On the Complexity of General Game Playing

Édouard Bonnet and Abdallah Saffidine

LAMSADE
Université Paris-Dauphine
edouard.bonnet@dauphine.fr

School of Computer Science and Engineering
The University of New South Wales
abdallahs@cse.unsw.edu.au

Abstract. The Game Description Language (GDL) used in General Game Playing (GGP) competitions provides a compact way to express multi-agents systems. Multiple features of GDL contribute to making it a convenient tool to describe multi-agent systems. We study the computational complexity of reasoning in GGP using various combinations of these features. Our analysis offers a complexity landscape for GGP with fragments ranging from NP to EXPSPACE in the single-agent case, and from PSPACE to 2EXPTIME in the multi-agent case.

1 Introduction

General Game Playing (GGP) research aims at developing systems capable of reasoning on a variety of multi-agent situations encoded in the Game Description Language (GDL) [4]. GDL can be seen as a domain specific logic programming language that provides a compact representation for transition systems. As such, GDL can be related both the Planning Domain Description Language (PDDL) and to Datalog.

The similarity between GDL and PDDL stems from the fact that both languages were created to describe dynamic situations. PDDL revolves around the agent's actions and allows modeling the preconditions and effects of actions chosen by a planning system. On the other hand, GDL focusses on predicates that characterize a current state and possible actions and their effects can be inferred via logical reasoning on the state.

While GDL is based on Datalog and incorporates many aspects of it such as negation-as-failure, it features multiple elements absent in Datalog such as nested function constants and more importantly a multi-agent system semantics as opposed to a (set of) model(s).

Many language features of GDL make it a convenient tool to describe games and other multi-agent systems. However, this very expressivity makes reasoning about described systems challenging.

In this paper, we tackle the computational complexity of reasoning about systems described in GDL. We study the impact, or lack thereof, of the most notable language features on the problems of finding a winning strategy for a given agent, and on the reachability problem.

Some standard results on the complexity of logic programming can be lifted to GDL [3]. While these results apply to the static part of GDL: state queries such as computing the legal moves for each agent, they do not tell much about the dynamical aspects of the game. Indeed, few complexity results for GGP have been established before. In particular, the multi-agent propositional fragment was known to be EXPTIME-complete [10]. In this paper, we provide the first comprehensive analysis of the complexity of GDL and its dependency on the language features used. Our results are summarized in Table 1.

Table 1. Complexity of the reachability problem with rational agents. This can also be seen as the complexity of determining whether a given player has a winning strategy. The multi-agent results hold with as few as one rational agent in a stochastic environment, i.e., with Nature as a player, as well as with two rational agents with conflicting goals. Markov chain features a single agent which is Nature.

Fragment	Single-agent	Markov chain	Multi-agent
Prop. and Monot.	NP-c	PP-c	PSPACE-c
Propositional	PSPACE-c	PSPACE-c	EXPTIME-c
Monotonic	NEXPTIME-c	PEXPTIME-c	EXPSPACE-c
Bounded	EXPSPACE-c	EXPSPACE-c	2EXPTIME-c
Full	UNDEC	UNDEC	UNDEC

Several researchers have investigated the complexity of planning in other description languages such as STRIPS [1] or PDDL [9].

After defining the syntax, semantics, and fragments of GDL, we introduce the complexity theoretic framework in which we carry our analysis. We then proceed to the core of the paper, establishing upper complexity bounds for the fragments of GDL and a reduction from the word problem for Turing Machines (TMs) with space or time restrictions to GDL providing matching lower bounds.

2 The Game Description Language

The Game Description Language (GDL) has been developed to formalise the rules of any finite game with complete information in such a way that the description can be automatically processed by a general game player.

GDL game descriptions are sets of *normal logic program clauses* [3] written in prefix notation using s-expressions, where variables are indicated by a leading `?`. A language especially designed for game descriptions, GDL uses a few pre-defined predicate symbols shown in Table 2.

In GDL it is assumed that gameplay happens synchronously, that is, all players move simultaneously and the world changes only in response to moves.

GDL imposes some syntactic restrictions on a set of clauses with the intention to ensure uniqueness and finiteness of the set of derivable predicate instances [6]. Specifically, the program must be stratified and satisfy the *recursion restriction*. Stratified logic programs are known to admit a unique stable model [3].

Definition 1. *A GDL program satisfies the* recursion restriction *if the following holds for every rule R. If the body of R contains a predicate q depending the head of R, p, then at least one of the following must hold for every argument v_j of q in R. Either v_j is ground, or v_j appears as an argument of p in R, or v_j appears as an argument of another predicate r of R such that r does not depend on p.*

Table 2. Predefined GDL predicates and their interpretation.

