Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning
Main Track

Repairing General Game Descriptions

Yifan He[1] , **Munyque Mittelmann**[2] , **Aniello Murano**[3] , **Abdallah Saffidine**[4] , **Michael Thielscher**[1]
[1]University of New South Wales, Australia
[2]CNRS, LIPN, Sorbonne Paris North University, France
[3]University of Naples Federico II, Italy
[4]Potassco Solutions, Germany
{yifan.he1,mit}@unsw.edu.au, mittelmann@lipn.univ-paris13.fr, aniello.murano@unina.it,
abdallahs@gmail.com

Abstract

The Game Description Language (GDL) is a widely used formalism for specifying the rules of general games. Writing correct GDL descriptions can be challenging, especially for non-experts. Automated theorem proving has been proposed to assist game design by verifying if a GDL description satisfies desirable logical properties. However, when a description is proved to be faulty, the repair task itself can only be done manually. Motivated by the work on repairing unsolvable planning domain descriptions, we define a more general problem of finding minimal repairs for GDL descriptions that violate formal requirements, and we provide complexity results for various computational problems related to minimal repair. Moreover, we present an Answer Set Programming-based encoding for solving the minimal repair problem and demonstrate its application for automatically repairing ill-defined game descriptions.

1 Introduction

The Game Description Language (GDL) has been developed as a lightweight knowledge representation formalism for describing the rules of arbitrary finite games (Love, Genesereth, and Hinrichs 2006). It is used as the input language for general game-playing (GGP) systems, which can learn to play any new game from the mere rules without human intervention, thus exhibiting a form of general intelligence.

However, GDL can also model ill-defined games. For instance, a game may end up in a state where some players have no legal moves. In other cases, a game may have no sequence of joint actions that allows a player to win, resulting in an unfair game. Additionally, some games may run indefinitely, making it impossible to determine a winner.

To ensure that a GDL description is usable for the GGP competitions, the notion of *well-formed* descriptions has been proposed (Genesereth and Thielscher 2014). A well-formed game should ensure that each player has at least one legal action in any non-terminal state, the game must terminate after finitely many steps, and for each player, there is a sequence of joint moves leading to one of its winning states.

Writing correct GDL descriptions can be challenging for non-experts. Automated theorem proving has been proposed to assist game design by verifying whether a GDL game satisfies desired properties (Schiffel and Thielscher 2009; Ruan, Van Der Hoek, and Wooldridge 2009). But theorem provers can only verify GDL descriptions, not automatically repair them, which as of now can only be done manually.

Motivated by this, and in line with recent interest and work on automatically repairing planning domain descriptions (Gragera et al. 2023), in this paper, we introduce the more general problem of *automatically repairing game descriptions* given in GDL that do not comply with any given formal requirements, such as well-formedness. We consider a repair as a set of modifications to the *legality* and *game evolution* rules. To avoid "redesigning" the original game, we focus on minimal repairs. We consider game properties expressed in Game Temporal Logic (GTL) (Thielscher and Voigt 2010), a logic similar to the Linear Temporal Logic on finite traces (LTLf) (Bansal et al. 2023) with only the temporal operator "weak next", supporting both positive and negative properties that a game should, or should not, satisfy.

Our contribution is manifold. We provide a formal definition of game repair along with theoretical results on the minimal repair problem: sufficient conditions on when certain repair problems have or do not have solutions, and tight complexity results for different computational problems related to minimal repairs. We provide an encoding based on the Answer Set Programming (ASP) technique *Guess and Check* (Eiter and Polleres 2006) to solve the minimal repair problem, thus offering the first automated method for repairing GDL descriptions.

Related work. Our work is related to ASP-based approaches for formal model repair in various contexts, e.g. biological networks (Gebser et al. 2010), service-based processes (Friedrich et al. 2010; Lemos, Lynce, and Monteiro 2019), logic programs (Merhej, Schockaert, and De Cock 2017), and Petri nets (Chiariello, Ielo, and Tarzariol 2024).

Another piece of related work is repairing unsolvable planning domain descriptions (PDDL) (Gragera et al. 2023). While PDDL and GDL differ, our work is more general: we can not only repair reachability to the goal (aka. winnability in GDL) but *any* game properties expressible in GTL.

Given their syntactic and semantic similarity, ASP has been widely used for reasoning about GDL games, including for solving single or multiplayer games with ASP (Thielscher 2009; He, Saffidine, and Thielscher 2024), and automatically verifying game-specific GTL properties with ASP (Thielscher and Voigt 2010).

```
[cr] role(p).    base(win).  base(loss).      terminal:- true(win). goal(p,100):- true(win).
     input(p,l). input(p,r).                  terminal:- true(loss). goal(p, 0):- true(loss).
[c1] legal(p,l). [c2] next(loss):- does(p,l). [c3] next(win):- does(p,r).
```

Figure 1: GDL description of a simple game. $c_1, .., c_r$ are rule numbers, with c_r containing all rules other than c_1, c_2 and c_3. c_1 says l is a legal action for p. c_2 says that $loss$ will always hold after p does l, while c_3 says that win would hold if p could do r instead.

Outline. After providing necessary background on GDL and GTL, we define the GDL minimal repair problem and provide theoretical results in Section 3. In Section 4, we present an ASP encoding to find minimal repairs. We demonstrate its use with a case study in Section 5. We conclude in Section 6. Omitted proofs are available in an accompanying technical report (He et al. 2025).

2 Preliminaries

We assume readers to be familiar with basic concepts of logic programming with negation (Lloyd 1987) and Answer Set Programming (ASP) (Gebser et al. 2012).

2.1 Game Description Language

The Game Description Language (GDL) can be used to describe the rules of any finite game with concurrent moves. GDL uses a normal logic program syntax along with the following preserved keywords used to describe the different elements of a game (Genesereth and Thielscher 2014):

$role(P)$	P is a player
$base(F)$	F is a base proposition for game positions
$input(P, A)$	Action A is in the move domain of player P
$init(F)$	base proposition F holds in the initial position
$true(F)$	base proposition F holds in the current position
$legal(P, M)$	P can do move M in the current position
$does(P, M)$	player P does move M
$next(F)$	F holds in the next position
$terminal$	the current position is terminal
$goal(P, N)$	P gets N points in the current position

There are further restrictions for a set of GDL rules to be **valid** (Love, Genesereth, and Hinrichs 2006): $role$ can appear only in facts; $init$ and $next$ can only appear as heads of rules; $true$ and $does$ can only appear in rule bodies. Moreover, $init$ cannot depend on $true$, $does$, $legal$, $next$, $terminal$, or $goal$ while $legal$, $terminal$, and $goal$ cannot depend on $does$. Finally, valid game descriptions must be *stratified* and *allowed*—such normal logic programs always admit a **finite grounding** and a unique stable model/answer set (Lloyd 1987; Gebser et al. 2012). A valid description of a very simple game with a single player is given in Fig. 1.

Henceforth, we abbreviate "GDL Description" as GD.

A valid GD G over ground terms Σ can be interpreted as a multi-agent state transition system: Let $\beta = \{f \in \Sigma \mid G \models base(f)\}$ be the **base propositions** and $\gamma = \{(p, a) \in \Sigma \times \Sigma \mid G \models input(p, a)\}$ the **move domain** for the players. Suppose that $S = \{f_1, \dots, f_n\} \subseteq \beta$ is any given position and $A = \{p_1, \dots, p_k\} \to \Sigma$ any function that assigns to each of $k \geq 1$ players an action from their move domain. In order to use the game rules G to determine the state update, S needs to be encoded as a set of facts using keyword $true$: $S^{true} = \{true(f_1)., \dots, true(f_n).\}$ and the joint action A by a set of facts using keyword $does$: $A^{does} = \{does(p_1, A(p_1))., \dots, does(p_k, A(p_k)).\}$.

Definition 1 (Schiffel and Thielscher 2010)**.** *The* semantics *of a valid GDL description G is the state transition system*

- $R = \{p \in \Sigma \mid G \models role(p)\}$ *(player names)*
- $S_0 = \{f \in \beta \mid G \models init(f)\}$ *(initial state)*
- $T = \{S \subseteq \beta \mid G \cup S^{true} \models terminal\}$ *(terminal states)*
- $l = \{(p, a, S) \mid G \cup S^{true} \models legal(p, a)\}$ *(legal moves)*
- $u(A, S) = \{f \in \beta \mid G \cup S^{true} \cup A^{does} \models next(f)\}$ *(update)*
- $g = \{(p, v, S) \mid G \cup S^{true} \models goal(p, v)\}$ *(goal value)*

Let $\gamma(p) = \{a \mid (p, a) \in \gamma\}$ be the ***move domain of*** p, and $B = \{S \subseteq \beta \mid \exists p \in R. \forall a \in \gamma(p). G \cup S^{true} \not\models legal(p, a)\}$ be all states of G in which some player has no legal action. We represent a *valid sequence* of n steps starting at the initial state S_0 as $S_0 \xrightarrow{A_0} S_1 \xrightarrow{A_1} \dots \xrightarrow{A_{n-1}} S_n$ where $S_i \notin T$ for $i < n$ and all moves are legal in the corresponding state, i.e. $(p, A(p), S_i) \in l$ for each $p \in R$. Valid sequences are sometimes abbreviated as $(S_0, S_1, \dots, S_n)$. A sequence *terminates in n steps* if $S_n \in T$. E.g. in the game in Fig. 1, the only terminating sequence is $\{\} \overset{(p,l)}{\to} \{loss\}$. A sequence *ends in a non-playable state after n steps* if $S_n \in B \setminus T$.

