M. FAREED ARIF

A LOGIC PROGRAMMING APPROACH FOR CONCURRENT REACHABILITY GAMES

A LOGIC PROGRAMMING APPROACH FOR CONCURRENT REACHABILITY GAMES

M. FAREED ARIF

Master in Computational Logic

Supervisor: Prof. Dr. Christel Baier
Algebraische und logische Grundlagen der Informatik
Fakultat Informatik
Technische Universität Dresden

September 2011

My work is a game, a very serious game.

— M.C. Escher (1898 – 1972)

ABSTRACT

An open system is a system which continuously interacts with its environment and made an explicit distinction between itself and its environment. It is a difficult task to verify the logical correctness of an open system because of its continuous interacts with its environment. A principle question in open system verification is the reachability question in which the goal is to determine whether a system can reach to a set of goal states. In order to solve this verification problem, open system has been modeled as a concurrent reachability game between the system and its environment. A concurrent reachability game is a two-player game with reachability objectives which captures the interactions of a system with its environment thus making it useful in specification and verification of open systems. In this game, the question of reachability is to determine whether a player can force a play in a game to a given set of goal states. To solve this reachability question, we had specified the concurrent reachability games in Game Description Language (GDL), a declarative language to define games, and then constitute a general game playing program using answer set programming for solving it.

ACKNOWLEDGMENTS

Und ich möchte allen in Deutschland danken,
dass sie mir geholfen haben dieses Wissen zu erlangen.

CONTENTS

Mathematics is a game played according to certain simple rules with meaningless marks on paper.

— David Hilbert (1862 – 1943)

1 INTRODUCTION

A `Concurrent Reactive System` comprises of a number of components that are working concurrently and maintaining an event-driven interaction with its environment. The examples are microprocessors, communication and security protocols, control of nuclear plants and air traffic control systems etc. Mang [46] argues that the logical correctness in the design of concurrent reactive systems are of paramount importance because any logical error in these systems could be devastating. A prominent incident causing a million of dollars loss is a bug in Intel P5 Pentium floating point unit (Pentium FDIV) [17]. Therac-25 [44] debacle is another example of human life loss. Therefore, verifying the correctness of such complex systems are of great interest.

Different formal methods have been investigated to verify the logical correctness of concurrent reactive systems [37, 46, 18]. Traditional approaches tend to view concurrent reactive systems as unstructured graphs [46] whereas in more recent approaches [46, 18, 5, 3] a concurrent reactive system which a collection of interacting components, its each component is viewed as an open system [46]. Alur et al. [6] argues in favor of adopting games for encapsulating the behavior of concurrent reactive systems because games constitute a natural model for open systems for solving control and verification problems [18].

1.1 CLOSED SYSTEMS VS OPEN SYSTEMS

A basic distinction between an `Open System` and a `Closed System` is the existence of an external `environment`. The behavior of a closed system is only determined by its internal structure because it does not interact with an external environment. It is in contrast to an open system where the behavior of a system is jointly determined by its internal structure and the behavior of its environment [18].

In order to verify the correctness of an open system or a closed system, one needs a model to capture its behavior. In a closed system where the system does not interact with its environment, Kripke structure [39] provides such a model [9]. For an open system, we need to model the internal structure of the system and captures its interactions with the environment.

1.2 OPEN SYSTEMS AS GAMES

Alur et al. [6] argued that the games are proving to be a natural model for open system because games generalize transition systems [18]. Abadi et al. [1] state that the games have been widely used to analyze and solve control problems for open systems. In particular, two-player games have proven to be an expressive model for open systems, enabling a distinction between the non-deterministic choices that originate within the system and the choices that originate with the environment [18]. Games have also been used in the specification and the verification of the interactions between component and their environment [4, 5]. All essential notations - composition, refinement, specification and verification- related to open systems can be phrased in terms of games between two players, the system and its environment [18]. Concurrent reachability game is such a two-player game. In this concurrent game, we have two players namely Player 0 and Player 1. Consider a player σ for $\sigma \in \{0, 1\}$, then opponent is $\overline{\sigma}$ (i.e., $\overline{\sigma} = 1 - \sigma$). The game has a starting state and in each round both players σ and $\overline{\sigma}$ choose a move independently and the parallel execution of the moves determines the successor state of the game. The game has a reachability objective defined for each player. The reachability objective for player σ is to reach a set of goal states and the objective its opponent $\overline{\sigma}$ is to prevent σ from achieving its goal. Krishnendu Chatterjee and Henzinger [1] state that the correctness properties (i.e., deadlock-free etc) of an open system can be modeled as reachability objectives for a two-player concurrent game.

1.3 REACHABILITY QUESTION

One of the central problems in system verification is the reachability question: "*given an initial state and a target state, can the system get from initial to the target state*" [3]. In closed systems settings where the system does not interact with its environment is best modeled as a state-transition graph and the reachability question is reduced to graph reachability [35].

It is in contrast with the dynamics of an open system where concurrent game captures the interaction of an open system with its environment and the question of reachability is to determine, *"can a player force a play in the game to a given set of goal states?"* [3]. Forcing a play by a players means that the player selects actions in such a way that the play reaches a state which is among the set of its goal states.

1.4 GAMES IN GDL

Using logic programming to solve a concurrent reachability game, we need to formalize the game description in a logic-based language. For this purpose, we had used Game Description Language (GDL) [45]. Game

Description Language (GDL) [45] has been developed to formalize the rules of any finite n-player game in such a way that the game description can be automatically processed by a general game player (GGP) [56, 41]. A general game player (GGP) is a program that plays any arbitrary game specified in GDL without any human intervention [45]. Game Description Language (GDL) was informally introduced in [45]. Love et al. [45] describe Game Description Language (GDL) as a variant of Datalog [2] and does not provide a mathematically precise syntax and semantics of the language. Love et al. emphasis is to explain the idea and the framework of General Game Playing (GGP) [27] rather than defining the syntax and semantics for GDL.

In this report, we solve the reachability question for a restricted case of two-player concurrent games where the game is specified in Game Description Language (GDL) and an answer set program provides the encoding of the reachability algorithm. We use Answer Set Programming (ASP) [47] to write the general game playing program because of its closeness in syntax and semantics with GDL. Answer Set Programming (ASP) [47] is a pure declarative programming language oriented towards solving difficult search program primarily NP-hard problems [47]. Guo, Thielscher [30, 62] had implemented general game players as answer set programs but only for single player games. Thielscher in his paper [62] provides a mapping from GDL specification onto a normal logic program and an answer set program which perform a complete, depth restricted forward search but only in case of single-player games. We extend this approach for multi-player games. Our game playing program tries to compute a winning strategy for any given player which helps it in winning the game thus solving the reachability question for two-player concurrent reachability games.

1.5 THESIS OVERVIEW

The out of this report is as follows:

- `Chapter 2` provides a reachability game structure which is used to formulate *Concurrent Reachability Games*. In order to develop a better understanding, we conclude `Chapter 2` with an intuitive concurrent reachability game example.

- In `Chapter 3` a declarative syntax and semantics of Game Description Language (GDL) is presented. This chapter also provides the more intuitive transition-based semantics for Game Description Language (GDL).

- In `Chapter 4`, we use a declarative language namely Answer Set Programming (ASP) to solve the reachability problem in concurrent games. We present a general game playing program that computes the winning strategy for any given player thus solving the reachability question for two-player concurrent reachability games.

- We had implemented the game playing program using answer set programming. In `Chapter 5`, we present the implementation details and results of this general game player.

We conclude the discussion by providing some future directions to our work in the final chapter of this report.