Predicate instance	Meaning
`(role` r`)`	r is a player
`(init` f`)`	f holds in the initial position
`(true` f`)`	f holds in the current position
`(legal` r m`)`	player r has legal move m
`(does` r m`)`	player r does move m
`(next` f`)`	f holds in the next position
`terminal`	the current position is terminal
`(goal` r n`)`	player r gets goal value n

The semantics of a set of game rules has been informally described by a state transition system [4] and later formalised as follows [12]. Let $G \models A$ denote that atom A is true in the unique answer set of a stratified set of rules G. The *players* in game G are $R = \{r : G \models (\texttt{role}\ r)\}$. The *initial state* is $\{f : G \models (\texttt{init}\ f)\}$. A move m is *legal in state* S if $G \cup S^{\texttt{true}} \models (\texttt{legal}\ r\ m)$. Here, $S^{\texttt{true}}$ is the collection of facts $\{(\texttt{true}\ f_1), \ldots, (\texttt{true}\ f_k)\}$ that compose the current state $S = \{f_1, \ldots, f_k\}$. A *joint move* assigns each role $r \in R$ a legal move. The *state transition* from state S by joint move M results in the state $\{f : G \cup S^{\texttt{true}} \cup M^{\texttt{does}} \models (\texttt{next}\ f)\}$. Here, $S^{\texttt{true}}$ is as above and $M^{\texttt{does}}$ denotes the collection of facts $\{(\texttt{does}\ r_1\ m_1), \ldots, (\texttt{does}\ r_n\ m_n)\}$ such that joint move M assigns m_i to player r_i. Finally, a *terminal state* is any S such that $G \cup S^{\texttt{true}} \models \texttt{terminal}$; and a *goal value* for player $r \in R$ in state S is any v for which $G \cup S^{\texttt{true}} \models (\texttt{goal}\ r\ v)$ holds.

We adopt the convention that `random` is a special role constrained to select its action uniformly at random among its legal moves [13]. This is necessary to model games involing chance events such as throwing a dice or tossing a coin.

The recursion restriction ensures that only a finite number of predicate instances are derivable from a fixed set of clauses. However, the set of clauses $G \cup S^{\texttt{true}}$ in the semantics of GDL is a dynamic. As a result, the recursion restriction is not enough to guarantee a bounded number of predicate instances over the course of a game. We therefore propose the following stronger restriction on GDL programs.

Definition 2. *Let Δ be a GDL program. Let Δ' be the program Δ extended with the set of rules* $\{(\leftarrow$(**true** ?f) (**next** ?f)), $(\leftarrow$(**true** ?f) (**init** ?f)), $(\leftarrow$(**does** ?r ?m) (**legal** ?r ?m))$\}$. *We say that Δ satisfies the* General Recursion Restriction (GRR) *exactly when Δ' satisfies the recursion restriction.*

The GRR makes the dependency between the `true`, `init`, and `next` as well as the `does` and `legal` predicates explicit. Intuitively, the recursion restriction bounds the size of terms for a fixed set of clauses, and the GRR bounds the size of terms across a sequence of sets of clauses related by the GDL dynamics.

Consider the game of *Tictactoe*. We can distinguish two sets of features: the `control` fluent determines which player is going to mark a cell, and the `cell` fluent determines which player, if any, has marked a given location of the board. `control` instances alternate as the game is played, but `cell` instances are more static. If a predicate instance of the form (**true** (cell m n x)) appears in a state, then it must appear in all subsequent states. Conversely, if a predicate instance of the form (**true** (cell m n b)) does not appear in a state, then it cannot appear in any subsequent states. Thus, the instances of the fluent `cell` are monotonic while the instances of `control` are not.

Definition 3. *A fluent f is* persistent *if for every state s where f holds, f holds in every successor of s. f is* anti-persistent *if for every state s where f does not hold, f does not hold in every successor of f. A fluent is* monotonic *if it is either persistent or anti-persistent.* [1]

Variables, non-monotonic fluent instances, and sets of clauses not satisfying the GRR are source of computational complexity in GDL reasoning. To study their influence formally, we define the following fragments of the Game Description Language.

Definition 4. *A game description is in* bounded domain *if it satisfies the GRR. A game description is* propositional *if it satisfies the GRR and has a bounded number of variables. A description is* monotonic *if the number of non-monotonic fluents is bounded by a constant.*[2]

Table 3 summarizes the dependency between the game features and the GDL fragment considered.

Table 3. Game features.

Class	Query	# fluents	# different	Longest
		# $\neq$ moves	states	acyclic path
Prop. and Monot.	$\in$ P	poly	expo	poly
Propositional	$\in$ P	poly	expo	expo
Monotonic	EXPTIME-c	expo	2-exp	expo
Bounded	EXPTIME-c	expo	2-exp	2-exp
Full	EXPTIME-c	unbounded	unbounded	unbounded

Game features such as the maximum number of fluents holding in any one state, or the number of legal moves depend mainly on whether variables are allowed. Variables also impact static state queries: the complexity jumps from P-complete to EXPTIME-complete when variables are allowed in the representation [3]. Conversely, as long as the GRR is satisfied, the nesting depth of function symbols remains polynomial. The

[1] We can actually have a more general definition where we just bound for each fact the number of alternations between being true and false.