A sequence $(S_0, \dots, S_m)$ is ***n*-max** if $m = n$ or $S_m \in T \cup B$ with $m < n$ (Haufe, Schiffel, and Thielscher 2012). The **horizon** of a game is the smallest n such that all n-max sequences terminate, end in a non-playable state, or enter a repeated state of the sequence.

Genesereth and Thielscher (2014) define a valid GD to be *well-formed* if:

1. For each p, there is a terminating sequence $(S_0, \dots, S_m)$ with $G \cup S_m^{true} \models goal(p, 100)$ (**weak winnability**).
2. No sequence ends in a non-playable state (**playability**).
3. All play sequences terminate (**termination**).

Our GD in Fig. 1 is not well-formed because it is not weakly winnable since action r is not legal for the player. A standard requirement for GD's used in the GGP competition is to be well-formed (Genesereth and Thielscher 2014).

We define a GD to be ***n*-well-formed** if it is well-formed and the game has a horizon of *no more than* n.

2.2 The Game Temporal Logic

The Game Temporal Logic (GTL) (Thielscher and Voigt 2010) is defined over GDs and allows the formulation of properties that involve finitely many successive game states.

Definition 2. *The set of* GTL formulas *over a GD G is:*

$$\varphi ::= q \mid \varphi \wedge \varphi \mid \neg\varphi \mid \bigcirc\varphi$$

*where q is an atom **true(f)** for some $f \in \beta$, or **legal(p,a)** for some $(p, a) \in \gamma$, or any other ground atom of predicates*

of G but which is neither ***init*** *nor* ***next*** *and which does not depend, in G, on* ***does***. $\bigcirc$ *is the temporal operator "next". Standard connectives like $\vee$ and $\rightarrow$ are defined as usual.*

The degree $deg(\varphi)$ *of a formula φ is the maximal "nesting" of the unary operator $\bigcirc$ in φ.*

Definition 3. *Let G be a valid GD and φ a GTL formula with $deg(\varphi) = n$. We say G* satisfies *φ (written $G \models_t \varphi$) iff for all n-max sequences $(S_0, \dots, S_m)$ we have $G, (S_0, \dots, S_m) \models_t \varphi$ as the following inductive definition:*

$$
\begin{array}{ll}
G, (S_i, \dots, S_m) \models_t q & \text{iff } G \cup S_i^{true} \models q \\
G, (S_i, \dots, S_m) \models_t \neg\varphi & \text{iff } G, (S_i, \dots, S_m) \not\models_t \varphi \\
G, (S_i, \dots, S_m) \models_t \varphi_1 \wedge \varphi_2 & \text{iff } G, (S_i, \dots, S_m) \models_t \varphi_1 \\
 & \text{and } G, (S_i, \dots, S_m) \models_t \varphi_2 \\
G, (S_i, \dots, S_m) \models_t \bigcirc\varphi & \text{iff } G, (S_{i+1}, \dots, S_m) \models_t \varphi \; (i < m) \\
G, (S_m) \models_t \bigcirc\varphi & \text{always}
\end{array}
$$

The *GTL model checking task* decides if $G \models_t \varphi$ holds, where φ is a GTL formula. Due to its syntactic and semantic similarities to GDL, ASP is a natural choice for GTL model checking. The encoding consists of (1) an ASP encoding of the GTL formula, (2) an ASP representation of the game rules, and (3) an action generator in ASP. Detailed description and examples of the encoding are available in the original article (Haufe, Schiffel, and Thielscher 2012).

Definition 4 (ASP encoding of GTL (Haufe, Schiffel, and Thielscher 2012))**.** *Let φ be a GTL, $i \in \mathbb{N}$, and $\eta(\varphi, i)$ a function that gives a unique atom of arity 0 for every φ and i. The* encoding $P_{\text{Enc}}(\varphi, i)$ of φ at level i *is recursively defined (below, $q(\vec{t})$ means the predicate q with arguments $\vec{t}$):*

- $P_{Enc}(q(\vec{t}), i) = \{\eta(q(\vec{t}), i) \,\texttt{:-}\, q(\vec{t}, i).\}$
- $P_{Enc}(\neg\varphi, i) = \{\eta(\neg\varphi, i) \,\texttt{:-}\, not\ \eta(\varphi, i).\} \cup P_{Enc}(\varphi, i)$
- $P_{Enc}(\varphi_1 \wedge \varphi_2, i) = \{\eta(\varphi_1 \wedge \varphi_2, i) \,\texttt{:-}\, \eta(\varphi_1, i), \eta(\varphi_2, i).\} \cup P_{Enc}(\varphi_1, i) \cup P_{Enc}(\varphi_2, i)$
- $P_{Enc}(\bigcirc\varphi, i) = \{\eta(\bigcirc\varphi, i) \,\texttt{:-}\, terminal(i)., \eta(\bigcirc\varphi, i) \,\texttt{:-}\, no\_play(i)., \eta(\bigcirc\varphi, i) \,\texttt{:-}\, \eta(\varphi, i+1).\} \cup P_{Enc}(\varphi, i+1)$

Definition 5 (Thielscher and Voigt 2010)**.** *The* ***Temporal-Extension*** *with horizon $n \geq 0$ of a valid GD G (denoted $P^n_{Ext}(G)$) is defined as $P^n_{Ext}(G) = \bigcup_{0 \leq i \leq n} \{c^i \mid c \in G\}$ where $\cdot^i$ replaces each occurrence of*

- $init(f)$ *by* $true(f, 0)$*; and* $next(f)$ *by* $true(f, i+1)$.
- $q(\vec{t})$ *by* $q(\vec{t}, i)$ *if $q \notin \{init, next\}$ is a predicate symbol that depends on* $true$ *or* $does$.

Definition 6 (Haufe, Schiffel, and Thielscher 2012)**.** *The* ***action generator*** *requires each player to make a legal action at each playable, non-terminal state up to step n: P^n_{legal} consists of the following clauses P_i for each $0 \leq i \leq n$.*

- $no\_play(i) \,\texttt{:-}\, role(R), not\ legal(R, A, i) : input(R, A).$
- $end(i) \,\texttt{:-}\, 1\ \{terminal(i);\ no\_play(i)\}.$
- $end(i) \,\texttt{:-}\, end(i-1).$ *for each* $i > 0$.
- $1\{does(R, A, i) : input(R, A)\}1 \,\texttt{:-}\, not\ end(i), role(R).$
- $\texttt{:-}\ does(R, A, i), not\ legal(R, A, i).$

Verifying $G \models_t \varphi$ is then achieved by checking that there is no $deg(\varphi)$-max with $G, (S_0, \dots, S_m) \not\models_t \varphi$.

Theorem 1 (Haufe, Schiffel, and Thielscher 2012)**.** *Let G be a valid GD, and φ be a GTL formula with $deg(\varphi)$=n. Then, $G \models_t \varphi$ iff the program $P^n_{legal} \cup P^n_{Ext}(G) \cup P_{Enc}(\varphi, 0) \cup \{\texttt{:-}\, \eta(\varphi, 0)\}$ has no stable model.*

We refer to the size of a logic program as the size of the ground program. Since the size of the ASP program according to Definitions 4–6 is polynomial w.r.t. the size of G and φ, it is known that GTL model-checking is in co-NP.

3 The GDL Repair Problem

In this section, we propose a formal definition of GDL game repair and then present theoretical results on this problem.

3.1 Problem Definition

We first define some auxiliary notation. For a valid GD G with base propositions β and move domain γ (cf. Section 2):

- $\mathcal{N} = \{next(f) \mid f \in \beta\}$ (the domain of $next$)
- $\mathcal{L} = \{legal(p, a) \mid (p, a) \in \gamma\}$ (the domain of $legal$)
- $\mathcal{F} = \{true(f),\ not\ true(f) \mid f \in \beta\}$
- $\mathcal{A} = \{does(p, a),\ not\ does(p, a) \mid (p, a) \in \gamma\}$
- $|P|$ = the total number of rules of a grounded program P.
- If r is a grounded rule, then $hd(r)$ denotes the atom in the head of r, and $bd(r)$ the set of literals in the body of r.

For a GTL formula φ and $op \in \{\wedge, \vee\}$ we define the abbreviation $nest(\varphi, op, n)$ recursively as follows:

- $nest(\varphi, op, 0) = \varphi$
- $nest(\varphi, op, n) = \varphi\ op\ (\bigcirc\ nest(\varphi, op, n-1))$.