The best games are not those in which all goes smoothly and steadily toward a certain conclusion, but those in which the outcome is always in doubt.

— George B. Leonard (1923 – 2010)

2 CONCURRENT REACHABILITY GAME

Krishnendu Chatterjee and Henzinger [40] describe concurrent reachability game as a game that is played between two players (i.e., Player 0 and Player 1). The game has a starting state and in each round both players σ and $\overline{\sigma}$ choose a move independently and the parallel execution of the moves determines the successor state of the game. The concurrent reachability game has a reachability objective defined for each player. The reachability objective of the player σ is to force the game by selecting actions in such a way that its opponent $\overline{\sigma}$ actions can not prevent the game to reach a state which is among σ goal states set.

Consider a player σ for $\sigma \in \{0, 1\}$, then opponent is $\overline{\sigma}$ (i.e., $\overline{\sigma} = 1 - \sigma$).

Krishnendu Chatterjee and Henzinger [40] argue that the correctness properties (i.e., deadlock-free etc) of an open system can be modeled as reachability objectives for a two-player concurrent game. As described in the preceding chapter that a concurrent reachability game is unlike a Kripke structure [39] and provides a natural model for any open system because it maintains the differentiation of a design into system component and the environment [40].

Alfaro et al. states that one of the central problems in system verification is the reachability question. In an open system this reachability question is a phrase like, "*given an initial state and a goal state, can the system get from initial to the goal state?*" [3]. In concurrent reachability game, this reachability question compiles to a way of specifying objectives as winning of a game by any given player [3]. As described earlier that in a concurrent reachability game, the player σ tries to win the game by forcing the play started from some initial state to a set of goal states and the objective its opponent player $\overline{\sigma}$ is to prevent the player σ in achieving his goal. Thus, the reachability question is to determine "*can a player force a play in the game to a given set of goal states?*".

In this chapter, we modeled the two-player concurrent reachability games using reachability game structure. The reachability game structure has an area and a goal set. The area models an n-player concurrent game and its playability whereas the goal set helps to formulate the reachability question in a two-player game scenario.

2.1 CONCURRENT GAME

A `Concurrent Reachability Game` is formulated using a `Reachability Game Structure` which is composed of an `Arena` and a `Goal Set`. The arena captures the behavior of an n-player concurrent game and the goal set helps to define the reachability objective in a concurrent game.

Definition 1 (Arena)**.** The *arena* of a game is defined as follows

$$\mathcal{A} = \langle \mathcal{P}, Q, q_0, \gamma, \xi, \delta, \tau \rangle$$

where:

- $\mathcal{P}$ is a finite set of players. Let $n \in \mathbb{N}$ represent the number of players in a game (i.e., $n = |\mathcal{P}|$),
- Q is a finite set of states,
- $q_0 \in Q$ is the initial state,
- γ is a finite set of moves where a move represents an action for a player in the game,
- $\xi : \mathcal{P} \times Q \to 2^{\gamma}$ is a legal move function. $\xi(i, q)$ constitutes a nonempty set of legal moves for player $i \in \mathcal{P}$ at state q. Each player can independently select a move from its pool of legal moves (i.e., $\alpha_i \in \xi(i, q)$).
- $\delta : Q \times \gamma^n \to Q$ is a partial transition function where at state $q \in Q$, each player $i \in \mathcal{P}$ chooses simultaneously and independently an action $\alpha_i \in \xi(i, q)$; which form a tuple $\langle \alpha_1, \dots, \alpha_n \rangle \in \gamma^n$ as a joint action at state q and the game proceeds to the successor state $\delta(q, \langle \alpha_1, \dots, \alpha_n \rangle) \in Q$.
- $\tau \subseteq Q$ is a finite set of terminal states. For all states $q \in \tau$ and for all joint actions $\langle \alpha_1, \dots, \alpha_n \rangle \in \gamma^n$ where $\alpha_i \in \xi(i, q)$ and $i \in \mathcal{P}$, the successor state $\delta(q, \langle \alpha_1, \dots, \alpha_n \rangle) = \infty$.

A game is played according to its defined rules. The following section defines the playability in an arena which conform with the game rules.

2.1.1 *Playability*

A single play in an *arena* is a set of states and the executions of simultaneous actions of the players in that arena. A play could be a finite play means it may terminate at some state or an infinite one which may continue running without termination.

Definition 2 (Finite Play)**.** A finite play $\pi \in Q \times (\gamma^n \times Q)^*$ in $\mathcal{A}$ is a finite sequence $\pi = q_0 \xrightarrow{\langle \alpha_1^0, \dots, \alpha_n^0 \rangle} q_1 \xrightarrow{\langle \alpha_1^1, \dots, \alpha_n^1 \rangle} q_2 \dots \xrightarrow{\langle \alpha_1^{m-1}, \dots, \alpha_n^{m-1} \rangle} q_m$ of states starting from the initial state $q_0 \in Q$ such that $q_{k+1} = \delta(q_k, \langle \alpha_1^k, \dots, \alpha_n^k \rangle)$ for all $0 \leq k \leq m - 1$.

We denote Ω the set of all finite plays.

Definition 3 (Infinite Play)**.** An infinite play $\pi \in Q \times (\gamma^n \times Q)^\omega$ in $\mathcal{A}$ is an infinite sequence $\pi^\omega = q_0 \xrightarrow{\langle \alpha_1^0,\ldots,\alpha_n^0 \rangle} q_1 \xrightarrow{\langle \alpha_1^1,\ldots,\alpha_n^1 \rangle} q_2 \ldots$ of states starting from initial state $q_0 \in Q$ such that $q_{k+1} = \delta(q_k, \langle \alpha_1^k, \ldots, \alpha_n^k \rangle)$ for all $k \geq 0$.

We denote Ω_{inf} the set of all infinite plays. In the following, we define a play prefix for any finite or infinite play.

Definition 4 (Play Prefix)**.** The prefix of any given finite play $\pi = q_0 \xrightarrow{\langle \alpha_1^0,\ldots,\alpha_n^0 \rangle} q_1 \xrightarrow{\langle \alpha_1^1,\ldots,\alpha_n^1 \rangle} \ldots \xrightarrow{\langle \alpha_1^{m-1},\ldots,\alpha_n^{m-1} \rangle} q_m \in \Omega$ or an infinite play $\pi = q_0 \xrightarrow{\langle \alpha_1^0,\ldots,\alpha_n^0 \rangle} q_1 \xrightarrow{\langle \alpha_1^1,\ldots,\alpha_n^1 \rangle} \cdots \in \Omega_{inf}$ is defined as $\pi\langle l \rangle = q_0 \xrightarrow{\langle \alpha_1^0,\ldots,\alpha_n^0 \rangle} q_1 \xrightarrow{\langle \alpha_1^1,\ldots,\alpha_n^1 \rangle} \ldots \xrightarrow{\langle \alpha_1^{l-1},\ldots,\alpha_n^{l-1} \rangle} q_l$ where $0 \leq l \leq m$.

A play proceeds depending on the actions selected by the players. It selects a move from the set of its available moves at any given state using a strategy. The strategy is defined as follows.

2.1.2 *Player Strategy*

A strategy for a player is a procedure for selecting actions given the history of the play. During any play $\pi \in \Omega$ a strategy function determines the move for a player to be executed at any given state.

Definition 5 (Strategy)**.** A strategy for a player $i \in \mathcal{P}$ is a function $f_i : \Omega \to \gamma$ that associates each state q_k in a finite, play $\pi \in \Omega$ an action $\alpha_i^k \in f_i(\pi\langle k \rangle)$ such that $\alpha_i^k \in \xi(i, q_k)$ where $0 \leq k \leq |\pi|$.