[2] As usual in complexity theory, we implicitly consider sets of descriptions. For example, a set of descriptions corresponds to generalized chess, and each board size maps to one GDL description. When the number of variables is bounded, increasing the size of the chess instance can polynomially increase the size of the GDL code but not the number of variables.

game description can therefore be rewritten so that nested symbols are only allowed to depth 2 (to accommodate the fixed arity of the predefined keywords).

The number of facts is at most linear if there are no variables, and singly-exponential if there are variables. As the number of facts n_f and the number of possible states n_s are linked by the relation $n_s = 2^{n_f}$, the number of different states is singly-exponential or doubly-exponential depending on whether there are variables. The number of legal moves, in a given state, is at most linear if there are no variables, and singly-exponential if there are variables.

The following proposition justifies our definition of *propositional*.

Proposition 1. *One can transform a* GDL *encoding with a constant number of variables into a polynomial size* GDL *encoding without variables (in polynomial time).*

Proof. For each rule featuring a variable, write down all the possible instantiated rules. This has constant blow-up d^c where d is the size of the domain and c is the constant number of variables.

3 Turing Machines and Complexity Classes

The reader can find the following standard definitions in [8].

Definition 5. *A* Turing Machine (TM) *is a tuple $\langle Q, q_1, \Delta, g\rangle$ where Q is a finite set of* states*; $q_1 \in Q$ is a distinguished* initial *state; $\Delta \subseteq Q \times \{0,1\} \times Q \times \{0,1\} \times \{\leftarrow, \rightarrow\}$ is a set of* transition rules*; and $g : Q \rightarrow \{\exists, \forall, ?, \top, \bot\}$ is a* labeling *of the states. The labels denote respectively existential, universal, stochastic, (final) accepting, and (final) rejecting states.*

Definition 6. *A* configuration *is a triple $\langle w_1, q, w_2\rangle$ where q is the current state, w_1w_2 is the content of the tape, and the head is upon the first letter of w_2.*

Definition 7. *The probability p that a configuration $\langle w_1, q, w_2\rangle$ is* accepting *can be defined inductively as follows. If q is accepting then we set $p = 1$ and if q is rejecting then $p = 0$. Else let n be the number of possible transitions and for each transition i, let p_i be the probability that the resulting configuation is accepting. If q is existential then we set $p = \max_i p_i$, if q is universal then $p = \min p_i$, and if q is stochastic then $p = \frac{\sum_i p_i}{n}$.*

A word w is recognized if the probability that configuration $\langle \varepsilon, q_1, w\rangle$ is accepting is greater than $1/2$.

Equivalently, the acceptance condition can be seen as a game between Existential player who chooses the transition applied from existential states and Universal player who does so from universal ones.

Definition 8. *A* TM *is* deterministic *if in each non-final configuration there is exactly one applicable transition. A* TM *is* non-deterministic *if all non-final states are existential,* alternating *if they are existential or universal,* probabilistic *if they are stochastic, and* stochastic-alternating *if they are existential or stochastic.*

DTIME$(f(n))$ is the class of deterministic machines working in time $O(f(n))$. DSPACE$(f(n))$ is the class of deterministic machines working in space $O(f(n))$. NTIME$(f(n))$ and NSPACE$(f(n))$ are their non-deterministic counterpart and ATIME$(f(n))$ and ASPACE$(f(n))$ are their alternating counterpart.

Theorem 1 (Savitch [11]). NPSPACE = PSPACE *and* NEXPSPACE = EXPSPACE.

Theorem 2 (Chandra *et al.* [2]). APTIME = PSPACE*,* APSPACE = EXPTIME*,* AEXPTIME = EXPSPACE*, and* AEXPSPACE = 2EXPTIME*.*

SATIME$(f(n))$ is the class of stochastic-alternating machines working in time $O(f(n))$. SAATIME$(f(n))$ is the class of machines working in time $O(f(n))$ with existential, universal and stochastic states. PSPACE$(f(n))$, SASPACE$(f(n))$, and SAASPACE$(f(n))$ are defined similarly for space.