For example, $nest(true(win), \vee, 2)$ denotes the GTL formula $true(win) \vee (\bigcirc(true(win) \vee (\bigcirc true(win))))$.

For the definition of repair, we assume that the GD have been **grounded** and transformed into a simplified form according to the following definition.

Definition 7. *Suppose G is a valid, grounded GD. G is in* **restricted form** *iff for every rule $r \in G$ the following holds:*

- *"legal" does not appear in $bd(r)$.*
- *"does" does not appear in $bd(r)$ unless $hd(r) \in \mathcal{N}$.*
- *If r is a* **legal rule***, i.e. $hd(r) \in \mathcal{L}$, then $bd(r) \subseteq \mathcal{F}$.*
- *If r is a* **next rule***, i.e. $hd(r) \in \mathcal{N}$, then $bd(r) \subseteq \mathcal{A} \cup \mathcal{F}$.*

The last two items state that for a GD in restricted form, the body of any legal (resp. next) rule can only have positive or negative atoms of the predicate $true$ (resp. $true$ or $does$).

Any valid GD has a finite grounding, and GDL in restricted form can be shown to be expressive enough to model all finite perfect-information games using the same construction as by Thielscher (2011) for GDL-II. Thus, for simplicity, we only consider GD in restricted form. Informally speaking, we are given a GD that satisfies (violates) undesired (desired, resp.) properties formulated in GTL. Then, to *repair* a GD, we allow the following possible changes:

1. modifying existing legal/next rules by

- deleting the entire rule or some literals from its body,
- changing the head of a legal rule to a new atom in $\mathcal{L}$ (or the head of a next rule to a new atom in $\mathcal{N}$, resp.), or
- adding one or more literals in $\mathcal{F}$ to the body of a legal rule (or literals in $\mathcal{A} \cup \mathcal{F}$ to the body of a next rule);

2. adding a *bounded* number of new legal/next rules.

The goal is to make minimal changes so that the resulting GD **satisfies or dissatisfies** some GTL properties. We only allow modifications to legal/next rules because changing other rules, such as the definition of base propositions, initial state, and goal, would be akin to defining a new game rather than repairing an existing one. To ensure that the set of possible repairs is always finite, we also assume a given bound on the number of legal/next rules that can be added.

Since the input GD is in restricted form, we may assume w.l.o.g. that it has a form $G = G_L \cup G_N \cup G_E \cup G_R$, where

- G_L (resp. G_N) contains all the legal (resp. next) rules.
- G_E has a fixed number of "empty" rules of the form $\emptyset$.
- G_R has all other rules (e.g., rules defining $init, goal, ...$).

G_E is the section reserved for new legal/next rules, with $\emptyset$ being a "placeholder" symbol that does not appear elsewhere in G. Let $G_O = G_L \cup G_N$ and $G_C = G_L \cup G_N \cup G_E$. We associate each $r \in G_C$ with a unique ID from the set $\mathcal{I} = \{1, \dots, |G_C|\}$, and we denote the i-th rule as r_i. We assume a simple order by which the IDs of the rules in G_L are smaller than the IDs of rules in G_N, which in turn are smaller than the IDs of rules in G_E. A *change tuple* formally defines an individual change to a GD. Repairing a GD involves applying a set of change tuples (aka. *repair*) to the GD *simultaneously*. To conveniently describe individual changes to a GD, we define the following notation:

- $Dom = (\{c\} \times (\{\emptyset\} \cup \mathcal{L} \cup \mathcal{N})) \cup (\{+, -\} \times (\mathcal{A} \cup \mathcal{F}))$

Definition 8 (Change tuples and repairs)**.** *Let G be a valid game description. A* **change tuple** *on G has the form $\langle i, (tp, l)\rangle \in \mathcal{I} \times Dom$. A* **repair** *$\mathcal{R}$ is a* **set** *of change tuples. Suppose $L_i^+ = \{l \mid \langle i, (+, l)\rangle \in \mathcal{R}\}$ and $L_i^- = \{l \mid \langle i, (-, l)\rangle \in \mathcal{R}\}$, for each $i \in \mathcal{I}$. The repair is* **valid** *iff for all $i \in \mathcal{I}$, all of the following hold:*

a) $|\{h \mid \langle i, (c, h)\rangle \in \mathcal{R}\}| \leq 1$.
b) $L_i^+ \subseteq (\mathcal{A} \cup \mathcal{F}) \setminus bd(r_i)$ *and* $L_i^- \subseteq bd(r_i)$.
c) If $r_i \in G_L$ *and* $\langle i, (c, h)\rangle \in \mathcal{R}$, *then* $h \in \{\emptyset\} \cup \mathcal{L}$.
d) If $r_i \in G_N$ *and* $\langle i, (c, h)\rangle \in \mathcal{R}$, *then* $h \in \{\emptyset\} \cup \mathcal{N}$.
e) If $hd(r_i) \in \mathcal{L}$ ***or*** $\langle i, (c, h)\rangle \in \mathcal{R}$ *for* $h \in \mathcal{L}$, *then* $L_i^+ \subseteq \mathcal{F}$.

The resulting GD, written $rep(G, \mathcal{R})$, after **applying a valid repair** *$\mathcal{R}$ to G is $r'_1 \cup \dots \cup r'_{|G_C|} \cup G_R$ where, for each $i \in \mathcal{I}$:*

i) $hd(r'_i) = h$ *if* $\langle i, (c, h)\rangle \in \mathcal{R}$, *otherwise* $hd(r'_i) = hd(r_i)$
ii) $bd(r'_i) = bd(r_i) \cup L_i^+ \setminus L_i^-$ *if* $hd(r'_i) \neq \emptyset$; *else* $bd(r'_i) = \{\}$.

Intuitively, the change tuple $\langle i, (c, h)\rangle$ means changing $hd(r_i)$ to h. Deleting rule i is achieved by setting $hd(r_i)$ to $\emptyset$ with the change tuple $\langle i, (c, \emptyset)\rangle$. $\langle i, (+, l)\rangle$ with $l \notin bd(r_i)$ means adding l to the body of r_i while $\langle i, (-, l)\rangle$ with $l \in bd(r_i)$ means deleting l from r_i's body. Since a GDL rule can only have one head atom, condition a) must hold for a valid repair. Conditions c) and d) imply that a legal rule cannot be changed to a next rule or vice versa. Condition ii) for the repair operation ensures that a rule r_i in the resulting GD has an empty body when it has an empty head.

Adding a new rule with head h is represented by $\langle i, (c, h)\rangle$ along with zero or more $\langle i, (+, l)\rangle$ tuples for some $h \neq \emptyset$ and $r_i \in G_E$. Up to $|G_E|$ new rules can thus be added. Condition e) requires that the body of any legal rule in the repaired GD does not contain a literal $l \in \mathcal{A}$, because *legal* cannot depend on *does* in a valid GD.

Since both $|G_C|$ as well as the domains for the base propositions and moves are fixed, the number of repairs is finite, and the space complexity of a repair is $O(|G_C| \cdot (|\mathcal{F}| + |\mathcal{A}|))$.

Example 1. *Let G be the GD consisting of the rules in Fig. 1 and $G_E = \{\texttt{[c4]}\ \emptyset\}$. Then, $G_L = c_1$, $G_N = c_2 \cup c_3$, $G_R = c_r$, and $|\mathcal{I}| = |G_C| = 4$. A valid (but not necessarily minimal) repair $\mathcal{R}$ of G is*

$$\{\langle 3, (-, does(p, r))\rangle,\ \langle 4, (c, legal(p, r))\rangle\}$$

In words, remove $does(p, r)$ from the body of c_3 and change the head of rule c_4 (the empty rule) to $legal(p, r)$. Thus, $rep(G, \mathcal{R}) = c_1 \cup c_2 \cup \{next(win).\} \cup \{legal(p, r).\} \cup c_r$.