We denote, ρ_i be set of all strategies for a player $i \in \mathcal{P}$.

In a game, a play proceeds depending on the strategies chosen by the players and defined as follows.

Definition 6 (Outcome)**.** $\chi : Q \times \rho_1 \times \cdots \times \rho_n \to \Omega \cup \Omega_{\text{inf}}$ is called an outcome. For any play $\pi = q_0 \xrightarrow{\langle \alpha_1^0,\ldots,\alpha_n^0 \rangle} q_1 \xrightarrow{\langle \alpha_1^1,\ldots,\alpha_n^1 \rangle} q_2 \cdots \in \Omega \cup \Omega_{inf}$:
$\pi = \chi(q_0, f_1, \ldots, f_n)$ iff

- $\alpha_i^k = f_i(\pi\langle k \rangle)$ for all $0 \leq k \leq |\pi|$ where $1 \leq i \leq n$
- $q_{k+1} \in \delta(q_k, \langle \alpha_1^k, \ldots, \alpha_n^k \rangle)$.

In this report, we restrict ourselves to two-player concurrent reachability games in which an *arena* has at most two players (i.e., $1 \leq |\mathcal{P}| \leq 2$). Consider, a player $\sigma \in \mathcal{P}$ then the opponent is $\overline{\sigma}$ (i.e., $\overline{\sigma} = 1 - \sigma$). In a two player game, given a state $q \in Q$ and two strategies f_σ and $f_{\overline{\sigma}}$ for the players σ and its opponent $\overline{\sigma}$, the outcome $\chi(q, f_\sigma, f_{\overline{\sigma}}) \in \Omega \cup \Omega_{\text{inf}}$ is a play that can be followed when the game starts from state q and both players use the strategies f_σ and $f_{\overline{\sigma}}$ respectively. The outcome is always unique.

In the next section, we define reachability objectives for the players in a concurrent game in such a way that a player has a reachability objective and the objective of his opponents is the dual (an invariant).

2.2 REACHABILITY GAME

It is described earlier that a concurrent reachability game is just a specialization of an n-player concurrent game. We define `Reachability Game Structure` which models concurrent reachability games in such manner that the player σ has a reachability objective and the objective of its opponent player $\overline{\sigma}$ is dual to it.

Definition 7 (Reachability Game Structure)**.** The *two-player reachability game structure* is a pair

$$\mathcal{G} = (\mathcal{A}, \mathcal{R}_\sigma)$$

where

- $\mathcal{A} = \langle \mathcal{P}, Q, q_0, \gamma, \xi, \delta, \tau \rangle$ is the arena where $\mathcal{P} = \{0, 1\}$,
- $\mathcal{R}_\sigma \subseteq \tau$ is the goal set for player σ (i.e., the goal of player σ is to reach any state in $\mathcal{R}_\sigma$) whereas the player $\bar{\sigma}$ tries to prevent the player σ by forcing the game to remain in its goal set $\mathcal{R}_{\overline{\sigma}} := Q \setminus \mathcal{R}_\sigma$.

For player σ, the reachability condition is specified as a way to reach any state in a goal set $\mathcal{R}_\sigma$. The objective of the opponent $\overline{\sigma}$ is to prevent σ by forcing the play within its goal set $\mathcal{R}_{\overline{\sigma}}$. The reachability condition is specified by defining reachability objectives as a set of winning plays for the players in the game which is done in the following section.

2.2.1 *Winning Objective*

The reachability game structure models concurrent reachability game in such a way that the question of reachability is reduced to determine whether any given player in a game can force a play to its goal set. Alu et al. [5] formulates the answer to the reachability question is to find whether a player σ has a strategy so that for all strategies of player $\overline{\sigma}$, the game, if started from an initial state q_0 it always reaches to a state q such that $q \in \mathcal{R}_\sigma$. The task is to find an effective construction of a winning strategy for player σ against all spoiling strategies of opponent player $\overline{\sigma}$. For it first we define, what constitutes a winning play in a concurrent reachability game.

Definition 8 (Winning Play)**.** The set of winning plays $\mathcal{W}_\sigma$ for a player σ is defined as

$$\mathcal{W}_\sigma = \{\pi = q_0 \xrightarrow{\langle \alpha^0_\sigma, \alpha^0_{\overline{\sigma}} \rangle} q_1 \xrightarrow{\langle \alpha^1_\sigma, \alpha^1_{\overline{\sigma}} \rangle} \cdots \in \Omega \cup \Omega_{inf} : q_k \in \mathcal{R}_\sigma \text{ for } 0 \le k \le |\pi|\}$$

The player σ has a winning strategy to reach its goal set and the objective of his opponent $\overline{\sigma}$ is to act as a spoiler to prevent player σ to achieve its goal. During any finite or infinite play the player σ tries to force the play to a set of states called the winning region states for player σ. The winning region $\mathcal{WR}_\sigma \subseteq Q$ consist of a set of states from which the player σ has a strategy to force the game to $\mathcal{R}_\sigma$ and is defined as follows.

Definition 9 (Winning Region)**.** Any state $q \in Q$ is in winning region for player σ, $q \in \mathcal{WR}_\sigma$ iff there exist $f_\sigma \in \rho_\sigma$ against all opponent strategies $f_{\overline{\sigma}} \in \rho_{\overline{\sigma}}$ such that $\chi(q, f_\sigma, f_{\overline{\sigma}}) \in \mathcal{W}_\sigma$.

As described earlier that in concurrent reachability game the objectives of the players are strictly competitive. The player σ is playing for winning and then the player $\overline{\sigma}$ counteract as a spoiler. The player σ tries to adopt a strategy which let it win a game (i.e., forcing the play to reach a state which is in its winning region). The winning strategy is a strategy which helps the player σ to achieve his goal. A winnings strategy is defined w.r.t to the winning region. It is a strategy f_σ for player σ that act as a witness to states in $\mathcal{WR}_\sigma$. The winning strategy is formally defined as follows.

Definition 10 (Winning Strategy)**.** A winning strategy f_σ for player σ is a witness to a state $q \in \mathcal{WR}_\sigma$ such that $\chi(q, f_\sigma, f_{\overline{\sigma}}) \in \mathcal{W}_\sigma$ against all strategies $f_{\overline{\sigma}} \in \rho_{\overline{\sigma}}$ of player $\overline{\sigma}$.

The spoiler goal is to exhibit a strategy which prevent the winner to achieving its goal (i.e., forcing the play to reach remain within its goal set). A spoiling strategy w.r.t. to the winning region is a strategy $f_{\overline{\sigma}}$ for player $\overline{\sigma}$ that act as a witness to all states $q \notin \mathcal{WR}_\sigma$. This spoiling strategy is defined formally as follows.

Definition 11 (Spoiling Strategy)**.** A spoiling strategy $f_{\overline{\sigma}}$ for a player $\overline{\sigma}$ is a witness to a state $q \notin \mathcal{WR}_\sigma$ such that $\chi(q, f_\sigma, f_{\overline{\sigma}}) \notin \mathcal{W}_\sigma$ against all strategies $f_\sigma \in \rho_\sigma$ of player σ.

Two players in a counteract each other with winning and spoiling strategies. A player σ has a winning strategy in $\mathcal{G}$ if and only if the player σ can force the outcome of the play within its winning region $\mathcal{WR}_\sigma$.