Proposition 2. *For any function $f \in \Omega(n)$ growing no slower than linearly,* SAATIME$(f(n)) =$ SATIME$(f(n)) =$ ATIME$(f(n))$*, and* SAASPACE$(f(n)) =$ SASPACE$(f(n)) =$ ASPACE$(f(n))$*.*

Proof. We recall an idea first used to show that NP $\subseteq$ PP [5], and to show that APTIME $\subseteq$ SAPTIME [7]. This idea, more generally, allows to show that ATIME$(f(n)) \subseteq$ SATIME$(f(n))$ and that ASPACE$(f(n)) \subseteq$ SASPACE$(f(n))$.

To simulate an alternating machine with a stochastic-alternating machine, one can start on a stochastic state. In the first branch all the runs are rejecting but one, and in the second branch the alternating machine is mimicked by switching the universal states to stochastic states. To win the Existential player needs to win all his games "against nature" in the second branch, which is equivalent to defeating Universal player.

Now, we show that SAATIME$(f(n)) \subseteq$ NSPACE$(f(n)) =$ ATIME$(f(n))$. Let $\mathcal{A}$ be a general machine (existential, universal and stochastic states) working in time $O(f(n))$.

Without loss of generality, we can assume that in every non-final configuration there are exactly two applicable transitions; that all the runs have the same length $3f(n)$; and that, besides the transition towards final state, that there are only three possible kinds of transition: from existential state to universal state, from universal state to stochastic state, and from stochastic state to existential state [7].

We traverse the computing tree of $\mathcal{A}$ as follows. We maintain on the tape of our NSPACE$(f(n))$ machine $\mathcal{A}'$ two counters c_W and c_L up to $2^{f(n)}$ and a pointer which represents our position in the computing tree. We maintain one additional counter c which indicates how many stochastic state have been encountered along the branch. This can be done using space $O(f(n))$. Basically, c_W will count the number of accepting runs, and c_L the number of rejecting runs. If the state is existential, we guess which transition to apply. If the state is universal, we check all the transitions one after the other. If the state is stochastic, we increment c, and we check the left transition, then the right one. When we reach a leaf, we add $2^{f(n)-c}$ to c_W if the final state is accepting, or we add $2^{f(n)-c}$ to c_L if the final state is rejecting. When all the computation tree has been explored, we accept if c_W contains a bigger number than c_L.

SAASPACE$(f(n)) \subseteq$ DTIME$(2^{f(n)}) =$ ASPACE$(f(n))$. Indeed, the configuration graph of the general machine has $O(2^{f(n)})$ vertices, and we can decide the value of the stochastic reachability game in polynomial time in the number of vertices, *i.e.*, $O(2^{f(n)})$.

4 Upper bounds

A GDL state is defined by a set of (true) facts. An *extensive* representation of a state is an exhaustive list of grounded facts. An *implicit* representation of a state is a list of terms, such that all the possible instantiations of the terms should exactly match the set of true facts. In the former case, we say that the state is *implicitly represented*, and in the latter, *extensively represented*. An extensive representation can be exponentially (in the arity of the function constant) larger than an implicit representation. For instance, if the set of constants is $D = \{0, 1\}$, $\{g\ ?x_1\ ?x_2, h\ 1\ ?y\}$ is the implicit representation and $\{g\ 0\ 0, g\ 0\ 1, g\ 1\ 0, g\ 1\ 1, h\ 1\ 0, h\ 1\ 1\}$ is the extensive representation of the same state. In the multi-agent case, the problem is to decide whether the first agent can win even if the other agents cooperate against him. Thus, we can merge all the opponents into one unique opponent, and we can consider that there are only two agents. In the following propositions, we show the corresponding result for the single-agent case, and Theorem 2 on alternation transfers the result from the single-agent to the multi-agent case.

Proposition 3. *Single-agent propositional monotonic GDL is in* NP. *Multi-agent propositional monotonic GDL is in* PSPACE.

Proof. The length of a shortest win is polynomial. Indeed, from a winning sequence of joint moves, you can remove all those joint moves which have no effect on the list of facts. It is still a winning sequence. Now, the list of facts is different from one state to the next. As it is propositional, the total number n_f of facts is linear in the GDL encoding. As it is monotonic, one can characterise fully the winning sequence by a set of n_f intervals representing the period of validity of each fact. At each state, one interval starts or ends, so the winning sequence is of size at most $2n_f$.

A strategy for the single agent consists of finding the one move to apply, at each move. The number of (propositional) moves is linear in the description of the game. Thus, you can guess a polynomial word encoding all the moves to apply from the beginning to the end of the game. Check if that word corresponds to a winning strategy can be done in polynomial time. At each step, you maintain the list of facts by applying a linear number of "next" rule, and you check if the moves are legal by considering a linear number of "legal" rules. If you reach the desired *goal value* at the end then you accept.

Proposition 4. *Single-agent propositional GDL is in* PSPACE. *Multi-agent propositional GDL is in* EXPTIME.

