sink state is the only successor of a terminal state and itself.

The other elements of $\mathcal{A}_G$ are:

- $Q = Q_1 \cup Q_2$, where $Q_1 = S$, and $Q_2 = \{\mathtt{s}_q \mid q \in Q_1\}$ which contains sink states. Put $q_0 = s_0$;
- $\rho : Ac_{Ag} \to 2^Q$ is the *action precondition function*, which agrees, for each agent, with $legal(i,a_i) = t(\mathtt{legal\ i\ a_i})$, i.e. $\rho(a_i) = \{q \mid G,q \models_{\text{GDL}} (\mathtt{legal\ i\ a_i})$ and $G,q \models_{\text{GDL}} (\mathtt{not\ terminal}), q \in Q_1\}$. Moreover, $\rho(fin_i) = Q_2 \cup \{q \mid G,q \models_{\text{GDL}} \mathtt{terminal}, q \in Q_1\}$, for every agent i.
- $\tau' : Ac_1 \times \cdots \times Ac_n \times Q \to Q$ is based on τ. We keep all the mappings in τ and add these: $\tau'(\langle fin_1, \ldots, fin_n \rangle, q) = \mathtt{s}_q$, for all $q \in Q_1$ such that $G,q \models_{\text{GDL}} \mathtt{terminal}$;
- $\pi' : Q \to 2^\Phi$ is such that $\pi'(q)$ is the minimal set satisfying the following conditions:
  (1) For all $q, q' \in Q_1$:
    - $init \in \pi'(q_0)$
    - $t(\mathtt{p}) \in \pi'(q) \Leftrightarrow \mathtt{p} \in \pi(q)$
    - for each action profile $\vec{a} = \langle a_1, \ldots, a_n \rangle$ such that $q' = \tau'(\vec{a}, q)$, we require $\forall i \in Ag, done(i,a_i) \in \pi'(q')$, and $t(\mathtt{p})_{old} \in \pi'(q) \Leftrightarrow \mathtt{p} \in \pi(q)$.
  (2) For all $\mathtt{s}_q \in Q_2$, we require $\pi'(\mathtt{s}_q) = \pi'(q) \cup \{\mathtt{s}_\perp, done(fin_i, i)\} \setminus \{done(i,a_i) \mid i \in Ag, a_i \in Ac_i\}$.

Our intuition behind π' is that each state in $Q_1 \setminus \{q_0\}$ has exactly one *done*-proposition for each agent to record the action made in its unique predecessor, and a set of p_{old}-propositions to record the atomic propositions that are true in that same predecessor. Moreover, for $\mathtt{s}_q \in Q_2$, the atom $\mathtt{s}_\perp$ is true, the atoms $done(i,fin_i)$ are true, but for the other atoms, $\mathtt{s}_q$ is exactly as q.

Given a game description Γ, we now have two game models *GMod* and $\mathcal{A}_\Gamma$. To show that they satisfy all the rules of game defined by Γ, we first define a translation from GDL rules to ATL formulae.

DEFINITION 4.2 (Translation from GDL rules to ATL formulas)
Let Γ be a GDL game description. A translation from any GDL rules in $\Gamma_{\mathtt{init}} \cup \Gamma_{\mathtt{glob}} \cup \Gamma_{\mathtt{next}}$ to ATL formulas $\mathcal{R} : \text{GDL} \to \text{ATL}$ is defined as follows:

- $\mathcal{R}(\Leftarrow(\mathtt{init\ p})) = init \to t(\mathtt{p})$
- $\mathcal{R}(\Leftarrow(\mathtt{p})(\mathtt{e}_1)\ldots(\mathtt{e}_m)) = \langle\langle\rangle\rangle \Box(\neg \mathtt{s}_\perp \wedge \mathcal{R}(\mathtt{e}_1) \wedge \cdots \wedge \mathcal{R}(\mathtt{e}_m) \to t(\mathtt{p}))$

- $\mathcal{R}(\Leftarrow(\texttt{next p})(\texttt{e}_1)\ldots(\texttt{e}_m)(\texttt{does i}_1\ \texttt{a}_1)\ldots(\texttt{does i}_k\ \texttt{a}_k)) = \langle\langle\rangle\rangle\Box(\neg\texttt{s}_\bot \wedge \mathcal{R}(\texttt{e}_1)\wedge\cdots\wedge\mathcal{R}(\texttt{e}_m) \rightarrow \langle\langle\{i_1,\ldots,i_k\}\rangle\rangle \bigcirc(t(\texttt{p})\wedge done(i_1,a_1)\wedge\cdots\wedge done(i_k,a_k)))$

where $t : At_{\text{GDL}} \rightarrow At_{\text{ATL}}$ is as in Definition 4.1, and for a GDL expression $\texttt{e}_i$, we stipulate: $\mathcal{R}(\texttt{e}_i) = t(\texttt{p})$ if $\texttt{e}_i = \texttt{p}$ or $\texttt{true p}$, and $\mathcal{R}(\texttt{e}_i) = \neg t(\texttt{p})$ if $\texttt{e}_i = \texttt{not p}$ or $\texttt{not (true p)}$.

THEOREM 4.1
Let Γ be a GDL game description, G_Γ its game model and $\mathcal{A}_\Gamma$ be the associated AATS structure. For each rule $r \in \Gamma_{\texttt{init}} \cup \Gamma_{\texttt{glob}} \cup \Gamma_{\texttt{next}}$, each $s \in S$ and all $\texttt{e} \in AtExpr_{\text{GDL}}$, we have

$$G_\Gamma, s \models_{\text{GDL}} \texttt{e} \text{ iff } \mathcal{A}_\Gamma, s \models_{\text{ATL}} \mathcal{R}(\texttt{e}) \text{ and } G_\Gamma \models_{\text{GDL}} r \text{ iff } \mathcal{A}_\Gamma \models_{\text{ATL}} \mathcal{R}(r)$$

PROOF. The proof is straightforward by induction on GDL rules. ■

4.2 *From GDL descriptions to ATL-theories: $\mathcal{T}_{syn}$*

Now we turn to the syntactic level of the correspondence. Given a GDL description Γ_{GDL}, we translate it into an ATL theory $\mathcal{T}_{syn}(\Gamma_{\text{GDL}}) = \Gamma_{\text{ATL}}$ which characterizes the same game. Here, by 'same game' we mean the following. From the description Γ, we can derive a game model *GMod*, and hence a unique AATS $\mathcal{A}_\Gamma$. And for Γ_{ATL}, there might be several AATSs that satisfy it. They all amount to the same in the sense that there is no ATL formulae that can distinguish these AATSs and $\mathcal{A}_\Gamma$. We will prove this formally later by showing that there is an alternating bisimulation between $\mathcal{A}_\Gamma$ and any AATS model for Γ_{ATL}.

Given Γ, we define the ATL theory Γ_{ATL} as a conjunction of ATL formulae:

$$\Gamma_{\text{ATL}} = DIST \wedge INIT \wedge MEM \wedge ONE\_DONE \wedge LEGAL \wedge STRAT \wedge TERM \wedge SINK$$

Intuitively, these conjuncts play the following roles. *DIST* characterizes the `distinct` predicate. *INIT* characterizes the initial state. Next, *MEM* is to remember the previous state; *ONE_DONE* and *LEGAL* ensure that for each non-terminal state, there is a legal action selected by each agent. Combined with *MEM*, *ONE_DONE* and *LEGAL*, *STRAT* computes the current state given the old state and the actions that have been done. Finally, *TERM* and *SINK* ensure all terminal states will transit to their special sink state.

We now define these properties in detail. First, we need to ensure the intended meaning of the special atom $distinct(\cdot,\cdot)$, as follows:

$$DIST = \langle\langle\rangle\rangle\Box distinct\ (t_1, t_2) \text{ for all terms } t_1 \text{ and } t_2 \text{ that are syntactically different.}$$

We now explain Γ_{ATL} in more detail. Let $S_0 = DatlogPMod(\delta(\Gamma_{\texttt{init}}) \cup \delta(\Gamma_{\texttt{glob}}))$, which gives the set of atomic consequences (using the global rules) of all $(\texttt{init p})$ expressions. We want an ATL formula that characterizes the full initial state. Consider:

$$INIT = init \wedge \langle\langle\rangle\rangle \bigcirc \langle\langle\rangle\rangle \Box \neg init \wedge P_{S_0} \wedge \bigwedge_{p_{old} \in At_{\text{ATL}}} \neg p_{old} \wedge N_{done} \wedge \neg \texttt{s}_\bot$$

where

$$P_{S_0} = \bigwedge_{\texttt{p} \in S_0} p \wedge \bigwedge_{\texttt{p} \notin S_0} \neg p,$$

and

$$N_{done} = \bigwedge_{i \in Ag} \bigwedge_{a \in Ac_i \cup \{fin_i\}} \neg done(i,a).$$

This ensures that the special atom *init* is true in the initial state, and is false everywhere else, and that the truth values of the other atoms in the initial state of *GMod* are reflected properly. It also ensures that all the *old*− and *done*− propositions are false, since there is no previous state, and this is not a `s` state.