Proposition 12. In a concurrent reachability $\mathcal{G}$, the player σ using a winning strategy $f_\sigma \in \rho_\sigma$ against all opponent strategies $f_{\overline{\sigma}} \in \rho_{\overline{\sigma}}$ can force any play $\pi \in \Omega \cup \Omega_{inf}$ such that $\pi\langle k\rangle \in \chi(q_k, f_\sigma, f_{\overline{\sigma}})$ for some $q_k \in \mathcal{WR}_\sigma$ where $0 \leq k \leq |\pi|$.

In the following, we illustrate a concurrent game example.

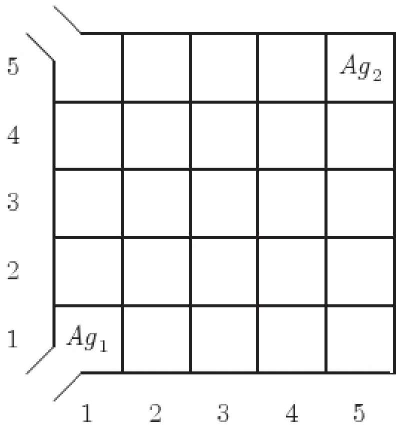

Figure 1: Guard and Intruder Arena[58]

EXAMPLE: GUARD AND INTRUDER: As a running example in this chapter, we shall consider an area of a game depicted in Figure 1. In this game, a guard which is represented as player Ag_1 tries to catch an intruder represented as player Ag_2. The player Ag_2 goal is to escape via one of the two exit locations $(1, 1)$ and $(1, 5)$. Both players act synchronously and can move horizontally or vertically to an adjacent position. Ag_2 is caught when it ends up in the same location as Ag_1.

This example game can be formally described using reachability game structure. However, even though it is obviously just a toy-example and a subsidiary to a large example presented in [58].For illustration, consider the following formalization of this concurrent reachability game environment depicted in Figure 1.
Let $\mathcal{G}$ be a reachability game structure

$$\mathcal{G} = (\langle \mathcal{P}, Q, q_0, \gamma, \xi, \delta, \tau \rangle, \mathcal{R}_{Ag_2})$$

where

- $\mathcal{P} = \{Ag_1, Ag_2\}$;
- $Q \subseteq l_{Ag_1} \times l_{Ag_2}$ where $l_i = \{(x, y) \mid x, y \in \{1, \ldots, 5\}\} \cup \{\textit{escaped}\}$ for player $i \in \mathcal{P}$;
- $q_0 = \{((1, 1), (5, 5))\}$;
- $\gamma = \{\textit{north}, \textit{south}, \textit{east}, \textit{west}, \textit{stay}, \textit{exit}\}$;
- The legal moves are defined for each player $i \in \mathcal{P}$ as follows:
  - *stay* $\in \xi(i, q)$ is a legal action for all players $i \in \mathcal{P}$ at any state $q \in Q$.
  - *north* $\in \xi(i, q)$ is a legal action for each player $i \in \mathcal{P}$ at any given state $q = (l_{Ag_1}, l_{Ag_2})$ where $l_{Ag_1} = (x_1, y_1)$ and $l_{Ag_2} = (x_2, y_2)$ such that $y_1, y_2 < 5$.

- *south* $\in \xi(i, q)$ is a legal action for each player $i \in \mathcal{P}$ at any given state $q = (l_{Ag_1}, l_{Ag_2})$ where $l_{Ag_1} = (x_1, y_1)$ and $l_{Ag_2} = (x_2, y_2)$ such that $y_1, y_2 > 0$.
- *east* $\in \xi(i, q)$ is a legal action for each player $i \in \mathcal{P}$ at any given state $q = (l_{Ag_1}, l_{Ag_2})$ where $l_{Ag_1} = (x_1, y_1)$ and $l_{Ag_2} = (x_2, y_2)$ such that $x_1, x_2 < 5$.
- *west* $\in \xi(i, q)$ is a legal action for each player $i \in \mathcal{P}$ at any given state $q = (l_{Ag_1}, l_{Ag_2})$ where $l_{Ag_1} = (x_1, y_1)$ and $l_{Ag_2} = (x_2, y_2)$ such that $x_1, x_2 > 0$.
- *exit* $\in \xi(Ag_2, q)$ if
  - $q = \{((1, 1), (x, y))$ where $x \neq 1, y \neq 1\}$ or
  - $q = \{((5, 1), (x, y))$ where $x \neq 5, y \neq 1\}\}$

• $\delta : Q \times \gamma^2 \to Q$ is the transition function. The effects of any action $\alpha_i \in \xi(i, q)$ executed by a player $i \in \{Ag_1, Ag_2\}$ in a transition $\delta(q, \langle \alpha_{Ag_1}, \alpha_{Ag_2} \rangle)$ at state $q = (l_1, l_2)$ where $l_j = (x_j, y_j)$, $j \in \{1, 2\}$ then updated locations are defined accordingly with respect to every action taken by i:
  - If $\alpha_i = $ *north* then for any $y_j = y_j + 1$ and x_j remains unchanged.
  - If $\alpha_i = $ *south* then for any $y_j = y_j - 1$ and x_j remains unchanged .
  - If $\alpha_i = $ *east* then for any $x_j = x_j + 1$ and y_j remains unchanged.
  - If $\alpha_i = $ *west* then for any $x_j = x_j - 1$ and y_j remains unchanged.
  - If $\alpha_i = $ *stay* then both x_j, y_j remain unchanged.

• The set of terminal states $\tau = \tau_1 \cup \tau_2$ contains all the following states:
  - $\tau_1 = \{(\mathsf{escaped}, (x, y))$ where $x, y \in \{1, \dots, 5\}\}$ (i.e., intruder Ag_2 has escaped),
  - $\tau_2 = \{((x_1, y_1), (x_2, y_2))$ for all $x_1 = x_2 \wedge y_1 = y_2\}$ (i.e., if guard Ag_1 has caught the intruder Ag_2.).

• The *Goal Set* $\mathcal{R}_{Ag_2}$ contains all the states in which the intruder Ag_2 has escaped: $\mathcal{R}_{Ag_2} = \{(\mathsf{escaped}, (x, y))$ where $1 \leq x, y \leq 5\}$.

• The dual *Goal Set* is $\mathcal{R}_{Ag_1} := \tau_2 \cup Q \setminus \mathcal{R}_{Ag_2}$ contains all the states in which the intruder Ag_1 is either caught or not able to escape the grid.

The reachability objective for the intruder Ag_2 is to find a winning strategy f_{Ag_2} which let him escape without being caught. The dual objective of the guard Ag_1 is to find a spoiling strategy f_{Ag_1} which does not let the intruder Ag_2 escape.

When intuition and logic agree, you are always right.

— Blaise Pascal (1623 – 1662)

3 GAME DESCRIPTION LANGUAGE (GDL)

Love et al. [45] had introduced Game Description Language (GDL) as a declarative language to define games because logic-based rules provide a comprehensive encoding of a game. Love et al. describe Game Description Language (GDL) as a variant of Datalog [2] but do not provide a mathematically precise syntax and semantics of the language. Love et al. emphasis is to explain the idea and the framework of General Game Playing (GGP) [27] rather than defining the syntax and semantics for GDL.

General Game Playing (GGP) is the design and development of intelligent programs that enable to play any arbitrary game specified in GDL without any human intervention [45].