Proof. A state can be extensively represented in polynomial space. The number of legal moves in a given state is linearly bounded. Non-deterministic polynomial space allows to guess which move to play and maintain the current list of facts. Thus, it belongs to NPSPACE, which is equal to PSPACE by Savitch's theorem (Theorem 1).

Proposition 5. *Single-agent monotonic GDL is in* NEXPTIME. *Multi-agent monotonic GDL is in* EXPSPACE.

Proof. The number of facts is singly-exponential, but the depth of a game is also only singly-exponential 2^{n^c} since it is monotonic. The number of legal moves in a given state is exponentially bounded by $2^{n^{c'}}$. Deriving in GDL the next state from the current state, and that a state is terminal, is of singly-exponential depth bounded by, say, $2^{n^{c''}}$. Thus, a certificate for a winning strategy of the single-agent is of size at most $n^{c'}2^{n^c}2^{n^{c''}}$. Hence, it belongs to NEXPTIME.

Proposition 6. *Single-agent bounded GDL is in* EXPSPACE. *Multi-agent bounded GDL is in* 2EXPTIME.

Proof. A state can be represented extensively in exponential space. The number of legal moves in a state is exponentially bounded. Non-deterministic exponential space allows to guess which move to play and maintain the current list of facts. Thus, it belongs to NEXPSPACE, which is equal to EXPSPACE by Savitch's theorem (Theorem 1).

5 Lower bounds

In this section, we obtain the lower bounds for the complexity results described in Table 1. We describe how we can encode TMs with various restrictions in different fragments GDL in a three-step reduction.

The first part is a set of generic axioms of constant size. The second part is a set of machine specific axioms encoding the transition rules and the labeling of the states. The third part encodes the restriction set on the running time or the tape space used by the machine. There, we provide a different encoding for each specific restriction.

The choice of the third part alone determines to which fragment of GDL the program belongs. Indeed, the number of variables appearing in the first and second part is constant and the fluents introduced in the first part can be made monotonic in the third part.

Generic Axioms In our description, variables starting with `?p` are GDL variables ranging over tape *position* indices. The variables starting with `?t`, `?q`, `?a`, `?d`, and `?r` range respectively over the *time* domain, the states of the machine, the letters of the *alphabet*, the *direction* of transitions, and the players (or *roles*).

We use the following auxiliary predicates. `accept` and `reject` take a state index as argument and characterize final states. Similarly, `label` characterizes non-final state labels and indicates their type. `delta` takes 5 arguments a, i, b, j, d such that a and b denote alphabet symbols, i and j represent states, and d represents a direction. This rigid predicate records the transition rules of the machine: there are as many instances of `delta` as there are elements in Δ.

The `zerop` and `succp` predicates encode the relation between the possible positions of the machine cursor on the tape. `(zerop` p`)` holds for the unique leftmost position p of the cursor on the tape. `(succp` p_1 p_2`)` holds exactly when p_1 represents a cursor position immediately to the right of p_2. `zerot`, `succt` encode the relation between the different times represented. `input` takes a possible cursor position i and a tape symbol a and denotes that the ith letter of the input word is a. `now` characterizes the current time.

Listing 1. GDL simulation of a TM: generic termination and acceptance.

```
1  (← (role ?r) (label ?r ?q))
2  (← terminal (true (state ?t ?q)) (accept ?q))
3  (← terminal (true (state ?t ?q)) (reject ?q))
4
5  (← (goal exists 100) (role exists) (true (state ?t ?q)) (accept ?q))
6  (← (goal exists   0) (role exists) (true (state ?t ?q)) (reject ?q))
7  (← (goal univer   0) (role univer) (true (state ?t ?q)) (accept ?q))
8  (← (goal univer 100) (role univer) (true (state ?t ?q)) (reject ?q))
```

Listing 2. GDL simulation of a TM: generic initial configuration.

```
9  (← (init (head ?t ?p)) (zerot ?t) (zerop ?p))
10 (← (init (state ?t 1)) (zerot ?t))
11 (← (init (tape ?t ?p ?a)) (zerot ?t) (input ?p ?a))
12 (← (init (tape ?t ?p2 0)) (zerot ?t) (less ?p1 ?p2) (endinput ?p1))
13
14 (← (less ?p1 ?p2) (succp ?p1 ?p2))
15 (← (less ?p1 ?p3) (succp ?p1 ?p2) (less ?p2 ?p3))
```

We also use three main sets of fluents. The `tape` fluent encodes the content of the machine tape. `state` encodes the current state of the machine. `head` encodes the position of the cursor on the tape.

We have one agent per non-final state label and the game ends when the machine reaches a final configuration (line 1–3, Listing 1).