To capture the total cost of a repair to G, we consider a cost function based on the cost of each individual change:

Definition 9 (cost function)**.** *Let G be a valid GD. A* cost function *over G is a mapping $cost : \mathcal{I} \times Dom \to \mathbb{N}_{>0}$. The* cost *$cost(\mathcal{R})$ of a repair $\mathcal{R}$ to G is $\sum_{\langle i, dom\rangle \in \mathcal{R}} cost(i, dom)$.*

Definition 10 (Minimal repair problem)**.** *A* ***repair task*** *is a tuple $\langle G, \Phi^+, \Phi^-, cost\rangle$, with G a GD, Φ^+ and Φ^- sets of GTL formulas over G, and a cost function.*

A ***solution*** *is a valid repair $\mathcal{R}$ with $rep(G, \mathcal{R}) \models_t \varphi^+$ for all $\varphi^+ \in \Phi^+$, and $rep(G, \mathcal{R}) \not\models_t \varphi^-$ for all $\varphi^- \in \Phi^-$.*

The ***minimal repair problem*** *(MRP) takes a repair task $\mathcal{T}$ as input and outputs the lowest-cost solution repair $\mathcal{R}$ to $\mathcal{T}$.*

Example 2. *Consider $\Phi^+ = \{\psi_{end}(n), \psi_{play}(n)\}$ and $\Phi^- = \{\psi_{loss}(p, n) \mid G \models role(p)\}$ for some GD G and $n > 0$, then the answer to the MRP $\langle G, \Phi^+, \Phi^-, cost\rangle$ is the lowest-cost repair to make G to be n-well-formed, where*

- $\psi_{end}(n) \stackrel{\text{def}}{=} nest(terminal, \vee, n)$
- $\psi_{lg}(p) = \bigvee_{a \in \gamma(p)} legal(p, a)$
- $\psi_{play}(n) \stackrel{\text{def}}{=} nest(terminal \vee \bigwedge_{G \models role(p)} \psi_{lg}(p), \wedge, n)$
- $\psi_{loss}(p, n) \stackrel{\text{def}}{=} nest(\neg terminal \vee \neg goal(p, 100), \wedge, n)$

$\psi_{end}(n)$ requires all n-max sequences to terminate or end in a non-playable state. $\psi_{play}(n)$ requires all players to have some legal actions per non-terminal state. $\psi_{loss}(p, n) \in \Phi^-$ ensures some n-max sequence terminate with $goal(p, 100)$.

For the GD in Example 1, if $n = 1$ and $cost(i, dom) = 1$ uniformly for all $i \in \mathcal{I}$ and $dom \in Dom$, the repair with the change tuple $\langle 4, (c, legal(p, r)\rangle$ of cost 1 is a solution to the MRP, and p can now weakly win the game (by playing r).

Existence of a Solution Repair. Repair tasks (or MRPs) may have no solutions, for example, if the GTL formulas are contradictory, or when $|G_E|$ is too small to allow for a repair. The following theorem states sufficient conditions to ensure that a game can always be repaired to be n-well-formed for any $n > 0$.

Theorem 2. *Let G be a GD and R be the set of players in G. For any $n > 0$, there exists a repair $\mathcal{R}$ on G such that $rep(G, \mathcal{R})$ is n-well-formed if **all** of the following hold:*

- *$G \cup S_0^{true} \not\models terminal$, and for all $p_i \in R$, $|\gamma(p_i)| \geq 2$.*
- *For each $p_i \in R$, there exists a state $S_i \subseteq \beta$ such that $G \cup S_i^{true} \models goal(p_i, 100) \wedge terminal$.*
- *$|G_L| + |G_E| \geq 2 \cdot |R|$ and $|G_N| + |G_E| \geq |\mathcal{N}| \cdot |R|$ as well as $|G_L| + |G_N| + |G_E| \geq (2 + |\mathcal{N}|) \cdot |R|$.*

Proof (Sketch). We show that the conditions ensure that G can always be repaired to be 1-well-formed, hence, it is n-well-formed for any $n > 0$. Note that *any* GD in restricted form with at most $|G_L| + |G_E|$ legal, at most $|G_N| + |G_E|$ next, and at most $|G_C|$ legal or next rules and with the same set of base propositions, move domain, and other rules G_R, are obtainable from G by a valid repair. If all conditions hold, there is a GD with $2 \cdot |R|$ legal rules and at most $|\mathcal{N}| \cdot |R|$ next rules so that all **1-max** sequences end in one of the $|R|$ terminal states, with the i-th one a winning state for p_i. □

A repair task may have no solution if $|G_E|$ is chosen to be too small. The following theorem provides sufficient conditions to confirm that a repair task has no solution, even for arbitrarily large $|G_E|$: If $\psi_{end}(n) \in \Phi^+$ for some n (i.e., the desired maximal horizon of the resulting game so that all n-max sequences terminate or end in a non-playable state) and the repair task has no solution with $|G_E|$ reaching some polynomial bound, increasing $|G_E|$ (i.e., allow for more new rules to be added) is not enough to make the task solvable.

Theorem 3. *Let $\mathcal{T} = \langle G, \Phi^+, \Phi^-, cost \rangle$ be a repair task with $\psi_{end}(n) \in \Phi^+$ for some n. Let $K = max(1, |\Phi^-|)$ and R the set of players in G. If $\mathcal{T}$ has no solution and $|G_E| \geq K \cdot (n+1) \cdot |\mathcal{L}| + n \cdot K \cdot |\mathcal{N}| \cdot (|R|+1)$, then any repair task $\mathcal{T}' = \langle G', \Phi^+, \Phi^-, cost' \rangle$, where $G' = G_L \cup G_N \cup G'_E \cup G_R$ and $|G'_E| \geq |G_E|$, also has no solution.*

3.2 Complexity Results

After having formally defined the game description repair problem, we now present some complexity results related to MRP by considering different decision problems.

- MRP$_b$: Given a repair task $\mathcal{T}$ and a cost **bound** $C \in \mathbb{N}$, decide if there is a solution repair of cost at most C.
- MRP$_t$: For a repair task $\mathcal{T}$ and change **tuple** $t = \langle i, dom \rangle$, decide if some lowest-cost solution repair to $\mathcal{T}$ contains t.

Theorem 4. *MRP$_b$ is Σ_2^P-complete.*

Proof. (Membership) We can guess a repair $\mathcal{R}$, validate if $\mathcal{R}$ is valid with cost $\leq C$, and calculate $G' = rep(G, \mathcal{R})$ in PTIME. We can check if $G' \not\models_t \varphi_i^-$ for all $\varphi_i^- \in \Phi^-$ in NP time, and if $G' \models_t \varphi_i^+$ for all $\varphi_i^+ \in \Phi^+$ in co-NP time (Th. 1). Thus, MRP$_b$ is in $\text{NP}^{\text{NP}} = \Sigma_2^P$.

(Hardness) As a Σ_2^P-hard problem, we reduce deciding the validity of a quantified Boolean formula (QBF) of the following form (Stockmeyer 1976) to an MRP$_b$:

$$\Psi = \exists x_1 \dots x_n \forall y_1 \dots y_m \;\; E, \;\; n, m \geq 1 \qquad \text{(E1)}$$

where $E = D_1 \vee \dots \vee D_k$, each $D_i = l_i^1 \wedge l_i^2 \wedge l_i^3$, and each l_i^j is a literal over variables $X \cup Y$ with $X = \{x_1, \dots, x_n\}$ and $Y = \{y_1, \dots, y_m\}$.

Let G be a GD with players $p_1, \dots, p_m$, base propositions $r_1, x_1, \dots, x_n, y_1^+, \dots, y_m^+, y_1^-, \dots, y_m^-$, and the move domain of each player being $\{pos, neg\}$. G contains the following rules with $G_L = R_1 \cup R_2$, $G_N = R_3 \cup \dots \cup R_6$, $|G_E| = 0$, and G_R is R_7 plus the definitions of *role*, *base*, and *input*:

- $R_1 = \bigcup_{i=1}^m \{[r_i] \;\; legal(p_i, pos).\}$
- $R_2 = \bigcup_{i=1}^m \{[r_{i+m}] \;\; legal(p_i, neg).\}$
- $R_3 = \bigcup_{i=1}^n \{[r_{i+2m}] \;\; next(x_i) \text{:-} true(x_i).\}$
- $R_4 = \bigcup_{i=1}^m \{[r_{i+2m+n}] \; next(y_i^+) \text{:-} does(p_i, pos).\}$
- $R_5 = \bigcup_{i=1}^m \{[r_{i+3m+n}] \; next(y_i^-) \text{:-} does(p_i, neg).\}$
- $R_6 = \{[r_{4m+n+1}] \; next(r_1).\}$
- $R_7 = \bigcup_{i=1}^k \{terminal \text{:-} \sigma(l_i^1), \sigma(l_i^2), \sigma(l_i^3), true(r_1).\}$

where $\sigma(l_i^j) = \begin{cases} true(y_s^+) & \text{if } l_i^j = y_s \text{ and } y_s \in Y \\ true(y_s^-) & \text{if } l_i^j = \neg y_s \text{ and } y_s \in Y \\ true(x_s) & \text{if } l_i^j = x_s \text{ and } x_s \in X \\ not\ true(x_s) & \text{if } l_i^j = \neg x_s \text{ and } x_s \in X \end{cases}$

Let $C = 2^n - 1$, $\Phi^- = \{\}$, and $\Phi^+ = \{\bigcirc terminal\}$, and $cost(i, (-, l)) = 2^{2m+n-i}$ when $2m+1 \leq i \leq 2m+n$ and $l \in bd(r_i)$; otherwise, $cost(i, (tp, l)) = 2^n$. Put in words, removing $true(x_i)$ from the body of the i-th rule of R_3 ($1 \leq i \leq n$) costs 2^{n-i} (i.e., $2^{n-1}, \dots, 2^1, 2^0$) while any other change costs 2^n. We require the repaired GD to satisfy $G' \models_t \bigcirc terminal$ and the repair must cost $\leq 2^n - 1$. Note that the MRP$_b$ $\langle G, \Phi^+, \Phi^-, cost, C \rangle$ can be constructed in PTIME w.r.t. the size of Ψ. Ψ is valid iff the MRP$_b$ is true:

"$\Rightarrow$": If Ψ is valid, there exists $X_T \subseteq X$ such that if we set all $x \in X_T$ to true and all $x \in X \setminus X_T$ to false, then for any $Y_T \subseteq Y$ if we assign all $y \in Y_T$ to true and all $y \in Y \setminus Y_T$ to false, some $D_j \in E$ must be satisfied. Consider a repair where $\langle i+2m, (-, true(x_i)) \rangle$ is a change tuple in $\mathcal{R}$ (i.e., we remove $true(x_i)$ from the i-th rule of R_3) iff $x_i \in X_T$. The repair costs $\leq 2^n - 1$, and *terminal* holds in the next state for any joint action A. Concretely, if $A^{does} = \bigcup_{i=1}^m \{does(p_i, a_i).\}$ where $a_i = pos$ iff $y_i \in Y_T$, then the j-th rule of form R_7 in the repaired GD can be activated in the next state S_1 where $S_0 \xrightarrow{A} S_1$.