In order to use logic programming to solve the reachability question in concurrent reachability games, we specify the rules of concurrent games in Game Description Language (GDL) [45]. Thielscher [62] states that a GDL program can be understood as a description of a formal game model in form of a state transition system. Van et al., van der Hoek et al. [64, 65] and Thielscher [62] define the transition-based semantics of a GDL program in term of a state transition system. It suggested in [64, 65, 62] that defining the transition-based semantics for GDL programs makes the informal description of Game Description Language (GDL) presented in [45] mathematically precise as it is more intuitive to understand a game as a transition system rather than as a set of logical rules. We had used reachability game structure to specify the transition-based semantics for GDL. Thus, Game Description Language (GDL) provides a comprehensive encoding of a concurrent reachability game and reachability game structure is used to provide a game-based transition model for GDL programs. This approach is inspired from a similar work of Schiffel and Thielscher [57] in which a multiagent environment is specified in GDL which in turn is used to provide the formal semantics for a GDL program in terms of a transition system. Although, it is inefficient to perform automated reasoning on the GDL programs described as transition systems but it is more convenient in proving mathematical properties over GDL programs.

In this chapter, We define a declarative syntax, logic-based semantics and transition-based semantics for Game Description Language (GDL). The logic-based semantics provides a precise meaning to GDL programs and facilitate an efficient reasoning over game rules whereas transition-based semantics are more intuitive.

3.1 GDL SYNTAX

Game Description Language (GDL) is based on the standard syntax of logic programming where a program consists of a set of logical rules.

Definition 13 (Vocabulary)**.** The GDL vocabulary VOC is a set of function symbols and predicate symbols

$$\mathsf{VOC} = (\mathit{Func}, \mathit{Pred})$$

- $\mathit{Func} = \bigcup_{l \geq 0} \mathsf{Func}_l$ is a countable set of l-ary function symbols,
- $\mathit{Pred} = \bigcup_{m \geq 0} \mathsf{Pred}_m$ is a countable set of m-ary predicate symbols.
- $\{\mathsf{init}, \mathsf{true}, \mathsf{next}\} \subseteq \mathit{Pred}$ are fluent symbols.
- $\{\mathsf{terminal}, \mathsf{role}, \mathsf{legal}, \mathsf{does}, \mathsf{goal}, \mathsf{distinct}\} \subseteq \mathit{Pred}$ are auxiliary symbols.

A fluent is a property that can change over time [55].

We also have a set Var which contains a countable supply of variables symbols.

As a tailor-made specification language, GDL uses predicate symbols $\{\mathsf{init}, \mathsf{true}, \mathsf{next}\}$ and $\{\mathsf{terminal}, \mathsf{role}, \mathsf{legal}, \mathsf{does}, \mathsf{goal}, \mathsf{distinct}\}$ as keywords which are shown in Table 1 together with their informal meaning as described in [45]. These fluents and auxiliary symbols are formally represented by the transition-based semantics of GDL in the section 3.4.

The most basic construct in a declarative language is a term. We define GDL terms in the following way.

Definition 14. (GDL Term ($\mathsf{T}_{\mathsf{GDL}}$)) The set of GDL terms is inductively defined as follows

- Each variable $x \in \mathsf{Var}$ is a GDL term,
- If $f \in \mathsf{Func}_m$ is m-ary function symbol and $t_1, \ldots, t_m \in \mathsf{T}_{\mathsf{GDL}}$ are terms then $f(t_1, \ldots, t_m)$ is a GDL term,
- Each constant symbol $c \in \mathsf{Func}_0$ is a GDL term,

So, the GDL terms in a BNF-like notation are:

$$t ::= x \mid f(t_1, \ldots, t_m) \mid c$$

where $c \in \mathsf{Func}_0$, $f \in \mathsf{Func}_m$ for some $m \geq 1$ and $x \in \mathit{Var}$.
We use these GDL terms to define GDL atoms.

Definition 15. (Normal Atom ($\mathcal{A}_{\mathsf{NR}}$)) A `normal atom` is $p(t_1, \ldots, t_n) \in \mathcal{A}_{\mathsf{NR}}$ where $p \in \mathsf{Pred}_n$ be an n-ary predicate symbol, $p \notin \{\mathsf{init}, \mathsf{true}, \mathsf{next}\}$ and $t_1, \ldots, t_n \in \mathsf{T}_{\mathsf{GDL}}$.

The fluent atom has a unit arity and defined as follows.

$\mathsf{terminal}$	The current position is terminal.
$\mathsf{role}(t)$	t is a player.
$\mathsf{legal}(t_1, t_2)$	Player t_1 has a legal move t_2.
$\mathsf{does}(t_1, t_2)$	Player t_1 performs an action t_2.
$\mathsf{goal}(t_1, t_2)$	Player t_1 gets goal value t_2 in the current position.
$\mathsf{init}(t)$	The fluent t holds in the initial position.
$\mathsf{true}(t)$	The fluent t holds in the current position.
$\mathsf{next}(t)$	The fluent t holds in the next position.

Table 1: GDL fluents and auxiliary symbols [58].

Definition 16. (Fluent Atom ($\mathcal{A}_{\mathsf{FL}}$)) A `fluent atom` is $p(t) \in \mathcal{A}_{\mathsf{FL}}$ where $p \in \{\mathsf{init}, \mathsf{true}, \mathsf{next}\}$ and $t \in T_{\mathsf{GDL}}$.

The auxiliary symbols form the predefined atoms and defined as follows.

Definition 17. (Predefined Atom ($\mathcal{A}_{\mathsf{PD}}$)) The `predefined atoms` are the following set of GDL atoms:

- $\mathsf{terminal} \in \mathcal{A}_{\mathsf{PD}}$ where $\mathsf{terminal} \in \mathsf{Pred}_0$,
- $\mathsf{role}(t) \in \mathcal{A}_{\mathsf{PD}}$ where *role* $\in \mathsf{Pred}_1$ and term $t \in T_{\mathsf{GDL}}$,
- $\mathsf{legal}(t_1, t_2) \in \mathcal{A}_{\mathsf{PD}}$ where *legal* $\in \mathsf{Pred}_2$ and terms $t_1, t_2 \in T_{\mathsf{GDL}}$,
- $\mathsf{does}(t_1, t_2) \in \mathcal{A}_{\mathsf{PD}}$ where *does* $\in \mathsf{Pred}_2$ and terms $t_1, t_2 \in T_{\mathsf{GDL}}$,
- $\mathsf{goal}(t_1, t_2) \in \mathcal{A}_{\mathsf{PD}}$ where *goal* $\in \mathsf{Pred}_2$ and terms $t_1, t_2 \in T_{\mathsf{GDL}}$,
- $\mathsf{distinct}(t_1, t_2) \in \mathcal{A}_{\mathsf{PD}}$ where $\mathsf{distinct} \in \mathsf{Pred}_2$ and $t_1, t_2 \in T_{\mathsf{GDL}}$ are two arbitrary distinct terms.

A GDL atom is defined to be either a *normal atom* or *fluent atom* or *predefined atom*. It is formally defined as follows.

Definition 18. (GDL Atom ($\mathcal{A}_{\mathsf{GDL}}$)) The set of GDL atoms are:

$$\mathcal{A}_{\mathsf{GDL}} \overset{\text{def}}{:=} \mathcal{A}_{\mathsf{NR}} \cup \mathcal{A}_{\mathsf{FL}} \cup \mathcal{A}_{\mathsf{PD}}$$

So, a GDL atom in a BNF-like notation is

$$\phi ::= \mathsf{terminal} \mid \mathsf{role}(t) \mid \mathsf{init}(t) \mid \mathsf{true}(t) \mid \mathsf{next}(t) \mid p(t_1, \ldots, t_m) \mid \\ \mathsf{legal}(t_1, t_2) \mid \mathsf{does}(t_1, t_2) \mid \mathsf{goal}(t_1, t_2) \mid \mathsf{distinct}(t_1, t_2)$$

$\mathcal{A}_{\mathsf{GDL}}$ is a set of atoms in GDL. A `literal` is an atom or a negation of an atom.