The goal of an existential player, if such a player exists for the game, is to bring the game into an accepting configuration (lines 5 to 6). Conversely, the goal of a universal player, if such a player exists for the game, is to bring the game into a rejection configuration (lines 7 to 8). When we simulate TMs without existential (resp. universal) states, the predicate `role` does not hold for `exist` (resp. `univer`) and the utility of the existential (resp. universal) player does not need to be defined. A player `random` may also be introduced if the machine has stochastic states but we do not need to specify goal values for that player. Note that the `random` role is a distinguished player in GDL which is assumed to select a move uniformly at random among its legal moves.

At the beginning, the head is on the first cell of the tape, the machine is in the initial state, and the tape contains the input word and then blank symbols (line 9 to 12 in Listing 2). We have introduced the `less` predicate such that (`less` p_1 p_2) holds when p_2 if situated further right that p_1. It can be based on the more elementary successor predicate `succp`.

The semantics of GDL assume simultaneous actions by all agents. However in our reduction from TMs, only the agent corresponding to the label of the current state makes a meaningful decision at a time. To comply with the semantics of the language, we use two kind of actions: an `apply` action taking a letter, a state, and a direction as arguments and recording the transition effects, and a `pass` action. The mapping from

Listing 3. GDL simulation of a TM: generic applicable transitions.

```
16 (← (legal ?r (apply ?a2 ?q2 ?d)) (delta ?a1 ?q1 ?a2 ?q2 ?d)
17    (true (head ?t ?p)) (true (tape ?t ?p ?a1))
18    (label ?q1 ?r) (now ?t) (true (state ?t ?q1)))
19 (← (legal ?r2 pass) (role ?r2) (distinct ?r1 ?r2)
20    (label ?q ?r1) (now ?t) (true (state ?t ?q)))
```

Listing 4. GDL simulation of a TM: generic evolution of the configuration.

```
21 (← (next (tape ?t2 ?p  ?a)) (now ?t1) (succt ?t1 ?t2)
22    (does ?r (apply ?a ?q ?d)) (true (head ?t1 ?p)))
23 (← (next (tape ?t2 ?p2 ?a)) (now ?t1) (succt ?t1 ?t2) (distinct ?p1 ?p2)
24    (true (tape ?t1 ?p2 ?a)) (true (head ?t1 ?p1)))
25
26 (← (next (state ?t2 ?q)) (now ?t1) (succt ?t1 ?t2)
27    (does ?r (apply ?a ?q ?d)))
28
29 (← (next (head ?t2 ?p2)) (now ?t1) (succt ?t1 ?t2) (succp ?p2 ?p1)
30    (true (head ?t1 ?p1)) (does ?r (apply ?a ?q  left)))
31 (← (next (head ?t2 ?p2)) (now ?t1) (succt ?t1 ?t2) (succp ?p1 ?p2)
32    (true (head ?t1 ?p1)) (does ?r (apply ?a ?q right)))
```

possible transitions into legal agent moves is given in Listing 3. For instance, if the current state has label $\exists$ then the player `exists` chooses among the instances of `apply` to select a transition for the TM, and the other agents, if any, perform a `pass` action.

The evolution of the configuration of the machine as the transitions are selected is described in Listing 4. The cell under the head changes according to the transition effects, but the rest of the tape remains unaffected (line 21 to 24). The next state is determined by the transition effects recorded in the `apply` action. After a transition, the head moves one cell to the left or one cell to the right depending on the kind of transition performed (line 29 to 32).

Machine-dependent Axioms Assuming numbered states, $Q = \{q_1, \ldots, q_{|Q|}\}$, Listing 5 collects the machine-dependent GDL axioms: state labels and transitions.

The number of variables appearing in the fragments described so far is bounded by a constant that does not depend on the size of the input. Indeed, variables only appear in the generic part of the translation that does not depend on the specific machine to be simulated. The size of Listing 5 naturally depends on the specific machine but it only contains ground terms and it does not introduce any new fluent.

Time and Tape Axioms Let us detail how we can ensure that a fragment satisfies the monotonicity assumption. Listing 6 provides axioms to be included when monotonicity is needed. We first add *inertia rules* that guarantee the persistence of the `state`, `head`, and `tape` fluents (Line 1 to 3 in Listing 6).

Listing 5. GDL simulation of a TM: machine specific axioms.

```
For all state $q_i \in Q$, add
(accept $i$)  when $g(q_i) = \top$
(reject $i$)  when $g(q_i) = \bot$
(label exists $i$)  when $g(q_i) = \exists$
(label univer $i$)  when $g(q_i) = \forall$
(label **random** $i$)  when $g(q_i) = ?$

For all transition rules $(a, q_i) \to (b, q_j, d) \in \Delta$, add
(delta $a$ $i$ $b$ $j$  left)  when $d = \leftarrow$
(delta $a$ $i$ $b$ $j$ right)  when $d = \rightarrow$
```