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*:

$$MEM = \langle\langle\rangle\rangle \Box \bigwedge_{\mathtt{p} \in At_{\mathrm{GDL}}} ((t(\mathtt{p}) \wedge \neg terminal \rightarrow \langle\langle\rangle\rangle \bigcirc t(\mathtt{p})_{old}) \wedge (\neg t(\mathtt{p}) \wedge \neg terminal \rightarrow \langle\langle\rangle\rangle \bigcirc \neg t(\mathtt{p})_{old}))$$

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

$$ONE\_DONE = \langle\langle\rangle\rangle \Box (\neg init \rightarrow \bigwedge_{i \in Ag} XOR_{a \in Ac_i \cup \{fin_i\}} done(i,a))$$

where *XOR* is the *exclusive OR* operator, a Boolean operator that returns a value of true only if exactly one of its operands is true. One assumption for playing GDL games is that each agent must play legal moves. This is captured by the following:

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

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. For $done(\cdot,\cdot)$ atoms, we have the following equivalence (the left-to-right direction is valid in ATL for arbitrary formulas):

$$\models_{\mathrm{ATL}} \langle\langle i \rangle\rangle \bigcirc done(i,a_i) \leftrightarrow \langle\langle Ag \rangle\rangle \bigcirc done(i,a_i)$$

Let $bd_1, bd_2, \ldots$ be variables over possible bodies of rules, that is, sets of literals, but not including any (`does i a`). Let $\mathtt{p} \in At_{\mathrm{GDL}}$. Now suppose that *all* the rules r in Γ with $hd(r) \in \{(\mathtt{p}), (\mathtt{next\ p})\}$ are the following:

$$\begin{array}{lll}
r_1 : \Leftarrow (\mathtt{p}) & bd_1 & \\
\vdots \quad \Leftarrow (\mathtt{p}) & \vdots & \\
r_h : \Leftarrow (\mathtt{p}) & bd_h & \\
s_1 : \Leftarrow (\mathtt{next\ p}) & bd'_1 & (\mathtt{does\ i}_{1_1}\ \mathtt{a}_{1_1}) \ldots (\mathtt{does\ i}_{m_1}\ \mathtt{a}_{m_1}) \\
\vdots \quad \vdots \quad \vdots & \vdots \quad \vdots \quad \vdots & \\
s_k : \Leftarrow (\mathtt{next\ p}) & bd'_k & (\mathtt{does\ i}_{k_1}\ \mathtt{a}_{k_1}) \ldots (\mathtt{does\ i}_{m_k}\ \mathtt{a}_{m_k})
\end{array}$$

We map all these rules for `p` to an ATL formula $\varphi(\mathtt{p})$. For this, we first translate the symbols from GDL to those of ATL using the functions t and t_{old} defined in Definition 4.1. For convenience, we

denote $t(bd_i)$ as the translation of all the expressions in bd_i by t, and similar for $t_{old}(bd_j)$. For each atom $\mathtt{p} \in At_{\text{GDL}}$, we can now define an ATL constraint $MIN(\mathtt{p})$, as follows:

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

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

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

$$STRAT = \langle\langle\rangle\rangle \Box \bigwedge_{\mathtt{p} \in At_{\text{GDL}}} \left(\neg init \wedge \neg \mathtt{s}_\perp \rightarrow MIN(\mathtt{p}) \right)$$

When a terminal state is reached, no further 'real' moves are played by agents, i.e. they always play the fin_i actions:

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

Reaching a sink state should not change anything:

$$SINK = \langle\langle\rangle\rangle \Box \bigwedge_{p \in At_{\text{ATL}}} (((terminal \vee \mathtt{s}_\perp) \wedge p) \leftrightarrow \langle\langle\rangle\rangle \bigcirc (\mathtt{s}_\perp \wedge p))$$

In Section 4.1, we have shown that we can conceive a GDL game model as an AATS. The following is essentially a soundness result for our transformation. Let Γ be a game description, and *GMod* its game model with initial state s_0; $\mathcal{A}_\Gamma$ is the corresponding AATS. We have:

$$\mathcal{A}_\Gamma, s_0 \models_{\text{ATL}} \Gamma_{\text{ATL}}$$

In the following, we add a requirement resulting in *uniform* AATS structures:

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

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 uniformly true or false. Notice that $\mathcal{A}_\Gamma$ satisfies this requirement.

We now turn to the complexity of reasoning about the uniform AATS structures. The ATL satisfiability checking problem with respect to uniform AATSs is as follows: *given an ATL formula φ, does there exist a uniform AATS $\mathcal{A}$ and a state s in $\mathcal{A}$ such that $\mathcal{A}, s \models_{\text{ATL}} \varphi$?*

LEMMA 4.1
The ATL satisfiability checking problem with respect to uniform AATSs is in EXPTIME.

PROOF. Given an input formula φ, we apply the incremental tableau construction of [16]. This algorithm, which runs in time $O(2^{|\varphi|^2})$, generates a set of 'concurrent game Hintikka structures', which correspond to all the possible models for φ (uniform or otherwise). Once generated, we can check each one to see whether it corresponds to a uniform model: we eliminate all structures that do

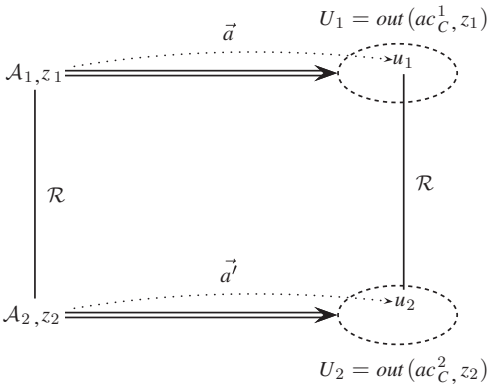

FIGURE 3. Alternating bisimulation between $\mathcal{A}_1$ and $\mathcal{A}_2$.

not correspond to uniform models. If, at the end of this stage, we are left with any structures, then the input formula is satisfiable in a uniform model. Notice that checking whether a concurrent game Hintikka structure corresponds to a uniform model can trivially be done in time polynomial in the size of the structure. ■

Note that the translation of the GDL description Γ_{GDL} into the ATL specification Γ_{ATL} can be done in time polynomial in the size of Γ_{GDL}, where the size of Γ_{GDL} is the number of symbols it contains.

Now we prove an important result: *every model for* Γ_{ATL} *is bisimilar to* $\mathcal{A}_\Gamma$.

THEOREM 4.2
Let G_Γ 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 a uniform AATS that satisfies Γ_{ATL}. Then there exists an alternating bisimulation $\mathcal{R}$ between $\mathcal{A}_1$ and $\mathcal{A}_2$, with $\mathcal{R}q_1q_2$. As a consequence, all AATS models for Γ_{ATL} verify the same formulas.

The proof will be constructive: $\mathcal{R}z_1z_2$ can be chosen to be $\pi_1(z_1) = \pi_2(z_2)$. This means that all that can be said about the game in the language of ATL given a state s, is completely determined by the propositional contents of that current state. If two states agree on propositional content, the sets of options for any agent are the same in those states, and the effect of doing specific combinations of actions is the same as well; hence, all the possible futures are the same. This will be reflected in the proof, which is outlined as follows (see also Figure 3).

First, if in z_1 the atom *terminal* is true, this will also be the case in z_2, and hence the only action available in both states is fin_i, for all agents i. To be more precise, in z_2, we know that all successors verify $done(i, fin_i)$, by *TERM*. (See our remark below, explaining why the proof is a little more involved: we do not actually know that the agents have an action fin_i, in z_2). For z_1, this is so because of the construction of $\mathcal{A}_1$ (it is associated to a game G_Γ), and for z_2, this is because $\mathcal{A}_2$ satisfies *TERM*. So in both models, the system can only proceed to a sink state $\mathtt{s}$ and stay there forever. Since a sink state satisfies the same atoms as it predecessor (for $\mathcal{A}_1$, this is true by construction of π, and

for $\mathcal{A}_2$, this is true due to $\mathit{TERM} \wedge \mathit{SINK}$), and we already assume that those predecessors z_1 and z_2 verify the same atoms, the two sink states are propositionally equivalent, and hence bisimular.