"$\Leftarrow$": Similarly, if the MRP$_b$ has a solution $\mathcal{R}$, then assigning $x_i = \top$ iff $\langle i+2m, (-, true(x_i)) \rangle \in \mathcal{R}$, gives a satisfiable partial assignment to the existential variables in Ψ. □

Theorem 5. *MRP$_t$ is Δ_3^P-complete.*

Proof (Sketch). (Membership) Since the maximum possible cost of a valid repair is bounded (Def. 8 and 9) and the answer to MRP$_b$ is monotonic, we can do a binary search for the lowest cost bound C such that the MRP$_b$ returns "yes". Once we have that C, consider a new cost function: $cost'(j, dom') = 2 \cdot cost(j, dom')$ if $\langle j, dom' \rangle \neq \langle i, dom \rangle$, and $cost'(i, dom) = 2 \cdot cost(i, dom) - 1$. The answer to the MRP$_t$ is the same as MRP$_b$ with the new cost function and a cost bound $2 \cdot C - 1$. Since the number of MRP$_b$ oracle calls is polynomial w.r.t. the size of the input, MRP$_t$ is in Δ_3^P.

(Hardness) For a TQBF (Krentel 1992) of the form (E1), we create an MRP_t with the same G, Φ^+, Φ^-, and cost function as in Th. 4. The Δ_3^P-hard problem "Deciding if the lexicographically smallest satisfiable partial assignment to the existential variables of a TQBF of the form (E1) has $x_n = \top$" can be reduced to checking if the change tuple $\langle n+2m, (-, true(x_n))\rangle$ is in some optimal repair. □

The complexity class of the actual function problem MRP is given as follows, and the proof can be easily obtained by modifying the proof of Theorem 5.

Theorem 6. *MRP is $F\Delta_3^P$-complete.*

We conclude by pointing out that solving an MRP with $\Phi^+ = \{\}$ is simpler; e.g., fixing weak winnability only can be achieved by $\Phi^+ = \{\}$ and the same Φ^- as in Example 2 for some $n > 0$, so that for every player, there is an n-max sequence that terminates with $goal(p, 100)$.

Theorem 7. *MRP_b is NP-complete, MRP_t Δ_2^P-complete, and MRP $F\Delta_2^P$-complete if $\Phi^+ = \{\}$.*

4 Encoding

We discuss an ASP-based approach to solving minimal repair problems. Theorem 6 shows that MRP is $F\Delta_3^P$-complete, beyond the expressiveness of normal logic programs with optimization statements ($F\Delta_2^P$). Disjunctive logic programs with optimization statements, however, can express $F\Delta_3^P$ problems (Romero 2025). Hence, we can encode an MRP by a disjunctive ASP with optimization statements using the saturation technique (Eiter and Gottlob 1995), and extract the change tuples as well as the repaired GD from the stable model of the program.

With the help of an existing *guess and check* tool[1], a suitable disjunctive ASP can be automatically generated rather than having to be written from scratch. A *guess and check* (G&C) program (Eiter and Polleres 2006) is a pair of logic programs $\langle P_G, P_C\rangle$. $\mathcal{M}$ is a *stable model of* $\langle P_G, P_C\rangle$ iff $\mathcal{M}$ is a stable model of P_G, and $P_C \cup \mathcal{H}$ is unsatisfiable, where $\mathcal{H}$ is the set of atoms of the form $holds(x)$ in $\mathcal{M}$. The Potassco software translates a G&C program into a disjunctive program and solve it. To solve MRP using G&C, we map rules in the G_C part of a GD to ASP atoms as follows.

Definition 11. *Let G be a valid GD. We define a mapping τ from literals in G_C to tuples: for each $q \in \emptyset \cup \mathcal{A} \cup \mathcal{F} \cup \mathcal{L} \cup \mathcal{N}$,*

- $\tau(q) = \emptyset$ *if* $q = \emptyset$
- $\tau(q) = (ba, f)$ *if* $q = next(f)$*, for some* f
- $\tau(q) = (ac, (p, a))$ *if* $q = legal(p, a)$*, for some* p, a
- $\tau(q) = (pos, ba, f)$ *if* $q = true(f)$*, for some* f
- $\tau(q) = (neg, ba, f)$ *if* $q = not\ true(f)$*, for some* f
- $\tau(q) = (pos, ac, (p, a))$ *if* $q = does(p, a)$*, for some* p, a
- $\tau(q) = (neg, ac, (p, a))$ *if* $q = not\ does(p, a)$*, for* p, a

We define the inverse mapping from tuples to GDL atoms as τ^{-1} such that $\tau^{-1}(\tau(q)) = q$ and Π be the mapping from rules in G_C to a **set** *of ASP atoms of the form $ha()$ and $lit()$:*

- $ha(i, \tau(q)) \in \Pi(G_C)$ *iff* $hd(r_i) = q$ *for* $i \in \mathcal{I}$
- $lit(i, \tau(q)) \in \Pi(G_C)$ *iff* $q \in bd(r_i)$ *for* $i \in \mathcal{I}$

Π^{-1} is the inverse mapping such that $\Pi^{-1}(\Pi(G_C)) = G_C$.

For example, c_2 in Fig. 1 is mapped to $ha(2, (ba, loss))$ and $lit(2, (pos, ac, (p, l)))$, and Π^{-1} maps them back to c_2.

First, we create a program $P_{\mathcal{D}}$ defining the **domain** of: the IDs of old rules $1, \ldots, |G_O|$ (o_rule), the IDs of rules in G_C ($rule$), the IDs of empty rules (e_rule), the polarity of literals pos/neg (pol), and the set of atomic propositions ($atom$), which contains $atom(ac, (p, a))$ for each (p, a) in the move domain and $atom(ba, f)$ for each f as a base proposition. To avoid naming conflicts, we use o_ha (old head) and o_lit when referring to the ASP representation (cf. Def. 11) of the input G (i.e., $\Pi(G_C)$), and lit and hd for the repaired G'.

Based on $P_{\mathcal{D}}$ we create a **generator** $P_{\text{Gen}}(G)$ for the MRP,

- $P_{\text{Gen}}(G) = P_{\mathcal{D}} \cup \Pi(G_C) \cup \{(1), \ldots, (12)\}$ (cf. Fig. 2)

to manipulate the ASP representation of all the rules G'_C that can be obtained from G with some valid repair. We also create a GD G_{Inv} (not in restricted form) to **simulate** Π^{-1}:

- $G_{\text{Inv}} = \{(13), \ldots, (18)\}$ (cf. Fig. 2)

which ensures that if $\mathcal{X} = \Pi(G'_C)$ for some repaired GD G' obtained from G, $G_{\text{Inv}} \cup \mathcal{X} \cup G_R$ is equivalent $\Pi^{-1}(\mathcal{X}) \cup G_R$.

Before we discuss the G&C encoding, let's take a closer look at P_{Gen} and G_{Inv}. Clauses (1)-(2) specify that every rule in G_E can remain empty or be modified to a legal/next rule. We use $rtype(i, ba)$ (resp. $rtype(i, ac)$) to denote the *type of* the i-th rule as a next (resp. legal) rule. Clauses (3)-(4) state that every rule in G_O can be deleted (i.e., its head changed to $\emptyset$) or acquire a new head of the *same type*. Clause (5) says that if the new head $hd(r'_i)$ of a rule differs from the original one, the repair must contain the change tuple $\langle i, (c, hd(r'_i))\rangle$. We use $tup(i, tp, l)$ to denote the change tuple $\langle i, (tp, \tau^{-1}(l))\rangle$ (cf. Def. 11).