Listing 6. Ensuring monotonicity: inertia axioms and linear time axioms.

```
1 (← (next (state ?t ?q))    (true (state ?t ?q)))
2 (← (next (head ?t ?p))     (true (head ?t ?p)))
3 (← (next (tape ?t ?p ?a)) (true (tape ?t ?p ?a)))
4
5 (← (next (past ?t)) (true (state ?t ?q)))
6 (← (now ?t) (true (state ?t ?q)) (not (true (past ?t))))
7
8 (← (zerot ?x) (zerop ?x))
9 (← (succt ?x ?y) (succp ?x ?y))
```

This importance of the time argument for these fluents now becomes clearer. We can have monotonicity without the head of the machine always pointing at the same cell at every stage of the game. The monotonic fact that is remembered throughout the rest of the game is that at some fixed time t, the head pointed at a given cell.

Since past configurations are remembered when monotonicity is enforced, we need to distinguish which is the current one. Recall that fluents need to be monotonic but not arbitrary predicates. We can therefore define a non-monotonic `now` predicate. To do so, we introduce a persistent `past` fluent such that `(true (past ?t))` only holds for past time points `?t` (Line 5 to 6 in Listing 6). `past` is persistent since it only depends on `state` which is persistent.

Finally, we give time a linear structure mapped from the linear structure of the tape. Thus, the simulation length inherits any bound on the size of the tape (Line 8 to 9 in Listing 6). When monotonicity is not required, history is not kept in the state and time points need not be distinguished. In that case, simpler axioms are used (Listing 7).

Listing 7. Dummy time axioms.

```
(now dummy) (zerot dummy) (succt dummy dummy)
```

Listing 8. Tape of size n^c and input w of size n: linear encoding.

```
(zerop 0)
(succp h h')  for all h ∈ {0,...,n^c − 1} and h' = h + 1
(input i w_i)  for all i ∈ {0,...,n − 1}
(endinput n)
```

Listing 9. Tape of size 2^{n^c} and input w of size n: binary encoding.

```
(bit 0)  (bit 1) (zerop (bin 0 ... 0))
(← (succp (bin ?b_{n^c} ... ?b_{h+1} 0 1 ... 1) (bin ?b_{n^c} ... ?b_{h+1} 1 0 ... 0))
    (bit ?b_{h+1}) ... (bit ?b_{n^c})) for all h ∈ {1,...,n^c}

For i ∈ {0,...,n − 1} with the binary writing $\overline{b_{\log n} \dots b_1}$, add
(input (bin 0 ... 0 b_{log n} ... b_1) w_i)
If the binary writing of n − 1 is $\overline{b_{\log n} \dots b_1}$, then add
(endinput (bin 0 ... 0 b_{log n} ... b_1))
```

We now provide the axioms defining the tape structure, zerop and succp, as well as the axioms defining the input word, input and endinput. If w is an input word of size n, then for each $i \in \{0, \dots, n-1\}$, w_i denotes the $i+1$-th letter of w. Listing 8 gives a linear encoding such that a polynomial number of consecutive tape positions can be represented. This encoding uses a polynomial number of axioms and does not use any variable. Listing 9 gives a binary encoding such that an exponential number of consecutive tape positions can be represented. The additional bit predicate provides the domain of bit variables, namely 0 and 1. This encoding uses a polynomial number of variables and a polynomial number of axioms.

Combining the Listings We have now described all the elements needed for the reduction of a TM to a GDL program.

Theorem 3. *Let c be a fixed constant. Propositional monotonic GDL can simulate a TM working in* TIME(n^c). *Propositional GDL can simulate a TM working in* SPACE(n^c).

Listing 10. Tape of unbounded size and input w of size n: unary encoding.

```
(init (access zero))
(← (next (access ?x)) (true (access ?x)))
(← (next (access (incr ?x))) (true (access ?x)))
(zerop zero)
(← (succp ?x (incr ?x)) (true (access ?x)))

For i ∈ {0,...,n − 1}, add
(input (incr ... (incr zero)) w_i)  with i nested incr.
(endinput (incr ... (incr zero)))  with n nested incr.
```

Monotonic GDL can simulate a TM working in $\text{TIME}(2^{n^c})$*. Bounded GDL can simulate a TM working in* $\text{SPACE}(2^{n^c})$*. GDL can simulate an unrestricted TM, using a bounded number of variables and only monotonic fluents.*

Proof. By combining Listings 1–5 with one time listing (6 or 7) and one tape listing (8, 9, or 10), we obtain a GDL description simulating a given TM M. The chosen listings determine the constraints on M and the properties satisfied by the description as indicated in Table 4.

Listings 9 and 10 are the only ones not using a constant number of variables or not satisfying the GRR, so we obtain a propositional GDL program as long as none of these two fragment is used. Similarly, the GRR is satisfied as long as Listing 10 is not used.