If z_1 and z_2 are not sink states, we can reason about successor states that are reachable by doing some actions. Conceptually, the proof follows the following pattern: if *terminal* is not true, for the **Zig** direction, we have to show that for any coalition C and action $ac^1_C \in \mathit{options}(C, z_1)$, there is a joint action $ac^2_C \in \mathit{options}(C, z_2)$ such that for any $u_2 \in U_2 = \mathit{out}(ac^2_C, z_2)$, there is a $u_1 \in U_1 = \mathit{out}(ac^1_C, z_1)$, such that u_1 and u_2 bisimulate, i.e. have the same propositional content (Figure 3). We like to take for ac^2_C the same as ac^1_C: U_2 and U_1 are then determined. Now take a $u_2 \in U_2$. With what will it bisimulate? Well, u_2 is obtained from z_2 by choosing an additional action for any agent outside C. We choose the same action for those agents in z_1 to find $u_1 \in U_1$. Now, any atom p in z_2 may be false because no action and no global rule made it true. This will be the same for z_1. Alternatively, p was true in z_2 'for a reason'. This reason can be the execution of an action (in which case it will be true in z_1 as well), or it may be true because of some global rule. Here, we apply induction on the stratum in which $\mathtt{p}$ occurs in Γ, to show that p must be true in z_1 as well. Reasoning for the **Zag** case is similar.

However, the details of the proof are more involved, for the reason that some syntactic atoms suggests some semantic properties, which in $\mathcal{A}_1$ must be ensured through the link with Γ and G_Γ, and for $\mathcal{A}_2$ they are guaranteed by Γ_{ATL}. For instance, an atom $\mathit{done}(i, a_i)$ may be true in $\mathcal{A}_1, u_1$, which then means that agent i has an option to perform action a_i in z_1. This would also imply $\mathit{legal}(i, a_i)$ is true in z_1. However, although the induction hypothesis for atoms then guarantees that $\mathit{legal}(i, a_i)$ is also true in z_2, there is *no* direct link with an action a_i being available in z_2 for agent i. Note that although we have actions in AATSs, we cannot directly refer to them in the object language. All the effects and availability of actions in $\mathcal{A}_2$ must be derived from Γ_{ATL}.

Proof. We define a relation $\mathcal{R} \subseteq Q_1 \times Q_2$ as follows,

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

We show that $\mathcal{R}$ is an alternating bisimulation which connects q_1 and q_2.

By *INIT*, one could easily check that $\mathcal{R} q_1 q_2$.

Suppose we have established $\mathcal{R} z_1 z_2$ for some $z_1 \in Q_1$ and $z_2 \in Q_2$ (cf. Figure 3). Easy to see that $\mathcal{R}$ satisfies the invariance condition in Definition 3.4. We need to show that it satisfies both the **Zig** and **Zag** conditions in Definition 3.4 as well.

We first show the **Zig** condition. In the case that $\mathcal{A}_1, z_1 \models_{\mathrm{ATL}} \mathit{terminal}$, by construction of $\mathcal{A}_1$ (a model associated with a game G_Γ), the only actions available to the agents are the fin_i actions, and performing them in z_1 will lead to the unique sink state $\mathtt{s}_{z_1}$. Since $\mathcal{A}_1$ is the model associated with a game G_Γ, by definition π' of such a model we have, for all atoms $p \neq \mathtt{s}_\bot, \mathit{done}(i, a_i)$, that $\mathcal{A}_1, z_1 \models_{\mathrm{ATL}} p$ iff $\mathcal{A}_1, \mathtt{s}_{z_1} \models_{\mathrm{ATL}} p$. We furthermore have $\mathcal{A}_1, \mathtt{s}_{z_1} \models \mathit{done}(i, \mathit{fin}_i) \wedge \mathtt{s}_\bot$. Using the assumption about z_1 and z_2, we also have $\mathcal{A}_2, z_2 \models_{\mathrm{ATL}} \mathit{terminal}$. But since $\mathcal{A}_2$ satisfies *TERM* and *SINK*, we also know that $\mathtt{s}_{z_2}$ satisfies the same atoms $p \neq \mathtt{s}_\bot, \mathit{done}(i, a_i)$ as z_2, and moreover, that $\mathtt{s}_{z_2}$ satisfies $\mathit{done}(i, \mathit{fin}_i) \wedge \mathtt{s}_\bot$. To prove **Zig**, take a coalition $C \subseteq Ag$ and take a $ac^1_C \in \mathit{options}(C, z_1)$: we have just argued that the only ac^1_C is a tuple fin_i ($i \in C$). We have to show that there is a $a^2_C \in \mathit{options}(C, z_2)$, such that for every $u_2 \in \mathit{out}(ac^2_C, z_2)$, there is a $u_1 \in \mathit{out}(ac^1_C, z_1)$ with $\mathcal{R} u_1 u_2$. Take an arbitrary action ac^2_C for C in z_2 and a $u_2 \in \mathit{out}(ac^2_C, z_2)$. Since $\mathcal{A}_2$ satisfies *TERM*, we have $\mathcal{A}_2, u_2 \models_{\mathrm{ATL}} \mathtt{s}_\bot \wedge \mathit{done}(i, \mathit{fin}_i)$ and, by *SINK*, $\mathcal{A}_2, u_2 \models_{\mathrm{ATL}} p$ iff $\mathcal{A}_2, z_2 \models_{\mathrm{ATL}} p$. So z_2 and u_2 agree on all atoms apart from $\mathtt{s}_\bot$ and $\mathit{done}(\mathit{fin}, i)$, which are both true in u_2. We can take $u_1 = \mathtt{s}_{z_1}$, then u_1 and u_2 both verify $\mathtt{s}_\bot \wedge \mathit{done}(i, \mathit{fin}_i)$ and u_1 agrees for the other atoms with z_1, and u_2 agrees for the other atoms with z_2. Since by assumption, z_1 and z_2 verify the same atoms, we conclude that u_1 and u_2 verify the same atoms, and hence $\mathcal{R} u_1 u_2$.

In the following, we suppose $\mathcal{A}_1, z_1 \not\models_{\text{ATL}} \mathit{terminal}$. By assumption, we then also have $\mathcal{A}_2, z_2 \not\models_{\text{ATL}} \mathit{terminal}$. Take an arbitrary coalition C, with a joint action $ac_C^1 \in \mathit{options}(C, z_1)$, and consider $U_1 = \mathit{out}(ac_C^1, z_1) \subseteq Q_1$ in $\mathcal{A}_1$. We need to find $ac_C^2 \in \mathit{options}(C, z_2)$ such that for every $u_2 \in \mathit{out}(ac_C^2, z_2)$, there is a $u_1 \in U_1$ so that $\mathcal{R}u_1u_2$.

It follows from $ac_C^1 \in \mathit{options}(C, z_1)$ that $\mathcal{A}_1, z_1 \models_{\text{ATL}} \mathit{legal}(i, a_i^1)$ for all $i \in C, a_i^1 \in ac_C^1$. Therefore, $\mathcal{A}_2, z_2 \models_{\text{ATL}} \mathit{legal}(i, a_i^1)$ for all $i \in C, a_i^1 \in ac_C^1$. And by *LEGAL*, we have $\mathcal{A}_2, z_2 \models_{\text{ATL}} \langle\langle i \rangle\rangle \bigcirc \mathit{done}(i, a_i^1)$ for all $i \in C$. So, for each $i \in C$, there is $ac_i^2 \in \mathit{options}(i, z_2)$ such that for all $x \in \mathit{out}(ac_i^2, z_2) \subseteq Q_2$, $\mathcal{A}_2, x \models_{\text{ATL}} \mathit{done}(i, a_i^1)$. Let ac_C^2 be an action profile that consists of a_i^2 for all $i \in C$ and $U_2 = \mathit{out}(ac_C^2, z_2) \subseteq Q_2$. It is easy to see that for all $x \in U_2$, we have $\mathcal{A}_2, x \models_{\text{ATL}} \mathit{done}(i, a_i^1)$ 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 *ONE_DONE*, there is one and only one $\mathit{done}(i, a)$ true in u_2 for each $i \in Ag$. We already know $\mathcal{A}_2, u_2 \models_{\text{ATL}} \mathit{done}(i, a_i^1)$ for $i \in C$, and we assume $\mathcal{A}_2, u_2 \models_{\text{ATL}} \mathit{done}(j, b_j^1)$ for all $j \in Ag \setminus C$. As u_2 is a successor of z_2, we have $\mathcal{A}_2, z_2 \models_{\text{ATL}} \langle\langle Ag \rangle\rangle \bigcirc \mathit{done}(j, b_j^1)$ for all $j \in Ag \setminus C$, and by *LEGAL*, we have $\mathcal{A}_2, z_2 \models_{\text{ATL}} \mathit{legal}(j, b_j^1)$ for all $j \in Ag \setminus C$, hence $\mathcal{A}_1, z_1 \models_{\text{ATL}} \mathit{legal}(j, b_j^1)$ for all $j \in Ag \setminus C$. We collect the actions a_i^1 for $i \in C$, and b_j^1 for $j \in Ag \setminus C$ to make a complete action profile $\vec{a}$.