Definition 19. (GDL Literal ($\mathcal{E}_{\mathsf{GDL}}$)) The set of literals $\mathcal{E}_{\mathsf{GDL}}$ is defined as, if $\phi \in \mathcal{A}_{\mathsf{GDL}}$ then $\phi, \neg\, \phi \in \mathcal{E}_{\mathsf{GDL}}$.

A GDL rule is defined as follows.

Definition 20. (GDL Rule) A GDL rule r is

$$h \Leftarrow e_1 \wedge \ldots \wedge e_m$$

where $h \in \mathcal{A}_{GDL}$, $e_1, \ldots, e_m \in \mathcal{E}_{GDL}$. If $m = 0$, we say that rule has an *empty body* and is called a `GDL fact`.

The head of any given GDL rule r is obtained using function $H(r) = h$ and body of the rule is a set of literals $B(r) = \{e_1, \ldots, e_m\}$, $B(r) = B^+(r) \cup B^-(r)$ where $B^+(r)$ is positive set of literals and $B^-(r)$ is negative set of literals.
A rule is called `ground` if no variable occurs in it. A rule is *safe* if and only if every variable in the rule appears in some positive sub-goal in the body [8]. If a GDL program contains only safe rules then it guarantees variable-free facts.

Definition 21. (GDL Program (Γ_{GDL})) A GDL program Γ_{GDL} is a finite set of GDL rules with the following constraints. For any rule $r \in \Gamma_{GDL}$

- $role(t) \notin B(r)$.
- $init(t) \notin B(r)$.
- $true(s) \notin H(r)$.
- $next(u) \notin B(r)$.
- If $does(t_1, t_2) \in B(r)$ then $legal(t_1, t_2) \notin B(r)$, $goal(t_1, t_2) \notin B(r)$ and $terminal \notin B(r)$.
- r is a safe rule.

Definition 22 (Safe)**.** A rule r is called *safe* if every variable x occurring in $H(r) \cup B^-(r)$ also occurs in at least one literal $B^+(r)$ in the same rule. A GDL program Γ_{GDL} is safe if all of its rules are safe.

We can categorized the GDL rules into seven different types and explain the informal meaning of each category. This helps us to better understand the game descriptions specified in GDL.

Definition 23. A GDL program $\Gamma_{GDL} := \Upsilon_{role} \cup \Upsilon_{init} \cup \Upsilon_{glob} \cup \Upsilon_{next} \cup \Upsilon_{legal} \cup \Upsilon_{goal} \cup \Upsilon_{terminal}$ is partitioned into seven different types:

- Υ_{role} contains all the facts of form $role(v) \Leftarrow$. It is to specify the player(s) in the game.
- Υ_{init} is a set of facts of the form $init(t) \Leftarrow$ and represent constraints about the initial state of the game.
- Υ_{glob} is a set of global rules of form $h \Leftarrow e_1 \wedge \ldots \wedge e_m$ where $h \in \mathcal{A}_{GDL}$ and each $e_i \in \mathcal{E}_{GDL}$ for $1 \leq i \leq m$.

- Υ_{next} contains all the rules of form $next(t) \Leftarrow e_1 \wedge \ldots \wedge e_m$ where each $e_i \in \mathcal{E}_{GDL}$ for $1 \leq i \leq m$. It represents the transition function of the game.
- Υ_{legal} contains all the rules of form $legal(t_1, t_2) \Leftarrow e_1 \wedge \ldots \wedge e_m$ where each $e_i \in \mathcal{E}_{GDL}$ for $1 \leq i \leq m$. These rules constraint the set of legal moves available for each player in the game.
- Υ_{goal} contains all the rules of form $goal(t_1, t_2) \Leftarrow e_1 \wedge \ldots \wedge e_m$ where each $e_i \in \mathcal{E}_{GDL}$ for $1 \leq i \leq m$. It is to specify the goal-hood.
- $\Upsilon_{terminal}$ contains all the rules of form $terminal \Leftarrow e_1 \wedge \ldots \wedge e_m$ where each $e_i \in \{1 \ldots m\}$ for $1 \leq i \leq m$. It represents the terminal conditions in the game.

A GDL program which does not contain negation is called *definite GDL program* and rules in the program are called *definite GDL rules*. In other words, in a definite GDL program only positive literals occur in the body of a rule (i.e., $\forall r \in \Gamma_{GDL}\ B^-(r) = \emptyset$) otherwise, the GDL program is called normal GDL program (i.e., $\exists r \in \Gamma_{GDL}$: $B^-(r) \neq \emptyset$).

In the following section, we provide the logic-based semantics of GDL and then continue in the next section by presenting the transition-based semantic for GDL programs.

3.2 GDL SEMANTICS

We had provided both the logic-based semantics and transition-based semantics for Game Description Language (GDL). As described earlier that logic-based semantics are necessary because it provides a precise meaning to the GDL programs and facilitate an efficient inference mechanism to reason on game rules specified in GDL. We also define the transition-based semantics for GDL because it is more intuitive to understand a games as a transition system rather than as a set of logical rules. Although, logic-based rules provides a comprehensive encoding of any given game description but are difficult to comprehend. On the other hand, GDL transition-based semantics has an inefficient computational machinery but more intuitive in proving mathematical properties of GDL programs. In the following section, we define the declarative semantics for definite GDL program as well as for normal GDL programs. In next section, we explain the transition-based semantics for GDL programs.

3.3 GDL LOGIC-BASED SEMANTICS

Przymusinski [50] states that the most commonly used declarative semantics of logic programs is based on Clark's completion [16]. Clark's completion is a rewriting of the rules of a logic program which is formally explained in Section 3.3.1. The Clark's completion is not suitable for recursive-free programs because it fails to provide correct semantics for the programs which have recursions in the rules. The recursion in logic programs can occur in two ways namely `positive recursion` and `negative recursion`. These recursions are defined as follows.

Definition 24 (Positive Recursion)**.** A GDL program contains `positive recursion` if any atom $\phi \in \mathcal{A}_{\mathrm{GDL}}$ in the program depends on a positive occurrence of itself (i.e., a rule of form $\phi \Leftarrow \phi \wedge \ldots \wedge \psi$).

Definition 25 (Negative Recursion)**.** A GDL program contains `negative recursion` if any atom $\psi \in \mathcal{A}_{\mathrm{GDL}}$ in the program depends on a negative occurrence of itself (i.e., a rule of form $\psi \Leftarrow \neg\psi \wedge \ldots \wedge \phi$).

Clark's completion is not sufficiently expressive to deal with the transitive closure [42] in case where positive recursion occurs in a program. We explain it using the following example.

Example 26. [50] Consider the following definite GDL program

$$\begin{aligned}\Gamma_{\mathrm{GDL}} = \{&\mathrm{edge}(a, b) \Leftarrow\\ &\mathrm{edge}(c, d) \Leftarrow\\ &\mathrm{edge}(d, c) \Leftarrow\\ &\mathrm{reachable}(a) \Leftarrow\\ &\mathrm{reachable}(X) \Leftarrow \mathrm{reachable}(Y), \mathrm{edge}(X, Y)\}.\end{aligned}$$

In this example a graph is encoded as a Γ_{GDL} program. In this graph, the vertices c and d are not reachable from the vertex a. This fact should be derivable from Γ_{GDL} program. But using Clark's completion such a conclusion cannot be derived. The difficulty is caused because there exist a symmetry between $\mathrm{edge}(c, d)$ and $\mathrm{edge}(d, c)$. Removal of $\mathrm{edge}(c, d)$ or $\mathrm{edge}(d, c)$ eliminates the problem.