Positions of the game correspond to configurations of the machine and joint moves correspond to transitions. If we assume the players `exists` and `univer` to be rational and the player `random` to be making each decision uniformly at random, then we can conclude that the likelihood of reaching a position such that `accept` holds is more than ½ if and only if M accepts w.

The potential time/space constraint on the TM result in potential properties satisfied by the GDL program, and the type of the machine (non-deterministic, alternating, . . .) induces the number and type of agents in the corresponding game. Using Theorem 1, 2, 3 and Proposition 2, we derive the lower bounds for the results in Table 1.

Table 4. Effect of the time and tape listings added to Listings 1–5 on the TM restrictions and the GDL properties satisfied by the encoding.

Listing		Restriction on the TM	GDL properties		
Time	Tape		Monot.	GRR	Prop.
6	8	$\text{TIME}(n^c)$	✓	✓	✓
6	9	$\text{TIME}(2^{n^c})$	✓	✓	✗
6	10	—	✓	✗	✗
7	8	$\text{SPACE}(n^c)$	✗	✓	✓
7	9	$\text{SPACE}(2^{n^c})$	✗	✓	✗
7	10	—	✗	✗	✗

6 Conclusion

We have established the complexity of the adversarial reachability problem in the most natural fragments of GGP. That is, can a specified agent ensure a win assuming the other agents are adversaries or are playing a fix mixed strategy. Using backward induction, our results directly generalize to finding Nash equilibria in GGP when the number of agents is polynomial in the size of the GDL description. However, it is possible to create contrived GDL descriptions involving exponentially many agents. Whether our results carry over to finding Nash equilibria in arbitrary GDL games remains open at this stage.

GDL has recently been extended to allow defining imperfect information (II) games [13]. The only extensions to the language are that the official specification of the `random` role and the introduction of `sees`, a new keyword indicating the knowledge of each player on the state of the game. We have investigated how a `random` role affected the complexity. A natural avenue for future work is to extend the complexity landscape when the predicate `sees` is allowed.

A recent paper shows that the General Game Playing problem is universal in the sense that there is a tight relation between extensive-form games and models of GDL programs [14]. We have focused here on another dimension of universality: computability and complexity. Besides the Turing-completeness of GDL, we have shown that a wide range of standard complexity classes could be captured as finding a winning strategy in GGP via natural syntactic assumptions.

Bibliography

[1] Tom Bylander, 'The computational complexity of propositional STRIPS planning', *Artificial Intelligence*, **69**(1), 165–204, (1994).

[2] Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer, 'Alternation', *Journal of the ACM*, **28**(1), 114–133, (1981).

[3] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov, 'Complexity and expressive power of logic programming', *ACM Computing Surveys*, **33**(3), 374–425, (2001).

[4] Michael Genesereth and Nathaniel Love, 'General Game Playing: Overview of the AAAI competition', *AI Magazine*, **26**, 62–72, (2005).

[5] John Gill, 'Computational complexity of probabilistic turing machines', *SIAM J. Comput.*, **6**(4), 675–695, (1977).

[6] Nathaniel C. Love, Timothy L. Hinrichs, and Michael R. Genesereth, 'General Game Playing: Game Description Language specification', Technical report, LG-2006-01, Stanford Logic Group, (2006).

[7] Christos H. Papadimitriou, 'Games against nature', *Journal of Computer and System Sciences*, **31**(2), 288–301, (1985).

[8] Christos H. Papadimitriou, *Computational complexity*, Addison-Wesley, Reading, Massachusetts, 1994.

[9] Jussi Rintanen, 'Complexity of planning with partial observability', in *14th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 345–354. AAAI Press, (2004).

[10] Ji Ruan, Wiebe Van der Hoek, and Michael Wooldridge, 'Verification of games in the Game Description Language', *Journal of Logic and Computation*, **19**(6), 1127–1156, (2009).

[11] Walter J. Savitch, 'Relationships between nondeterministic and deterministic tape complexities', *Journal of Computer and System Sciences*, **4**(2), 177–192, (1970).

[12] Stephan Schiffel and Michael Thielscher, 'A multiagent semantics for the Game Description Language', in *Agents and Artificial Intelligence (ICAART)*, eds., Joaquim Filipe, Ana Fred, and Bernadette Sharp, volume 67 of *Communications in Computer and Information Science*, 44–55, Springer, Berlin / Heidelberg, (2010).

[13] Michael Thielscher, 'A general Game Description Language for incomplete information games', in *24th AAAI Conference on Artificial Intelligence (AAAI)*, pp. 994–999, Atlanta, (July 2010). AAAI Press.

[14] Michael Thielscher, 'The general Game playing Description Language is universal', in *22nd International Joint Conference on Artificial Intelligence*, IJCAI, pp. 1107–1112, (July 2011).