Now go back to $\mathcal{A}_1$ and consider $u_1 = \mathit{out}(\vec{a}, z_1)$. We claim that this u_1 is the state we are looking for: it satisfies $\mathcal{A}_1, u_1 \models_{\text{ATL}} p$ iff $\mathcal{A}_2, u_2 \models_{\text{ATL}} p$, for all $p \in \Phi$. By *MEM*, we have $p_{old} \in \pi_1(u_1)$ iff $p_{old} \in \pi_2(u_2)$ for all $p_{old} \in \Phi$. By *ONE_DONE*, we have $\mathit{done}(i, a_i) \in \pi_1(u_1)$ iff $\mathit{done}(i, a_i) \in \pi_2(u_2)$ for all $\mathit{done}(i, a_i) \in \Phi$.

We now claim:

$$\forall \mathtt{p} \in At_{\text{GDL}}, G, u_1 \models_{\text{GDL}} \mathtt{p} \text{ iff } \mathcal{A}_1, u_1 \models_{\text{ATL}} t(\mathtt{p}) \text{ iff } \mathcal{A}_2, u_2 \models_{\text{ATL}} t(\mathtt{p}) \tag{1}$$

The first 'iff' immediately follows from Theorem 4.1, and we will use it to know 'why' a certain atom is true in G, u_1. Since $u_1 = \mathit{out}(\vec{a}, z_1)$, we know that in G, we have $u_1 = \tau(\vec{a}, z_1)$, i.e. u_1 is calculated from Γ as $\mathit{DatlogPMod}(F_\Gamma(\vec{a}, z_1) \cup \delta(\Gamma_{\mathtt{glob}}))$.

We distinguish two cases:

- Either there is no rule $r \in F_\Gamma(\vec{a}, z_1) \cup \delta(\Gamma_{\mathtt{glob}})$ with $hd(r) = \mathtt{p}$. Then $\mathtt{p} \notin \mathit{DatlogPMod}(F_\Gamma(\vec{a}, z_1) \cup \delta(\Gamma_{\mathtt{glob}}))$ and hence $G, u_1 \not\models_{\text{GDL}} \mathtt{p}$, and, by Theorem 4.1, $\mathcal{A}_1, u_1 \models_{\text{ATL}} \neg p$. Now consider the axiom *STRAT*, which says that $\mathit{MIN}(\mathtt{p})$ is true everywhere in $\mathcal{A}_2$ except the initial state and the sink state. In case that $\mathtt{p}$ does not appear in the head of any rule in Γ, $\mathit{MIN}(\mathtt{p}) = \neg p$, which implies that $\mathcal{A}_2, u_2 \models_{\text{ATL}} \neg p$, as desired. Otherwise, $\mathtt{p}$ must appear in the head of some rule $r \in \Gamma$. Since in this case we assume this is not so for $\delta(\Gamma_{\mathtt{glob}})$, the only way to make p true, using

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

  is that we have some bd'_j, generated by some rule

$$s_j : (\Leftarrow \mathtt{next(p)}\ bd'_j\ \mathtt{does(j_1, a_{j_1})} \cdots \mathtt{does(j_m, a_{j_m})})$$

  for which $\mathcal{A}_2, u_2 \models_{\text{ATL}} t_{old}(bd'_j) \wedge \mathit{done}(j_1, a_{j_1}) \wedge \cdots \wedge \mathit{done}(j_m, a_{j_m})$. By *ONE_DONE*, we know that for any $i \in Ag$, the only action b_i for which $\mathcal{A}_2, u_2 \models_{\text{ATL}} \mathit{done}(i, b_i)$ is true is $b_i = a_i$. By *MEM*, since $\mathcal{A}_2, u_2 \models_{\text{ATL}} t_{old}(bd'_j)$, we have that $\mathcal{A}_2, z_2 \models_{\text{ATL}} bd'_j$. Using the induction hypothesis, we

get $\mathcal{A}_1, z_1 \models_{\text{ATL}} bd'_j$. Now looking at $\mathcal{A}_1$ as a game model G for Γ, we see that $(\Leftarrow \mathtt{p}) \in F_\Gamma(\vec{a}, z_1)$, contradicting our assumption that there is no rule in $F_\Gamma(\vec{a}, z_1) \cup \delta(\Gamma_{\mathtt{glob}})$ with $\mathtt{p}$ as a head.

- Or, for some rule $r \in F_\Gamma(\vec{a}, z_1) \cup \delta(\Gamma_{\mathtt{glob}})$, we have $hd(r) = \mathtt{p}$. We distinguish two sub-cases:
  - $r \in F_\Gamma(\vec{a}, z_1)$. It follows that $r = (\Leftarrow \mathtt{p})$. And since $u_1 = DatlogPMod(F_\Gamma(\vec{a}, z_1) \cup \delta(\Gamma_{\mathtt{glob}}))$, we have $G, u_1 \models_{\text{GDL}} \mathtt{p}$. It also follows from $r \in F_\Gamma(\vec{a}, z_1)$, that $G, z_1 \cup \{\mathtt{does\ 1\ a_1} \wedge \cdots \wedge \mathtt{does\ n\ a_n}\} \models_{\text{GDL}} bdl(r') \wedge bda(r')$ for some $\mathtt{next}$ rule r' in the form of $(\Leftarrow \mathtt{next(p)}\ bdl\ bda)$, where bdl is the literal part of this rule, and bda is the action part. This means that $a_x \in \vec{a}$ for all $\mathtt{does\ x\ a_x} \in bda$. By construction of G, we have $G, z_1 \models_{\text{GDL}} bdl(r')$, and, by Theorem 4.1, we have $\mathcal{A}_1, z_1 \models_{\text{ATL}} t(bdl(r'))$ which gives, by the induction hypothesis, $\mathcal{A}_1, z_2 \models_{\text{ATL}} t(bdl(r'))$ and, by *MEM*, $\mathcal{A}_2, u_2 \models_{\text{ATL}} t_{old}(bdl(r'))$. By choice of u_2, we also have $\mathcal{A}_2, u_2 \models_{\text{ATL}} done(1, a_1) \wedge \cdots \wedge done(n, a_n)$, thus $\mathcal{A}_2, u_2 \models_{\text{ATL}} t_{old}(bda(r'))$. By $MIN(p)$, we then have $\mathcal{A}_2, u_2 \models_{\text{ATL}} p$.
  - $r \in \delta(\Gamma_{\mathtt{glob}})$ and $r \notin F_\Gamma(\vec{a}, s)$. Now we consider a level mapping $f : \mathtt{e}(At_{\text{GDL}}(\Gamma)) \to \mathbb{N}$. We claim

$$\forall n \in \mathbb{N} : f(\mathtt{x}) = n \Rightarrow (G, u_1 \models_{\text{GDL}} \mathtt{x} \Leftrightarrow \mathcal{A}_1, u_1 \models_{\text{ATL}} x \Leftrightarrow \mathcal{A}_2, u_2 \models_{\text{ATL}} x)$$

    We do induction on $f(\mathtt{p})$.
    - *Base case:* $f(\mathtt{p}) = 0$. There must be a global rule $(\Leftarrow \mathtt{p})$ in $\delta(\Gamma_{\mathtt{glob}})$, thus $G, u_1 \models_{\text{GDL}} \mathtt{p}$ and $\mathcal{A}_1, u_1 \models_{\text{ATL}} p$. And by $MIN(\mathtt{p})$, we have $\mathcal{A}_2, u_2 \models_{\text{ATL}} p$. This proves the claim.
    - *Induction step:* suppose $f(\mathtt{p}) = k+1$, and the claim proven for all $\mathtt{q}$ with $f(\mathtt{q}) \leq k$. We have to show that $G, u_1 \models_{\text{ATL}} \mathtt{p} \Leftrightarrow \mathcal{A}_1, u_1 \models_{\text{ATL}} p \Leftrightarrow \mathcal{A}_2, u_2 \models_{\text{ATL}} p$. The fact $G, u_1 \models_{\text{GDL}} \mathtt{p}$ is true if and only if there exists a rule $r = (\Leftarrow \mathtt{p}\ bd) \in \delta(\Gamma_{\mathtt{glob}})$ such that $G, u_1 \models_{\text{GDL}} bd$. For any atom $\mathtt{q} \in bd$, $f(q) < k+1$, so by induction hypothesis, we know that $G, u_1 \models_{\text{ATL}} \mathtt{q} \Leftrightarrow \mathcal{A}_1, u_1 \models_{\text{ATL}} q \Leftrightarrow \mathcal{A}_2, u_2 \models_{\text{ATL}} q$, for all $\mathtt{q} \in bd$. It follows that $G, u_1 \models_{\text{ATL}} bd \Leftrightarrow \mathcal{A}_1, u_1 \models_{\text{ATL}} t(bd) \Leftrightarrow \mathcal{A}_2, u_2 \models_{\text{ATL}} t(bd)$. And by $MIN(\mathtt{p})$, we have $G, u_1 \models_{\text{GDL}} \mathtt{p}$ if and only if $\mathcal{A}_2, u_2 \models_{\text{ATL}} p$.