Clauses (6)-(10) model repairs to the rule bodies after we have fixed the heads. For every rule r_i that does not have a dummy head in the resulting GD (i.e., $hd(r'_i) \neq \emptyset$), (6) says that we can remove any literal l in the body with the change tuple $\langle i, (-, l)\rangle$; (7) says that we can add any $true(f)$ or its negation to the body if $f \in \beta$; and (8) says that we can add any $does(p, a)$ or its negation to the body if $(p, a) \in \gamma$ and r_i is a next rule. Clauses (9)-(10) ensure that if $hd(r'_i) = \emptyset$, $bd(r'_i) = \{\}$ (cf. Def. 8). Otherwise, $bd(r'_i)$ in the resulting GD contains all literals that have been added to r_i and all literals in $bd(r_i)$ in the input GD that have not been removed.

We also introduce constraints to improve the quality of the repair (clauses (11)–(12)): Rules with an atom that appears positively and negatively (a and $not\ a$) in the body, or with two different actions for the same player (i.e., $does(p, a)$ and $does(p, b)$ with $a \neq b$), are redundant, hence not allowed.

To sum up, due to the way $P_{\text{Gen}}(G)$ is encoded, by Definition 8, any stable model of $P_{\text{Gen}}(G)$ corresponds to a valid GD G' that can be obtained from G with some valid repair. The predicates lit and hd record the ASP representation of G'_C, and the predicate tup records all the change tuples.

We now prove that G_{Inv} models Π^{-1}. If $\mathcal{X} = \Pi(G'_C)$ for some repaired GD G', then $G'' = G_{\text{Inv}} \cup \mathcal{X} \cup G_R$ is equivalent to $G' = \Pi^{-1}(\mathcal{X}) \cup G_R$ in the sense that at any state, a ground atom holds in G' iff it holds in G''.

[1] Available at https://github.com/potassco/guess_and_check/

```
(1):  1{ha(I,(TP,F)):atom(TP,F);ha(I,∅)}1 :- e_rule(I).
(2):  rtype(I,TP) :- ha(I,(TP,F)), e_rule(I).
(3):  rtype(I,TP) :- o_ha(I,(TP,F)).
(4):  1{ha(I,(TP,F)):atom(TP,F);ha(I,∅)}1 :- rtype(I,TP), o_rule(I).
(5):  tup(I,c,F) :- rule(I), ha(I,F), not o_ha(I,F).
(6):  {tup(I,-,(Q,TP,F))} :- o_lit(I,(Q,TP,F)), not ha(I,∅).
(7):  {tup(I,+,(Q,ba,F))} :- atom(ba,F), pol(Q), not o_lit(I,(Q,ba,F)), rule(I), not ha(I,∅).
(8):  {tup(I,+,(Q,ac,F))} :- not o_lit(I,(Q,ac,F)), not ha(I,∅), rtype(I,ba), atom(ac,F), pol(Q).
(9):  lit(I,(Q,TP,F)) :- tup(I,+,(Q,TP,F)).
(10): lit(I,(Q,TP,F)) :- o_lit(I,(Q,TP,F)), not tup(I,-,(Q,TP,F)), not ha(I,∅).
(11): :- lit(I,(pos,TP,F)), lit(I,(neg,TP,F)).
(12): :- lit(I,(pos,ac,(P,A1))), lit(I,(pos,ac,(P,A2))), A1<A2.
(13): err_t(I) :- lit(I,(pos,ba,F)), not true(F).
(14): err_t(I) :- lit(I,(neg,ba,F)), true(F).
(15): err_d(I) :- lit(I,(pos,ac,(P,A))), not does(P,A).
(16): err_d(I) :- lit(I,(neg,ac,(P,A))), does(P,A).
(17): legal(P,A) :- ha(I,(ac,(P,A))), not err_t(I).
(18): next(F) :- ha(I,(ba,F)), not err_t(I), not err_d(I).
(19): :~ tup(I,TP,LIT). [cost(I,(TP,τ^-1(LIT))),I,TP,LIT]
```

Figure 2: The ASP encoding of GDL repair, symbols like $+$,$-$,$\emptyset$ should be replaced by constants in actual implementation

Theorem 8. *Let G be a valid GD, $\mathcal{M}$ a stable model of $P_{Gen}(G)$, and $\mathcal{X}$ the set of all atoms lit, ha in $\mathcal{M}$. If $q(\vec{t})$ is a ground atom with a predicate symbol in $\{true, does, legal, next\}$ or appears in G_R, then for any state S and joint action A over G's base propositions and move domain:*

- $\Pi^{-1}(\mathcal{X}) \cup G_R \cup S^{true} \cup A^{does} \models q(\vec{t})$ *iff* $G_{Inv} \cup \mathcal{X} \cup G_R \cup S^{true} \cup A^{does} \models q(\vec{t})$.

Proof (Sketch). Let $G' = \Pi^{-1}(\mathcal{X}) \cup G_R$ and $G'' = G_{\text{Inv}} \cup \mathcal{X} \cup G_R$. By construction of $P_{\text{Gen}}(G)$, G' must be a valid GD obtained from G with some valid repair. There are four cases. First, if $q \in \{true, does\}$, the statement trivially holds. Second, if $q \in G_R$, the statement holds because $q(\vec{t})$ solely depends on S^{true} and G_R since, as G is in restricted form, q cannot appear as head in $G_{\text{Inv}} \cup \mathcal{X}$ or $\Pi^{-1}(\mathcal{X})$. Third, if $q(\vec{t}) = legal(p, a)$ for some p, a. Since both G' and G'' are valid GDs, whether $legal(p, a)$ holds or not does not depend on A^{does}. Thus, $G' \cup S^{true} \models q(\vec{t})$ iff $ha(i, (ac, (p, a))) \in \mathcal{X}$ for some i and there is no f such that: i) $true(f) \in S^{true}$ and $lit(i, (neg, ba, f)) \in \mathcal{X}$, **or** ii) $true(f) \notin S^{true}$ and $lit(i, (pos, ba, f)) \in \mathcal{X}$. This is exactly what clauses (13)–(14) and (17) in Fig. 2 are modeling: (13) and (14) state that $err\_t(i)$ is justified iff for some f and i: i) $true(f) \in S^{true}$ and $lit(i, (neg, ba, f)) \in \mathcal{X}$, **or** ii) $true(f) \notin S^{true}$ and $lit(i, (pos, ba, f)) \in \mathcal{X}$. (17) enforces $G'' \cup S^{true} \models q(\vec{t})$ iff there is some i such that $ha(i, (ac, (p, a))) \in \mathcal{X}$ and $err\_t(i)$ is not justified. The final case when $q = next$ is analogous to the previous case (see (13)–(16) and (18)). □

Based on Theorem 1 and 8, we can check if the repaired game description $G' = G'_C \cup G_R$, with $\mathcal{X} = \Pi(G'_C)$ generated by $P_{\text{Gen}}(G)$, satisfies a given GTL property as follows.

Corollary 1. *Let G be a valid GD, φ be a GTL formula with $deg(\varphi) = n$, $\mathcal{M}$ a stable model of $P_{Gen}(G)$, and $\mathcal{X}$ the set of all atoms lit, ha in $\mathcal{M}$. Let $G' = \Pi^{-1}(\mathcal{X}) \cup G_R$, and $P_{Ver}(\varphi) = P^n_{legal} \cup P^n_{Ext}(G_{Inv} \cup G_R) \cup P_{Enc}(\varphi, 0) \cup \{:\!-\eta(\varphi, 0)\}$. Then, $G' \models_t \varphi$ iff $\mathcal{X} \cup P_{Ver}(\varphi)$ has no stable model.*

We return to the G&C framework. Consider an MRP instance $\langle G, \{\varphi^+_1, \ldots, \varphi^+_m\}, \{\varphi^-_1, \ldots, \varphi^-_n\}, cost\rangle$. It is trivial that in GTL, $G \models_t \varphi^+_i$ holds for all $1 \leq i \leq m$ iff $G \models_t \varphi_0$, where $\varphi_0 = \varphi^+_1 \wedge \ldots \wedge \varphi^+_m$. Thus, we can replace formulas in Φ^+ with the single formula φ_0. For the G&C, we define:

- $P_G = \{(19)\} \cup P_{\text{Gen}}(G) \cup P_H \cup \bigcup_{i=1}^{n} P_{\text{Ver}}(\varphi^-_i, i)$
- $P_C = P_{\text{Ver}}(\varphi_0) \cup P'_H$ (cf. Corollary 1).

where $P_H = \{holds(q(I, F)) :\!- q(I, F). \mid q \in \{lit, ha\}\}$ and $P'_H = \{q(I, F) :\!- holds(q(I, F)). \mid q \in \{lit, ha\}\}$.

Here, P_G consists of: First, the repair generator $P_{\text{Gen}}(G)$ to generate the ASP representation of $\mathcal{X} = \Pi(G'_C)$ of all possible resulting GDs $G' = G'_C \cup G_R$. Second, an ASP optimization statement written as a weak constraint (19) (cf. Fig. 2) which says that the stable model of P_G should minimize the total cost of the change tuples, ensuring that the repair output by $P_{\text{Gen}}(G)$ is an optimal one.