Przymusinski [50] also mentioned that Clark's completion semantics fails to provide a suitable semantics for logic programs contains negative recursion because of inconsistency. We explain it using the following example.

Example 27. [53] Consider the following definite logic program

$$\begin{aligned}\Gamma = \{&\mathrm{work} \Leftarrow \neg\mathrm{tired}\\ &\mathrm{sleep} \Leftarrow \neg\mathrm{work}\\ &\mathrm{tired} \Leftarrow \neg\mathrm{sleep}\\ &\mathrm{angry} \Leftarrow \neg\mathrm{paid} \wedge \mathrm{work}\\ &\mathrm{pain} \Leftarrow\}.\end{aligned}$$

The Clark's completion of the above program is:

$$\begin{aligned}\mathrm{comp}(\Gamma) = \{&\mathrm{work} \Leftrightarrow \neg\mathrm{tired}\\ &\mathrm{sleep} \Leftrightarrow \neg\mathrm{work}\\ &\mathrm{tired} \Leftrightarrow \neg\mathrm{sleep}\\ &\mathrm{angry} \Leftrightarrow \neg\mathrm{paid} \wedge \mathrm{work}\\ &\mathrm{paid} \Leftrightarrow \mathrm{true}\}.\end{aligned}$$

which is inconsistent.

As it is illustrated in [50], that Clark's completion is only consistent for programs which do not involve negative recursion. In the following section, we explain Clark's completion in detail and then define the least models semantics for definite GDL programs which eliminates the positive recursion problem of Clark's completion semantics. The definite GDL programs do not contain negative rules therefore the negative recursion problem never occurs. The least model semantics is well defined for the class of definite GDL programs but these semantic are not applicable to the normal GDL programs. Therefore, after presenting in the least models semantics, we define a syntactically restricted class of normal logic programs known as stratified GDL programs and provide perfect model semantics for these stratified GDL programs. It is stated in [45, 62] that all GDL programs should be stratified because it ensure finite driveability. The introduction of stratified logic programs and perfect model semantics constituted a major breakthrough because it cover the complete class of GDL programs which also contain positive and negative recursion and GDL programs provide the logic-based semantics to more expressive class of GDL programs.

3.3.1 *Clark's Completion Semantics*

Clark's completion [16] is specified syntactically as rules transformation. The idea is to replace the implications with equivalences in a logic program. We denote completion for any given GDL program Γ_{GDL} by $\mathtt{comp}(\Gamma_{\mathsf{GDL}})$ and it is defined as follows.

Definition 28 (Completion)**.** The completion of a GDL program Γ_{GDL} is:

$$\mathtt{comp}(\Gamma_{\mathsf{GDL}}) = \{\mathit{def}(p) : p \in \mathit{Pred}\}$$

1. For each rule r of form $p(t_1, \ldots, t_m) \Leftarrow e_1 \wedge \ldots \wedge e_m$ where p is a predicate symbol of arity m in $H(r)$, we choose pairwise distinct, fresh variables $x_1, \ldots, x_m$. Thus, each rule of form r in which the head of rule is p, we have

$$p(x_1, \ldots, x_m) \Leftarrow x_1 = t_1 \wedge \cdots \wedge x_m = t_n \wedge e_1 \wedge \cdots \wedge e_m$$

2. Let $\mathit{norm}(r)$ is the above rule then for each rule in $r \in \Gamma_{\mathsf{GDL}}$ where the predicate symbol is p, $\mathtt{comp}(\Gamma_{\mathsf{GDL}})$ contains a rule $\mathit{def}(p)$ of the following form

$$p(x_1, \ldots, x_m) \Leftrightarrow \bigvee_{r \in \mathit{Rules}(p)} \mathit{norm}(r)$$

   where $\mathit{Rules}(p)$ denotes the set of rules $r \in \Gamma_{\mathsf{GDL}}$ where the head predicate is p.

Clark's semantics of a GDL program Γ_{GDL} is then defined as the set of all sentences logically implied by $\mathtt{comp}(\Gamma_{\mathsf{GDL}})$ (i.e., as the set of all sentences satisfied in all models of $\mathtt{comp}(\Gamma_{\mathsf{GDL}})$). Clark's completion programs also allows to infer negative consequences [50]. As mentioned earlier that the inadequate behavior of Clark's completion semantics for recursive programs depend whether the recursion is negative or positive. Przymusinski illustrated that the positive recursion in a logic program often leads to a completion which is too weak to express the intended meaning of the program (i.e., a completion from which some intuitively obvious conclusions can no longer be derived). Clark's completion semantics fails to provide a suitable semantics for logic programs contains positive recursion because of insufficient expressibility [60, 52]. We had explained it earlier using Example 40. Van Emden and Kowalski [66] define the least models semantics for definite logic programs which eliminates the positive recursion problem of Clark's completion semantics.

In the following section, we define least models semantics for definite GDL programs. The least model semantics is well defined for the class of definite GDL programs.

3.3.2 *Least Model Semantics*

Gabbay et al. [23] suggest that it is convenient and sufficient to focus on the Herbrand domain of a definite logic program to capture its semantics. For grounding, we need Herbrand Universe and Herbrand Base for GDL programs which are defined int he following.

Definition 29 (Herbrand Universe)**.** Let Γ_{GDL} be a GDL program then Herbrand Universe of Γ_{GDL}, denoted $HU_{\Gamma_{\mathsf{GDL}}}$, is the set of all ground (i.e., variable-free) terms that are recursively constructed by letting the arguments of the functions be constants in Γ_{GDL} or ground terms in $HU_{\Gamma_{\mathsf{GDL}}}$. If there is no constant symbol in the program, we add an arbitrarily symbol.

The terms, atoms, rules and programs are *ground* if they do not contain any variable.

Definition 30 (*ground*)**.** Let $\mathtt{r} \in \Gamma_{\mathsf{GDL}}$ be a GDL rule then $ground(\mathtt{r})$ denotes the set of ground instances of $\mathtt{r}$ obtained by assigning to the variables in $\mathtt{r}$ values from the Herbrand Universe $HU_{\Gamma_{\mathsf{GDL}}}$ 29.

The ground version of a GDL program Γ_{GDL}, denoted $ground(\Gamma_{\mathsf{GDL}})$, is the set of the ground instances of its rules defined as

$$ground(\Gamma_{\mathsf{GDL}}) = \{ground(\mathtt{r}) : \mathtt{r} \in \Gamma_{\mathsf{GDL}}\}$$

Definition 31 (Herbrand Base)**.** Let Γ_{GDL} be a GDL program then the Herbrand Base of Γ_{GDL}, denoted $HB_{\Gamma_{\mathsf{GDL}}}$, is the set of atoms that are constructed from the predicate symbols appearing in Γ_{GDL} to the terms appearing in $HU_{\Gamma_{\mathsf{GDL}}}$.

We consider atoms in the Herbrand Base and ground rules whose variables have been instantiated to elements of the Herbrand universe. Thus, the *ground instantiation* eliminates the variables from rules and we are left with only ground instantiated programs for which the formal interpretations is defined below.