We now show the **Zag** condition.

Take an arbitrary coalition C and with a joint action ac^2_C and consider $U_2 = out(ac^2_C, z_2) \subseteq Q_2$ in $\mathcal{A}_2$. Pick an arbitrary $u_2 \in U_2$, we apply *ONE_DONE*, so for $i \in C$, we have a unique $done(i, a^1_i)$ true in u_2. Due to the *uniform requirement*, we have that for all $u \in U_2$ and all $i \in C$, $\mathcal{A}_2, u \models_{\text{ATL}} done(i, a^1_i)$. Take a^1_i into ac^1_C, we have an action profile for C. And let $U_1 = out(ac^1_C, z_1)$.

We want to demonstrate that for every $u_1 \in U_1$ there is a $u_2 \in U_2$ for which $\mathcal{R} u_1 u_2$. Choose $u_1 \in U_1$ arbitrarily. Let $\vec{a}$ be the action profile for which $u_1 = out(\vec{a}, z_1)$, it is easy to see that in G this means that $u_1 = \tau(\vec{a}, z_1)$, i.e. $u_1 = DatlogPMod(F_\Gamma(\vec{a}, z_1) \cup \delta(\Gamma_{\mathtt{glob}}))$. Now we have $\mathcal{A}_1, z_1 \models_{\text{ATL}} legal(j, a^1_j)$ for $j \in Ag \setminus C$ and $a^1_j \in \vec{a}$. And by assumption, $\mathcal{A}_2, z_2 \models_{\text{ATL}} legal(j, a^1_j)$ as well. Hence, by *LEGAL*, we have $\mathcal{A}_2, z_2 \models_{\text{ATL}} \langle\langle j \rangle\rangle \bigcirc done(j, a^1_j)$ for every $j \in Ag \setminus C$. For each $done(j, a^1_j)$, we can find an action a^2_j in $\mathcal{A}_2$ such that for all $u \in out(a^2_j, z_2)$, we have $\mathcal{A}_2, u \models_{\text{ATL}} done(j, a^1_j)$. We collect a^2_j for all $j \in Ag \setminus C$, and combine them with a^2_i for all $i \in Ag$, then we get a complete action profile $\vec{a'}$.

Now let $u_2 = out(\vec{a'}, z_2)$, and we claim that this is the one to complete $\mathcal{R} u_1 u_2$. The proof that $\forall \mathtt{p} \in At_{\text{GDL}}, G, z_1 \models_{\text{GDL}} \mathtt{p}$ iff $\mathcal{A}_1, u_1 \models_{\text{ATL}} p$ iff $\mathcal{A}_2, u_2 \models_{\text{ATL}} p$ is similar to the proof of (1) above. ■

COROLLARY 4.1
Given a game description Γ, the following two are equivalent, for any formula φ:

(1) The specific model $\mathcal{A}_\Gamma$ satisfies φ, i.e. $\mathcal{A}_\Gamma \models_{\text{ATL}} \varphi$, or, equivalently, $\mathcal{A}_\Gamma, q_0 \models_{\text{ATL}} \varphi$;
(2) φ follows from the theory Γ_{ATL}, i.e. $\Gamma_{\text{ATL}} \models_{\text{ATL}} \varphi$.

One way of interpreting the results of Theorem 4.2 and Corollary 4.1 above is as follows: GDL can be viewed as a *model specification language*, suitable for use in a *model checker* [6]. This gives rise to the formal decision problem of *ATL model checking problem over GDL game descriptions*, which can be described as follows: *given a GDL game description* Γ *and an ATL formula* φ *(containing only atoms and agents that occur in* Γ*), is it the case that* $\mathcal{A}_\Gamma \models_{\text{ATL}} \varphi$*?*

THEOREM 4.3
ATL model checking over propositional GDL game descriptions is EXPTIME-complete.

PROOF. We first show that the problem can be decided in time exponential in $|\Gamma|+|\varphi|$, where $|\Gamma|$ is the number of symbols in Γ and $|\varphi|$ is the number of symbols in φ. This follows from Theorem 4.2 and Lemma 4.1. Given a game description Γ, and ATL formula φ, construct Γ_{ATL}. The question whether $\mathcal{A}_\Gamma \models_{\text{ATL}} \varphi$ is equivalent to the question whether $\Gamma_{\text{ATL}} \wedge \neg\varphi$ is unsatisfiable in ATL; the correctness of this procedure follows from Corollary 4.1. The fact that ATL unsatisfiability is in EXPTIME is from [9, 29].

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 [25, p. 158]. An instance of PEEK-G_4 is a quadruple:

$$\langle X_1, X_2, X_3, \varphi \rangle$$

where:

- X_1 and X_2 are disjoint, finite sets of Boolean variables, with the intended interpretation that the variables in X_1 are under the control of agent 1, and X_2 are under the control of agent 2;
- $X_3 \subseteq (X_1 \cup X_2)$ are the variables deemed to be true in the initial state of the game; and
- φ is a propositional logic formula over the variables $X_1 \cup X_2$, representing the winning condition.

The game is played in a series of rounds, with the agents $i \in \{1,2\}$ alternating (with agent 1 moving first) to select a value (true or false) for one of their variables in X_i, with the game starting from the initial assignment of truth values defined by X_3. Variables that were not changed retain the same truth value in the subsequent round. An agent wins in a given round if it makes a move such that the resulting truth assignment defined by that round makes the winning formula φ true. The decision problem associated with PEEK-G_4 involves determining whether agent 2 has a winning strategy in a given game instance $\langle X_1, X_2, X_3, \varphi \rangle$. Notice that PEEK-$G_4$ only requires 'memoryless' (Markovian) strategies: whether or not an agent i can win depends only on the current truth assignment, the distribution of variables, the winning formula and whose turn it is currently. As a corollary, if agent i can force a win, then it can force a win in $O(2^{|X_1 \cup X_2|})$ moves. Given an instance $\langle X_1, X_2, X_3, \varphi \rangle$ of PEEK-G_4, we encode PEEK-G_4 in GDL as follows (cf. [26]). Let X_1, X_2 and X_3 be as described above. We give some GDL code for PEEK in Figure 4. Initialization (lines starting with an 0) is straightforward, where, in line 02, z is a variable from X_3, and we assume that player 1 starts the game (line 03). The lines starting with a 1 determine what the legal actions are, and define the turn taking. Action *noop* is always legal (lines 10, 11), and players can change their 'own' variables as long as they are not in a terminal state: in lines 12 and 14, we require $x_1 \in X_1$. Lines 16–19 specify that control alternates between the two players.

The lines starting with 2 describe how to compute the new value for a variable x. Ideally, we would like to say: 'in the next state, x is true if either it was true and not set to false, or else if it was set to true'. However, since we are not allowed to have a negated `(does player1 make_x_false)`, we do the following. For any variable $x \in X_1 \cup X_2$, we add three variables x_0, x_1 and x_{old}. The idea here is that x_0 signals that x has just been set to false (lines 20, 21), and x_1 means that x has just been

```
00 (<= role player1)
01 (<= role player2)
02 (<= init (z))
    ...
03 (<= control player1)
    ...
10 (<= (legal player1 noop))
11 (<= (legal player2 noop))
12 (<= (legal player1 make_x1_false)
13     (not terminal))
14 (<= (legal player1 make_x1_true)
15     (not terminal))
    ...
16 (<= (next (control player1))
17     (control player2))
18 (<= (next (control player2))
19     (control player1))
   ...
20 (<= (next x0)
21     (does player1 make_x_false))
22 (<= (next x1)
23     (does player1 make_x_true))
    ...
```

```
24 (<= (true (x))
25     (true (x_old))(not x0))
26 (<= (true (x))
27     (true (x1)))
28 (<= (next (x_old))
29     (true x))
    ...
30 (<= (terminal)
31     (true (phi_1)) ... (true (phi_n)))
    ...
32 (<= (true (phi_i))
33     (true (p_i_j)))
    ...
34 (<= (true (phi_i))
35     (not (q_i_j)))
    ...
40 (<= (goal player1 100)
41     (terminal)(control player2))
42 (<= (goal player2 100)
43     (terminal)(control player1))
44 (<= (goal player1 0)
45      (not (goal player1 100)))
46 (<= (goal player2 0)
47      (not (goal player2 100)))
```

FIGURE 4. Encoding PEEK-G_4 in GDL: the reduction for Theorem 4.3.

put true (lines 22 and 23). Now, x can be true in a next state because of two reasons: it was true in the previous state, and has not been made false (lines 24, 25), or because it has been made true (lines 26, 27). Recall that in all other cases, x will become false. Lines 28 and 29 specify that x_{old} is true exactly when x was true in the previous state. Note that when a player plays *noop*, neither x_0 nor x_1 will be set to true, and hence (lines 24–29) x will become true iff it was true.