Third, P_G uses a checker $P_{\text{Ver}}(\varphi^-_j, j)$ for every φ^-_j to ensure that $G' \not\models_t \varphi^-_j$. $P_{\text{Ver}}(\varphi^-_j, j)$ modifies $P_{\text{Ver}}(\varphi^-_j)$ in Corollary 1 by *extending* each occurrence of $q(\vec{t})$ with a constant j to $q(\vec{t}, j)$ if q is a predicate that is neither ha nor lit.

For example, $does(R, A, i)$ in P^n_{legal} become $does(R, A, i, j)$ (Def. 6). The extension is crucial because with $P_{\text{Ver}}(\varphi)$, we can only show $G' \not\models_t \varphi$ for a single φ by showing that $\mathcal{X} \cup P_{\text{Ver}}(\varphi)$ has a stable model, which involves generating a $deg(\varphi)$-max sequence such that $G', (S_0, ..., S_k) \not\models_t \varphi$. In our case, we need to show $G' \not\models_t \varphi^-_j$ for all $j \leq n$, which requires generating n sequences simultaneously that do not interfere with each other, where the j-th one dissatisfies φ^-_j. To do so, we create n "copies" of $P_{\text{Ver}}(\varphi)$ letting them *share* the set $\mathcal{X}$ and the j-th copy generates a sequence that dissatisfies φ^-_j to prove $G' \not\models_t \varphi^-_j$.

Finally, P_H in P_G "wraps" all atoms of lit and ha (i.e., atoms in $\mathcal{X}$) of a stable model of P_G with the G&C preserved predicate $holds$, allowing G' to be *shared* between P_G and P_C. Let $\mathcal{H}$ denote the set of all atoms of the form $holds(x)$ in the stable model of P_G. The P'_H part in P_C "decodes" every instances in $\mathcal{H}$ back to lit and ha (i.e., the set $\mathcal{X}$). (Recall that in G&C, the stable model of P_G is the stable model of the overall program iff $\mathcal{H} \cup P_C$ is unsatisfiable.) This is equivalent to $P_{\text{Ver}}(\varphi_0) \cup \mathcal{X}$ has no stable model, which proves that $G' \models_t \varphi_0$ (cf. Corollary 1).

To sum up, P_G generates an answer to the MRP by only considering the constraints in Φ^-. The repaired GD is shared between P_G and P_C via the special G&C predicate $holds$. Letting P_C together with the set of $holds$ instances to have no stable models ensures that all formulas in Φ^+ hold, which in turn implies that the answer generated by P_G is an answer to the overall MRP. Our construction also indicates that for MRP with $\Phi^+ = \{\}$ (cf. Theorem 7), P_G alone, a normal logic program with optimization statements, suffices to solve the problem.

5 Case Study

The motivation for introducing the GDL repair problem is that, in practice, game descriptions might violate the intention of the game designers. Once these violated properties are expressed in GTL and a cost function specified manually, we can use our encoding in Section 4 to automatically generate minimal changes to fix a GD. Our encoding works for **any** GDL repair task. We demonstrate this through a simple case study on Tic-Tac-Toe, beginning with the well-formedness property and then others.

Instance Description In Tic-Tac-Toe, two players x and o alternate in marking cells on the board, beginning with x. Turn-taking is modeled by the following pair of GDL rules:

1. $next(control(o)) :- true(control(x))$.
2. $next(control(x)) :- true(control(o))$.

The player who does not have control in a state can only perform a $noop$ action with no effect. The player who first places three of their marks in a horizontal, vertical, or diagonal row is the winner. If all 9 cells of the board are marked and neither x nor o wins, the game ends in a draw.

We purposefully break the GD by removing the second rule. As a result, player x can never take control after step 1, and o cannot take control after step 2. Consequently, the broken GD is not well-formed as it violates both the weak winnability and termination properties.

Experimental Setup To repair the broken GD, we need to formulate an MRP of the form $\langle G, \Phi^+, \Phi^-, cost\rangle$ where $G = G_L \cup G_N \cup G_R \cup G_E$ (cf. Section 3.1). We need to specify: $|G_E|$, the maximum number of new legal/next rules we can add; Φ^+, the GTL properties that the repaired GD should satisfy; Φ^-, the GTL properties that the repaired GD should dissatisfy; and a cost function.

For simplicity, we set $|G_E| = 2$ and use the following uniform cost function for all our experiments:

- $cost(i, (+, l)) = cost(i, (-, l)) = 1$
- $cost(i, (c, h)) = 1$, if $r_i \in G_E$
- $cost(i, (c, \emptyset)) = |bd(r_i)| + 1$
- $cost(i, (c, h)) = 2 \cdot |bd(r_i)| + 2$, if $h \neq \emptyset$ and $r_i \in G_o$

This models the "editing" cost. Adding or deleting a body literal costs 1, as does adding the head of a new legal/next rule. We model deleting a rule (i.e., setting the head to $\emptyset$) as deleting the head and all literals in the *old* body of a rule. Likewise, replacing the head of an old rule is considered a significant change, costed as: removing the old rule and creating a new rule with a new head and the same body.

All MRPs are solved by G&C with Clingo 5.7.2 with the inverse linear search-based core-guided optimization configuration (Lifschitz 2019) on a Latitude 5430 laptop [2].

Repairing the well-formedness property The most fundemantal, general property of any "good" GD is well-formedness, i.e., playability, termination, and weak winnability. We can repair a GD to be n-well-formed, for a give user-specified n as the desired maximal horizon for the repaired game. In practice, n would be based on domain knowledge of the game that the GD was intended to describe. For Tic-Tac-Toe, the desired maximal horizon is obviously 9, the number of cells that can be marked. Hence, in the MRP, we specify $\Phi^+ = \{\psi_{play}(9), \psi_{end}(9)\}$ and $\Phi^- = \{\psi_{loss}(x, 9), \psi_{loss}(o, 9)\}$ (cf. Example 2).

Our encoding from Section 4 is able to automatically solve this MRP with an optimal repair at a cost of 1: The solution generated by our G&C program is to simply create a new rule with an empty body: $next(control(x))$. The resulting GD is well-formed, and it is syntactically close to the input, requiring only a single modification to the input.

Refining the repair with additional constraints While the solution to our example Tic-Tac-Toe MRP successfully restores well-formedness, this may not be the only intended property for a game. Specifically, after repairing the GD with the creation of the simple fact $next(control(x))$, the base proposition (aka. *fluent*) $control(x)$ is true in all positions of the game. As a result, the repaired GD allows player x to always take control after step 1, which changes Tic-Tac-Toe from a turn-taking game to a simultaneous-move game.

In a GDL description, having a next rule with an empty body is usually undesired as it allows a base proposition to persist unconditionally across all play sequences after step 1. Such a behavior can reduce the dimension of the state space of a game and hence oversimplify the game.

[2] Source code link: https://github.com/hharryyf/gdlRepair

If a game designer wants to avoid this behavior, the following GTL formula, a kind of "fluent dynamic constraint" can be added to Φ^-:

$$\psi_s(f,n) \stackrel{\text{def}}{=} \neg terminal \wedge \bigcirc nest(true(f), \wedge, n-1)$$

This ensures that in the repaired GD, either the game terminates at S_0, or there exists some n-max sequences where $true(f)$ does not hold continuously after step 1 which effectively rules out repairs that introduce rules like $next(f)$.

Refining the MRP in this way means to find a repair with $\Phi^- = \{\psi_{loss}(x,9), \psi_{loss}(o,9), \psi_s(control(x),9)\}$ and $\Phi^+ = \{\psi_{play}(9), \psi_{end}(9)\}$.

Now, the optimal repair generated by our G&C has a cost of 2 to satisfy both well-formness and the additional fluent dynamic constraint. One repair output by G&C suggests adding the rule: $next(control(x)) :- true(control(o))$, which successfully restores the GD to the standard GDL description of Tic-Tac-Toe.

Further refine the repair with the turn-taking constraint While the above minimal repair is arguably the desired one, the automatic solver G&C outputs another lowest-cost repair of cost 2, namely, adding the rule: $next(control(x)) :- not\ does(x, mark(1,1))$.

The new rule says that x will take control of the game in the next step as long as that player does not mark the cell $(1,1)$ in the current round. This is a perfectly appropriate repair as the resulting variant of Tic-Tac-Toe is well-formed and does satisfy the additional constraint from above. However, this repair may still be considered undesirable because if x begins with marking the cell $(1,2)$ in step 1, for example, the player keeps control in step 2. In this case both $control(x)$ and $control(o)$ will be true in step 2, and hence they can still mark cells simultaneously in some game states.