Lines starting with 3 describe when the goal φ is reached. We assume the goal φ is in conjunctive normal form: $\varphi = \varphi_1 \wedge \cdots \wedge \varphi_n$. Let φ_i be $p_{i_1} \vee \cdots \vee p_{i_k} \vee \neg q_{i_1} \vee \cdots \vee \neg q_{i_m}$, where each p_{i_j} and q_{i_v} ($j \leq k, v \leq m$) is an atom. A clause φ_i will become true if any of its disjuncts becomes true (lines 32–35). Then if all clauses φ_i becomes true, we reach a terminal state (lines 30–31).

Finally, lines starting with 4 describe the payoffs. A player wins when we reach a terminal state (a state in which φ is true) and he has just moved, i.e. the other player has control (lines 40–43). In all other cases, the player is losing (lines 44–47).

Moving to the ATL language, the formula $\langle\langle i \rangle\rangle \Diamond win_i$ (where $win_i = goal\ player\ i\ 100$) formalizes that i has a winning strategy. In ATL, one might think this is equivalent to $\langle\langle i \rangle\rangle \Diamond \varphi$, but it is not. Let φ be $(p_1 \wedge p_2 \wedge p_2) \vee (q_1 \wedge q_2)$, with $X_1 = \{p_1, p_2, p_3\}$ and $X_2 = \{q_1, q_2\}$ and $X_3 = \emptyset$ (no atom is true, initially). In this game, we have $\langle\langle 1 \rangle\rangle \Diamond \varphi$ (player 1 just makes p_1, p_2 and p_3 true, in that order), but not $\langle\langle 1 \rangle\rangle \Diamond win_1$ (player 2 can 'get there faster'). Note that $\langle\langle 1,2 \rangle\rangle \Diamond \varphi$ is equivalent to $\langle\langle 1,2 \rangle\rangle \Diamond (win_1 \vee win_2)$ which is true initially iff φ is satisfiable. ■

Readers familiar with ATL model checking may wonder why the complexity of model checking ATL formulae against GDL game descriptions is exponentially harder than model checking ATL against semantic structures such as Concurrent Game Structures [3]. The reason is that GDL provides for the *succinct* specification of very large game models—game models that are exponentially large in the size of the GDL specification. This issue is discussed in more detail in [26]. Note that, although this seems a negative result, it means that interpreting ATL over propositional GDL descriptions is no more complex than interpreting ATL over apparently simpler model specification languages such as the Simple Reactive Systems Language [26].

5 Characterizing playability conditions with ATL

We claimed in Section 1 that ATL is an appropriate language for expressing playability conditions for GDL-specified games, and in this section, we put some flesh on this claim. First, some words about what we aim to accomplish. This is perhaps best explained by analogy with the literature on temporal logic for reactive systems [10]. Temporal logics, in various forms, have been used for reasoning about reactive systems for several decades, and a large literature has been established on classifying the properties of such systems via temporal formulae of various types; probably the best known classification is that of liveness and safety properties, although many more properties have been classified [18, 19, p. 298]. Our ultimate aim is, in much the same way, to use ATL to derive a similar classification of game properties. Note that ATL is, of course, a temporal logic, and we might expect the classification to include liveness and safety properties and similar; but the more novel aspect of the classification (and, crucially, the part of the classification which simply cannot be done in conventional temporal logic) is a classification of *strategic* properties of games. Perhaps the most important question is whether a game is 'balanced', in that all players have in some sense an equal chance to win. As it turns out, this apparently intuitive property is surprisingly hard to define, but we will see various notions of balancedness in what follows.

We begin our top-level classification of game properties by distinguishing between properties relating to the *coherence* of a game and those relating to its *strategic* structure. We assume to have a stock of state atoms $SAt = \{p, q, \ldots\}$ (in Tic-Tac-Toe, an example would be `(cell 1 2 x)`), old atoms $OAt = \{p_{old} \mid p \in SAt\}$ and *done atoms* $DAt = \{done(i,a) \mid i \in Ag, a \in Ac_i\}$. (Throughout this section, unless stated otherwise, properties that we discuss will be evaluated in the beginning of the game.)

5.1 Coherence properties

Roughly, coherence properties simply ensure that the game has a 'sensible' interpretation. To illustrate what we mean by this, we introduce a vocabulary of atomic propositions that we use within game property formulae. These propositions play an analogous role to propositions such as $at_i(\ell)$ in the temporal axiomatization of programs [13, p. 70].

- $turn_i$ will be true in a state if it is agent i's turn to take a move in that state;
- $legal(i,a)$ will be true in a state if action (move) a is legal for agent i in that state;
- $has\_legal\_move_i$ will be true in a state s if agent i has at least one legal move in that state;
- *terminal* will be true in a state if that state is terminal, i.e. the game is over.
- win_i will be true in a state if agent i has won in that state;
- $lose_i$ will be true in a state if agent i has lost in that state;
- *draw* will be true in a state if the game is drawn in that state;

Note that the specific *interpretation* of these atomic propositions will depend on the game at hand, but they will typically be straightforward to derive. In the context of GDL, we might have $win_i = goal(i,100)$, $lose_i = goal(i,0)$ and $draw = \bigwedge_{i \in Ag} goal(i,50)$.

On top of this, we assume that every agent has access to an action *noop*. The idea of such an action is that, when an agent performs this action, nothing will change *as an effect of that action*, but of course, in synchronous games, things may change because of another agent's actions. We also assume to have the *fin* action for every agent i that we introduced for the AATS $\mathcal{A}_\Gamma$. This action is slightly different from *noop*: we do not consider the action *fin* to be a legal one. We only added it for technical reasons, ensuring that in the $\mathcal{A}_\Gamma$, there is always a successor state. In a detailed analysis, one might like to have several actions modelling parts of the purpose of our *noop*: for instance, it is well

possible that all players (in a card game, for instance) want to '*pass*', and the rules of the game might specify that when such a situation occurs a given number of times, the game ends. Summarizing: the action *noop* is a legal action which signifies that an agent leaves making changes to the others, but when *all* agents decide to perform *noop*, we assume that the game terminates (we do not have a *pass* action), and the only action that agents can perform from then on are *fin* actions.

Now that we have such a vocabulary in place, we can start to define specific properties.

From the perspective of designing a game, the general game playing competition [12] suggests the following criteria to be a necessity: it should first of all be *playable*: every player has at least one move in every non-terminal state. We represent this constraint as follows.

$$\langle\langle\rangle\rangle\Box(\neg terminal \rightarrow \bigwedge_{i\in Ag} has\_legal\_move_i) \qquad \text{(Playability)}$$

where $has\_legal\_move_i = \bigvee_{a\in Ac_i, a\neq fin_i} legal(i,a)$.

In a turn-based game, all but one player will only be able to submit a *noop* action, and this should be considered legal: it is the only action allowed, for those agents. In other words, even when it is not a player's turn, he should be able to submit a legal move. Hence, what non-Playability checks is, whether that are non-terminal states in which an agent has no action to perform. Note that in our sense, when all players decide to play the *noop*, this is considered playable.

We can define turn-based games by the following.

$$\langle\langle\rangle\rangle\Box(turn_i \leftrightarrow \neg legal(i, noop)) \qquad \text{(Turn)}$$

In a finite extensive game, the terminal states are exactly those in which no player can perform a move. This signals a fundamental difference with ATL, where computations are by definition infinite. We can bridge this gap by letting a terminal state in a game correspond with a 'sink-state', from which transitions are possible, but only to (copies of) itself. So, our first property says that a terminal state really is terminal: once we reach a terminal state, nothing subsequently changes. For all properties only involving state atoms, we have:

$$\langle\langle\rangle\rangle\Box((terminal \wedge \varphi) \rightarrow \langle\langle\rangle\rangle\Box(terminal \wedge \varphi)) \qquad \text{(GameOver)}$$

The above property is in fact shorthand for infinitely many properties: one for every instantiation of φ. In other words, we not only claim that specific atoms stay true in a terminal state, but arbitrary properties do. Expressing this involves a scheme φ, and as such it would lend itself more naturally to the theorem proving paradigm, rather than that of model checking. That is, when doing theorem proving, one can use schemes and give a deduction for a property like GameOver. However, in our set-up we can deal with this as follows.

Let p be a state atom in SAt. We assume that state atoms cannot be changed by the players' *noop* or *fin* actions. So true state atoms remain true if all agents perform either *noop* or *fin*. This is captured by the following property:

$$\langle\langle\rangle\rangle\Box(p \rightarrow \langle\langle\rangle\rangle\Box((\bigwedge_{i\in Ag} (done(i, noop) \vee done(i, fin))) \rightarrow p)) \qquad \text{(No Change)}$$

Now, we can establish (GameOver) by imposing the following, from which (GameOver) would then follow by induction over φ:

$$\langle\langle\rangle\rangle\Box(terminal \rightarrow \langle\langle\rangle\rangle \bigcirc \langle\langle\rangle\rangle\Box(terminal \wedge \bigwedge_{i \in Ag} done(i, fin))) \quad \text{(Ind)}$$

Next, we often have that a state is terminal if the game is either won or drawn.

$$\langle\langle\rangle\rangle\Box((draw \vee \bigvee_{i \in Ag} win_i) \rightarrow terminal)$$

Note that we may or may not have the converse implication, as we can specify more subtle results using $goal(i, x)$.

There will typically be some coherence relation between win_i, $lose_i$ and *draw* propositions, although the exact relationship will depend on the game. For example, the following says that a draw excludes a win.

$$\langle\langle\rangle\rangle\Box(draw \rightarrow \bigwedge_{i \in Ag} \neg win_i) \quad \text{(Draw)}$$

Finally, one might add conditions like *termination*, which says that a game will eventually end:

$$\langle\langle\rangle\rangle\Diamond terminal \quad \text{(Termination)}$$

5.2 *Fair playability conditions*

All of the above conditions contain only coalition modalities with empty set of agents, i.e. of the form $\langle\langle\rangle\rangle T\varphi$. Recall that $\langle\langle C\rangle\rangle T\varphi$ means that the agents in C can chose a strategy such that no matter what the agents in $Ag \setminus C$ do, $T\varphi$ will hold. In particular, $\langle\langle\rangle\rangle T\varphi$ then means that no matter what the agents in Ag do, $T\varphi$ will hold. Thus, these conditions define invariants, i.e. safety properties, over games. Such properties could thus be specified using conventional temporal logics, such as CTL, and verified using conventional temporal logic model checkers. We now turn to a fundamentally different class of properties—those relating to the strategic structure of a game. As we argued above, such strategic properties cannot be specified using conventional temporal logics, whence our interest in logics such as ATL for this purpose.

In general, the kinds of properties we might typically hope for in a game relate to 'fairness'[3]—intuitively, the idea that no player has an inherent advantage in the game. In fact, it turns out to be rather hard to give a useful formal meaning to the term, let alone to capture such a meaning logically. Nevertheless, there are some useful fairness-related playability conditions that we can capture.

We first define the notion of *winnability*. A game is *strongly winnable* iff:

> for some player, there is a sequence of individual moves of that player that leads to a terminal state of the game where that player's goal value is maximal. [12, p. 9].

[3]The term 'fairness' is already used in a technical sense in the temporal logic/verification community, to mean something related but slightly different. Here, when we talk about fairness, we are appealing to the everyday meaning of the term, rather than the technical meaning as in [11].

Formally, Strong Winnability may be captured as follows.

$$\bigvee_{i \in Ag} \langle\langle i \rangle\rangle \Diamond win_i \qquad \text{(Strong Winnability)}$$

The Strong Winnability is too strong for games involving multiple players, as if it would hold in the initial state, then perfect play by that player would guarantee a win by that player, which makes the game inherently unfair. So, we have a more relaxed requirement, called *Weak Winnability*, for multi-player games:

> A game is weakly winnable if and only if, for every player, there is a sequence of joint moves of all players that leads to a terminal state where that player's goal is maximal.' [12, p. 9].

We capture this as follows:

$$\bigwedge_{i \in Ag} \langle\langle Ag \rangle\rangle \Diamond win_i. \qquad \text{(Weak Winnability)}$$

In general game playing, every game must be weakly winnable, and all single player games are strongly winnable. This means that in any general game, *every player at least has a chance of winning*.

One might also impose 'Weak Losability', which would be like (Weak Winnability), but with win_i replaced by $lose_i$: at least, in principle, every player could lose.

There are many other notions of fairness one can impose on a game. We say a game is *fair* if no player can lose without himself at least being involved. To put it another way, a player can only lose with less than perfect play. Such a condition is false in games where there is a player with a winning strategy in the initial state.

$$\bigwedge_{i \in Ag} \neg \langle\langle Ag \setminus \{i\} \rangle\rangle \Diamond lose_i \qquad \text{(Fair)}$$

5.3 *Characterizing different games*

The notions we just discussed can be considered as examples of minimal requirements to call a system a 'meaningful game'. We show how ATL can be used to characterize different kinds of games. In fact, we have already seen such a general property: our (Strong Winnability) is in the literature known as *determinacy* of the game. Other examples would include (Sequential): everywhere, the next state is determined by one agent. In ATL, such a situation is called *turn-based* [3]. Although the characteristic formula refers to arbitrary φ again, it can also be related to $\langle\langle\rangle\rangle \Box(XOR_{i \in Ag} turn_i)$, together with (Turn) and (Ind).

$$\langle\langle\rangle\rangle \Box(\langle\langle Ag \rangle\rangle \bigcirc \varphi \rightarrow \bigvee_{i \in Ag} \langle\langle i \rangle\rangle \bigcirc \varphi) \qquad \text{(Sequential)}$$

In many sequential games, the *order* in which players take their turns is crucial. Although 'young children are obsessed with making sure that they go first in any and every game that they play' [8, p. 56], sometimes, rather than a first-mover advantage, there is a second-mover advantage. Consider, for example, the well-known problem of dividing a piece of cake, where one player cuts the cake

and the other player chooses a piece: here, there is no advantage to being the cutter. We define the advantage of agent i as this: the payoff of agent i is strictly larger than payoffs to the other agents. Here is a formal expression:

$$adv_i = \bigvee_{x\in\{0\ldots100\}} (goal(i,x) \wedge \bigwedge_{j\in Ag\setminus\{i\},x>y\geq 0} \neg goal(j,y))$$

Second-mover advantage might be defined as follows:

$$\bigwedge_{i\in Ag} ((\neg turn_i \wedge \langle\langle\rangle\rangle \bigcirc turn_i) \rightarrow \langle\langle i\rangle\rangle \Diamond adv_i) \qquad \text{(Second-mover advantage)}$$

Other examples include (Zero-sum), which we here give for a two-player game:

$$\langle\langle\rangle\rangle \Box(terminal \rightarrow ((win_1 \wedge lose_2) XOR (draw_1 \wedge draw_2) XOR (lose_1 \wedge win_2)) \qquad \text{(Zero-sum)}$$

Note that although we currently have modelled the outcome propositions as booleans, one can do this easily as numbers as well, enabling the easy representation of constant-sum games.

Finally, we have the following formula that characterizes one-shot strategic form games with symmetric payoffs:

$$Strategic \wedge \bigwedge_{x,y\in\{0\ldots100\}} Symmetry(x,y) \qquad \text{(Strategic Symmetry)}$$

where

$$\bigwedge_{i\in Ag} turn_i \wedge \langle\langle\rangle\rangle \bigcirc terminal \qquad \text{(Strategic)}$$

$$(\langle\langle Ag\rangle\rangle \bigcirc (goal(1,x) \wedge goal(2,y)) \rightarrow \langle\langle Ag\rangle\rangle \bigcirc (goal(1,y) \wedge goal(2,x)) \qquad \text{(Symmetry}(x,y)\text{)}$$

Note that, since we assume that all Ac_i and Ac_j are disjunct when $i \neq j$, in (Symmetry(x,y)) agents do not need to be able to 'swap actions', they only need swap outcomes.

5.4 *Special properties for Tic-Tac-Toe*

We now consider properties specific to our running example, Tic-Tac-Toe. For this game, we denote the players by Xplayer and Oplayer, respectively.