If it is desirable to ensure that an n-well-formed two-player game after repairing satisfies a strict turn-taking property, which is to say that, in our example, at each step of the game either x or o take control of the game but not both, we can introduce the following additional GTL constraint:

$$\psi_{con}(n) \stackrel{\text{def}}{=} nest(\psi_x \vee \psi_o, \wedge, n)$$

where, $\psi_x = true(control(x)) \wedge \neg true(control(o))$ and $\psi_o = true(control(o)) \wedge \neg true(control(x))$.

To repair Tic-Tac-Toe with this extra constraint, the MRP uses: $\Phi^- = \{\psi_{loss}(x,9), \psi_{loss}(o,9), \psi_s(control(x),9)\}$ and $\Phi^+ = \{\psi_{con}(9), \psi_{play}(9), \psi_{end}(9)\}$.

Now the optimal repair still has cost 2, but the undesired repair suggested by G&C when only considering the constraint to rule out a static fluent $control(x)$ is eliminated. One automatically generated repair suggests to add the rule: $next(control(x)) :- does(x, noop)$. This new rule states that whenever x does $noop$, it will take control in the next step. In Tic-Tac-Toe, x can only do $noop$ when o takes control, which means this new rule is effectively "equivalent" to the original rule that we deleted—namely, x taking control in the next step if o takes control in the current step.

More importantly, after introducing the turn-taking constraint, all lowest-cost repairs computed by G&C result in a GD *equivalent* to the originally correct Tic-Tac-Toe game description, in the sense that at every state of the repaired game and the original Tic-Tac-Toe GD, the same set of legal actions are available to the players and each joint action leads to the same successor state in all these repaired games.

Summary The following table summarizes the lowest repair cost (C_{opt}) for each of the 3 MRPs that we considered in our study: repair the well-formedness property only (WF), repair the well-formedness property with the fluent dynamic constraint (WF + FD), and repair all 3 properties simultaneously (WF + FD + TT). We also record the time for G&C to find the first optimal repair (T_1), and the time to compute all optimal repairs (T_a).

Property	C_{opt}	T_1 **(sec)**	T_a **(sec)**
WF	1	34.76	38.72
WF + FD	2	50.99	326.41
WF + FD + TT	2	122.29	195.15

We observe that G&C can find an appropriate repair for Tic-Tac-Toe in all 3 situations in a feasible amount of time with the resulting GD syntactically close to the original. Moreover, the case study demonstrates the ability of our encoding to automatically repair ill-defined GDL descriptions. Upon detecting—either manually or using a GTL theorem prover (Haufe, Schiffel, and Thielscher 2012)—that a given GD violates human intentions, one can specify the desired properties as GTL formulas and use our approach to automatically generate a repaired GD that satisfies the intended properties while remaining syntactically closest to the original GD. And, if the repair suggested by G&C is still unsatisfactory, one can refine the repair by defining new MRPs with additional GTL properties based on *additional* human knowledge of the resulting GD, until the generated repair is satisfactory.

6 Conclusion

We investigated the problem of repairing GDL descriptions, with a focus on minimal repairs. We established sufficient conditions under which certain repair problems have, or do not have, solutions. We proved tight complexity bounds for the minimal repair problem and introduced the first automated method for repairing GDL descriptions using ASP, thereby extending the capabilities of automated theorem proving in GGP from mere fault detection to actual *rectification*. One potential limitation of our automated method concerns efficiency and scalability. This is due to the complexity of the problem, but may be improved with the development of more efficient disjunctive ASP solvers.

For future work, we plan to design a broken GDL description dataset from existing descriptions (GGP 2023) and systematically evaluate the performance of our encoding. We also intend to explore whether certain fragments of the repair problem (e.g., the $F\Delta_2^P$ fragment in Theorem 7) can be solved more efficiently. Another future direction is to explore the repair of game properties formulated in more expressive logics, such as LTLf (Bansal et al. 2023), as well as to investigate how our approach can be extended to games with imperfect information (Thielscher 2010), enabling the repair of epistemic properties (Haufe and Thielscher 2012).

Acknowledgements

We sincerely thank the anonymous reviewers for their insightful and thorough feedback and for the care they invested in reviewing our work. This project has received funding from the EU's H2020 Marie Sklodowska-Curie project with grant agreement No 101105549. This work has also been partially funded by the MUR PRIN project RIPER (No. 20203FFYLK), and the PNRR MUR projects FAIR, INFANT, and APLAND.

References

Bansal, S.; Li, Y.; Tabajara, L. M.; Vardi, M. Y.; and Wells, A. 2023. Model checking strategies from synthesis over finite traces. In *International Symposium on Automated Technology for Verification and Analysis*, 227–247. Springer.

Chiariello, F.; Ielo, A.; and Tarzariol, A. 2024. An ILASP-based approach to repair Petri nets. In *Proceedings of LPNMR*, volume 15245 of *LNCS*, 85–97. Springer.

Eiter, T., and Gottlob, G. 1995. On the computational cost of disjunctive logic programming: Propositional case. *Annals of Mathematics and Artificial Intelligence* 15:289–323.

Eiter, T., and Polleres, A. 2006. Towards automated integration of guess and check programs in answer set programming: a meta-interpreter and applications. *Theory and Practice of Logic Programming* 6(1-2):23–60.

Friedrich, G.; Fugini, M. G.; Mussi, E.; Pernici, B.; and Tagni, G. 2010. Exception handling for repair in service-based processes. *IEEE Transactions on Software Engineering* 36(2):198–215.

Gebser, M.; Guziolowski, C.; Ivanchev, M.; Schaub, T.; Siegel, A.; Thiele, S.; and Veber, P. 2010. Repair and prediction (under inconsistency) in large biological networks with answer set programming. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, 497–507.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2012. Answer set solving in practice. *Synthesis lectures on artificial intelligence and machine learning* 6(3):1–238.

Genesereth, M., and Thielscher, M. 2014. General game playing. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 8(2):1–229.

GGP. 2023. GGP base repository. http://games.ggp.org/base/.

Gragera, A.; Fuentetaja, R.; García-Olaya, Á.; and Fernández, F. 2023. A planning approach to repair domains with incomplete action effects. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 33, 153–161.

Haufe, S., and Thielscher, M. 2012. Automated verification of epistemic properties for general game playing. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*.

Haufe, S.; Schiffel, S.; and Thielscher, M. 2012. Automated verification of state sequence invariants in general game playing. *Artificial Intelligence* 187–188:1–30.

He, Y.; Mittelmann, M.; Murano, A.; Saffidine, A.; and Thielscher, M. 2025. Repairing general game descriptions (extended version). *arXiv preprint arXiv:2508.10438*.

He, Y.; Saffidine, A.; and Thielscher, M. 2024. Solving two-player games with qbf solvers in general game playing. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, 807–815.

Krentel, M. W. 1992. Generalizations of Opt P to the polynomial hierarchy. *Theoretical Computer Science* 97(2):183–198.

Lemos, A.; Lynce, I.; and Monteiro, P. T. 2019. Repairing boolean logical models from time-series data using answer set programming. *Algorithms for Molecular Biology* 14:1–16.

Lifschitz, V. 2019. *Answer set programming*. Springer.

Lloyd, J. W. 1987. *Foundations of Logic Programming*. Springer, second, extended edition.

Love, N.; Genesereth, M.; and Hinrichs, T. 2006. General game playing: Game description language specification. Technical Report LG-2006-01, Stanford University. `ggp.stanford.edu/readings/gdl_spec.pdf`.

Merhej, E.; Schockaert, S.; and De Cock, M. 2017. Repairing inconsistent answer set programs using rules of thumb: A gene regulatory networks case study. *International Journal of Approximate Reasoning* 83:243–264.

Romero, J. 2025. *A General Framework for Preferences in Answer Set Programming*. Ph.D. Dissertation, University of Potsdam.

Ruan, J.; Van Der Hoek, W.; and Wooldridge, M. 2009. Verification of games in the game description language. *Journal of Logic and Computation* 19(6):1127–1156.

Schiffel, S., and Thielscher, M. 2009. Automated theorem proving for general game playing. In *Proceedings of IJCAI*, 911–916.

Schiffel, S., and Thielscher, M. 2010. A multiagent semantics for the game description language. In Filipe, J.; Fred, A.; and Sharp, B., eds., *Agents and Artificial Intelligence*, 44–55. Berlin, Heidelberg: Springer Berlin Heidelberg.

Stockmeyer, L. J. 1976. The polynomial-time hierarchy. *Theoretical Computer Science* 3(1):1–22.

Thielscher, M., and Voigt, S. 2010. A temporal proof system for general game playing. In Fox, M., and Poole, D., eds., *Proceedings of the AAAI Conference on Artificial Intelligence*, 1000–1005.

Thielscher, M. 2009. Answer set programming for single-player games in general game playing. In Hill, P. M., and Warren, D. S., eds., *Proceedings of the International Conference on Logic Programming*, 327–341. Springer.

Thielscher, M. 2010. A general game description language for incomplete information games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, 994–999.

Thielscher, M. 2011. The general game playing description language is universal. In *Proceedings of IJCAI*, 1107–1112.