Now, our game designer may want to verify that the property that certain configurations on the board will never be reached [e.g. (iCell) expresses the invariant that we do not have two o's and one x in the game in the first row and only blanks everywhere else—recall that the player who starts is Xplayer]. Such properties need not be about invariants, but can also be, for instance, about the progress in the game, or about persistence of written cells or non-written ones (Persistence(x)) (saying that a written X is persistent over any move, a non-written X is persistent under any move of the other player(s)). Using our atoms that recall what is true in the previous state, we can also specify the exact effect of any action: (Write(x)) says that wherever we are in the game, saying that it is x's

turn is equivalent to saying that in every next state, there is exactly one cell that is written with an x now, but was blank before.

$$\langle\langle\rangle\rangle\Box\neg(c(1,1,o)\wedge c(1,2,o)\wedge c(1,3,x)\wedge\bigwedge_{i\neq 1}c(i,j,b)) \tag{iCell}$$

$$\langle\langle\rangle\rangle\Box(\bigwedge_{1\leq i,j\leq 3}(c(i,j,x)\rightarrow\langle\langle\rangle\rangle\bigcirc c(i,j,x))\wedge(\neg turn_{Xplayer}\rightarrow \bigwedge_{1\leq i,j\leq 3}(\neg c(i,j,x)\rightarrow\langle\langle\rangle\rangle\bigcirc\neg c(i,j,x)))) \tag{Persistence(x)}$$

$$\langle\langle\rangle\rangle\Box(turn_{Xplayer}\leftrightarrow\langle\langle\rangle\rangle\bigcirc XOR_{1\leq i,j\leq 3}(c(i,j,x)\wedge c(i,j,b)_{old})) \tag{Write(x)}$$

Regarding game playing, of course, it is interesting to know what parties can achieve, in a given game. The designer of player i might, in particular, be interested whether the following instantiation of (Strong Winnability) holds: is it the case that $\langle\langle i\rangle\rangle\Diamond win_i$? In Tic-Tac-Toe, no player can guarantee a win, i.e. (Strong Winnability) is not true for Tic-Tac-Toe. Indeed, for most interesting games, (Strong Winnability) does not hold.

Let $happy(C)=\bigwedge_{i\in C}(win_i\vee draw)$. For instance (Coalition) expresses that coalition C has some reason to cooperate: by doing so, everybody is reasonably happy, while there is no subset of C that guarantees that.

$$\langle\langle C\rangle\rangle\Diamond happy(C)\wedge\neg\bigvee_{C'\subset C}\langle\langle C'\rangle\rangle\Diamond happy(C') \tag{Coalition}$$

As another example, (R(i,a)) considers whether a is a reasonable move for i: i.e. it cannot achieve less than what it currently can achieve, by performing a. This is an example of a property one might want to check in several states of the game, not just the root.

$$happy(i)\wedge turn_i\wedge\langle\langle i\rangle\rangle\bigcirc(done(i,a)\wedge happy(i)) \tag{R(i,a)}$$

In [24], we describe our work on how to verify GDL-specified games using a pre-existing ATL model checker. The main purpose of this work is to show a method by which existing ATL model checking tools can be used to verify GDL games, rather than developing a model checking tool from the scratch. We have implemented a translator, GDL2RML, from GDL descriptions to representations in the Reactive Modules Language (RML). RML is the model description language of the ATL model checker MOCHA, which was developed by Alur *et al.* [1, 4]. Using GDL2RML, we can verify properties expressed in ATL via MOCHA. For details, we refer to [24].

6 Conclusions

There has been much interest recently in the connections between logic and games, and in particular in the use of ATL-like logics for reasoning about game-like multi-agent systems. In this article, we have investigated connections between ATL and GDL. In particular, we have made two main contributions. First, we have demonstrated that GDL can be understood as a specification language for ATL models, and proved that the problem of interpreting ATL formulae over GDL descriptions is EXPTIME-complete. Second, we have characterized a class of playability conditions which should hold in different games, and they can be used to express the correctness of the games specified in GDL.

In [24], we presented an automated tool that transforms a GDL description into an RML specification, so that we can verify the playability properties on the RML description using an off-the-shelf ATL model checker, MOCHA. In future research, we will apply our work to formal verification of further GDL descriptions: the GDL game designer can express desirable properties of games using ATL, and then automatically check whether these properties hold their GDL descriptions. The main issues are likely to be the efficiency and scalability of our automated tools.

Acknowledgement

We would like to thank Dirk Walther, Nicolas Troquard and Michael Thielscher for helpful comments and discussions. We also thank two anonymous reviewers and the participants from the Workshop on Logic Rationality and Interactions (Beijing, August 2007) for their helpful comments and discussions, and two anonymous reviewers from the Journal of Logic and Computation for their helpful comments.

References

[1] R. Alur, L. de Alfaro, T. A. Henzinger, S. C. Krishnan, F. Y. C. Mang, S. Qadeer, S. K. Rajamani, and S. Taşiran. MOCHA user manual. *University of Berkeley Report*, 2000.

[2] R. Alur, T. Henzinger, O. Kupferman, and M. Vardi. Alternating refinement relations. In *International Conference on Concurrency Theory*, Vol. 1466 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 163–178, 1998.

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

[4] R. Alur, T. A. Henzinger, F. Y. C. Mang, S. Qadeer, S. K. Rajamani, and S. Taşiran. Mocha: modularity in model checking. In *CAV 1998: Tenth International Conference on Computer-aided Verification, (LNCS Volume 1427)*, pp. 521–525. Springer, 1998.

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

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

[7] W. F. Clocksin and C. S. Mellish. *Programming in Prolog*. Springer, 1981.

[8] A. Dixit and S. Skeath. *Games of Strategy*, 2nd edn. W. W. Norton & Co., Inc., 2004.

[9] G. van Drimmelen. Satisfiability in alternating-time temporal logic. In *Eighteenth Annual IEEE Symposium on Logic in Computer Science (LICS 2003)*, Ottawa, Canada, pp. 208–217, 2003.

[10] E. A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science Volume B: Formal Models and Semantics*, J. van Leeuwen, ed. pp. 996–1072. Elsevier Science Publishers B.V., 1990.

[11] N. Francez. *Fairness*. Springer, 1986.

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

[13] R. Goldblatt. *Logics of Time and Computation (CSLI Lecture Notes Number 7)*. Center for the Study of Language and Information, Ventura Hall, 1987. (Distributed by Chicago University Press).

[14] V. Goranko. Coalition games and alternating temporal logics. In *Proceeding of the Eighth Conference on Theoretical Aspects of Rationality and Knowledge (TARK VIII)*, J. van Benthem, ed., Siena, Italy, pp. 259–272, 2001.

[15] V. Goranko and W. Jamroga. Comparing semantics of logics for multi-agent systems. *Synthese*, **139**, 241–280, 2004.

[16] V. Goranko and D. Shkatov. Tableau-based decision procedures for logics of strategic ability in multi-agent systems. *ACM Transactions on Computational Logic*, To appear, 2009.
[17] N. Love, T. Hinrichs, and M. Genesereth. General game playing: game description language specification. *Technical report*, Stanford University, Stanford, 2006.
[18] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer, 1992.
[19] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems — Safety*. Springer, 1995.
[20] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, eds, *Machine Intelligence 4*, pp. 463–502. Edinburgh University Press, 1969.
[21] M. Pauly. *Logic for Social Software*. PhD Thesis, ILLC Dissertation Series 2001–10. University of Amsterdam, 2001.
[22] M. Pauly and M. Wooldridge. Logic for mechanism design — a manifesto. In *Proceedings of the 2003 Workshop on Game Theory and Decision Theory in Agent Systems (GTDT-2003)*, Melbourne, 2003.
[23] B. Pell. *Strategy Generation and Evaluation for Meta-Game Playing*. PhD Thesis, Trinity College, University of Cambridge, 1993.
[24] J. Ruan, W. van der Hoek, and M. Wooldridge. Model checking GDL through MOCHA: a case study. *Technical Report ULCS-09-14*, Department of Computer Science, University of Liverpool, 2009.
[25] L. J. Stockmeyer and A. K. Chandra. Provably difficult combinatorial games. *SIAM Journal of Computing*, **8**, 151–174, 1979.
[26] W. van der Hoek, A. R. Lomuscio, and M. Wooldridge. On the complexity of practical ATL model checking. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS 2006)*, pp. 201–208. ACM, 2006.
[27] W. van der Hoek, M. Roberts, and M. Wooldridge. Social laws in alternating time: effectiveness, feasibility, and synthesis. *Synthese*, **156**, 1–19, 2007.
[28] W. van der Hoek and M. Wooldridge. Time, knowledge, and cooperation: alternating-time temporal epistemic logic and its applications. *Studia Logica*, **75**, 125–157, 2003.
[29] D. Walther, C. Lutz, F. Wolter, and M. Wooldridge. ATL satisfiability is indeed ExpTime-complete. *Journal of Logic and Computation*, **16**, 765–787, 2006.

Received 8 August 2008