Definition 32 (Herbrand Interpretation)**.** Let $\mathsf{VOC} = \{\mathit{Func}, \mathit{Pred}\}$ be a vocabulary and Var be a set of variables occurring in a GDL program Γ_{GDL}. A Herbrand interpretation of Γ_{GDL} is a subset of atoms $\mathcal{I} \subseteq \mathit{HB}_{\Gamma_{\mathsf{GDL}}}$ which is interpreted over terms as

- $x^{\mathcal{I}} \overset{\text{def}}{=} \mathcal{V}(x)$ for each variable $x \in \mathsf{Var}$, where $\mathcal{V} : \mathsf{Var} \to \mathit{HU}_{\Gamma_{\mathsf{GDL}}}$ is the variable valuation,
- For each m-ary function symbol $f \in \mathsf{Func}_m$

  $$f(t_1, \dots, t_m)^{\mathcal{I}} \overset{\text{def}}{=} f^{\mathcal{A}}(t_1^{\mathcal{I}}, \dots, t_m^{\mathcal{I}})$$

  where $f^{\mathcal{A}} : \mathit{HU}^m_{\Gamma_{\mathsf{GDL}}} \to \mathit{HU}_{\Gamma_{\mathsf{GDL}}}$ and $t_1^{\mathcal{I}}, \dots, t_m^{\mathcal{I}} \in \mathit{HU}_{\Gamma_{\mathsf{GDL}}}$. For $m = 0$, we get $c^{\mathcal{I}} = c^{\mathcal{A}}$ where each constant $c^{\mathcal{A}} \in \mathit{HU}_{\Gamma_{\mathsf{GDL}}}$.

The satisfiability relation $\models$ for GDL programs is defined as follows.

Definition 33 (Satisfaction relation ($\models$))**.** The satisfaction relation $\models$ for interpretation $\mathcal{I}$ and atom $\phi \in \Gamma_{\mathsf{GDL}}$ is defined by structural induction in the following way:

- $\mathcal{I} \models p(t_1, \dots, t_n)$ iff $p(t_1^{\mathcal{I}}, \dots, t_n^{\mathcal{I}}) \in \mathcal{I}$,
- $\mathcal{I} \models \mathsf{init}(t)$ iff $\mathsf{init}(t^{\mathcal{I}}) \in \mathcal{I}$,
- $\mathcal{I} \models \mathsf{true}(t)$ iff $\mathsf{true}(t^{\mathcal{I}}) \in \mathcal{I}$,
- $\mathcal{I} \models \mathsf{next}(t)$ iff $\mathsf{next}(t^{\mathcal{I}}) \in \mathcal{I}$,
- $\mathcal{I} \models \mathsf{role}(t)$ iff $\mathsf{role}(t^{\mathcal{I}}) \in \mathcal{I}$,
- $\mathcal{I} \models \mathsf{does}(t_1, t_2)$ iff $\mathsf{does}(t_1^{\mathcal{I}}, t_2^{\mathcal{I}}) \in \mathcal{I}$,
- $\mathcal{I} \models \mathsf{legal}(t_1, t_2)$ iff $\mathsf{legal}(t_1^{\mathcal{I}}, t_2^{\mathcal{I}}) \in \mathcal{I}$,
- $\mathcal{I} \models \mathsf{goal}(t_1, t_2)$ iff $\mathsf{goal}(t_1^{\mathcal{I}}, t_2^{\mathcal{I}}) \in \mathcal{I}$,
- $\mathcal{I} \models \mathsf{distinct}(t_1, t_2)$ iff $t_1^{\mathcal{I}} \neq t_2^{\mathcal{I}}$.
- $\mathcal{I} \models \neg p(t_1, \dots, t_n)$ iff $\mathcal{I} \not\models p(t_1, \dots, t_n)$,
- $\mathcal{I} \models \forall X.\phi(X)$ iff $\mathcal{I} \models \phi(t)$ for all ground terms t in $\mathit{HU}_{\Gamma_{\mathsf{GDL}}}$.

Definition 34 (Rule Satisfiability)**.** An interpretation $\mathcal{I}$ *satisfies* a GDL rule $r \in \Gamma_{\mathsf{GDL}}$ (i.e., $\mathcal{I} \models r$) iff $\mathcal{I} \models e_i$ for all $e_i \in B^+(r)$ implies $\mathcal{I} \models H(r)$ and $\mathcal{I} \cap B^-(r) = \emptyset$.

Definition 35 (Herbrand Model)**.** An interpretation $\mathcal{I}$ is a model of Γ_{GDL} (i.e., $\mathcal{I} \models \Gamma_{\mathsf{GDL}}$) iff it satisfies each rule in Γ_{GDL}.

The semantics of a GDL program Γ_{GDL} (i.e., finite set of GDL rules) is defined by the set of Herbrand models that satisfy Γ_{GDL} for a given vocabulary of that program.

Definition 36 (Minimal model)**.** A Herbrand model $M_{\Gamma_{\mathsf{GDL}}}$ of GDL program Γ_{GDL} is called *minimal*, if there exist no Herbrand model $M_{\Gamma'_{\mathsf{GDL}}}$ of Γ_{GDL} such that $M_{\Gamma'_{\mathsf{GDL}}} \subset M_{\Gamma_{\mathsf{GDL}}}$.

We denote $MM(\Gamma_{\mathsf{GDL}})$ be the set of all minimal models of Γ_{GDL}.

Definition 37 (Least model)**.** A model $M_{\Gamma_{\mathsf{GDL}}}$ of GDL program Γ_{GDL} is called *least*, if $M_{\Gamma_{\mathsf{GDL}}} \subseteq M_{\Gamma'_{\mathsf{GDL}}}$ for every model $M_{\Gamma'_{\mathsf{GDL}}}$ of Γ_{GDL}, denoted by $L(\Gamma_{\mathsf{GDL}})$.

Thus, the least model is a minimal model but not necessarily the converse holds. The least model captures all the ground atomic logical consequences of the logic program. The least Herbrand model of a logic program Γ_{GDL} is denoted by $LM(\Gamma_{\mathsf{GDL}})$ and defined as follows.

Definition 38 (Least Herbrand Model (LM))**.** Let Γ_{GDL} be a definite GDL program then

$$LM(\Gamma_{\mathsf{GDL}}) = \bigcap\{M_{\Gamma_{\mathsf{GDL}}} | M_{\Gamma_{\mathsf{GDL}}} \text{ is Herbrand Model of } \Gamma_{\mathsf{GDL}}\}$$

$LM(\Gamma_{\mathsf{GDL}})$ is the least Herbrand model of GDL program i.e., Γ_{GDL}. Herbrand models are closed under intersection.

Proposition 39. If any number of Herbrand interpretations $\mathcal{I}_i$ where $1 \leq i \leq k$ are Herbrand models of any given GDL program Γ_{GDL} then so does $\mathcal{J} = \mathcal{I}_i \cap \cdots \cap \mathcal{I}_k$.

Proof. Let us assume that $\mathcal{J}$ is not a Herbrand model of the GDL program Γ_{GDL}. Then, there should exist a Herbrand interpretation $\mathcal{I}_i$ in which a ground rule $r \in \Gamma_{\mathsf{GDL}}$ such that $B^+(r) \in \mathcal{J}$ but $H(r) \notin \mathcal{J}$. It implies that $\mathcal{I}_i$ is not a Herbrand model of Γ_{GDL} which contradicts the assumption. Hence, $\mathcal{J}$ is also a Herbrand is the least model of GDL program Γ_{GDL}. □

Example 40. Consider the following definite GDL program

$$\begin{aligned}\Gamma_{\mathsf{GDL}} = \{&\mathsf{path}(X, Y) \Leftarrow \mathsf{edge}(X, Y),\\ &\mathsf{path}(X, Y) \Leftarrow \mathsf{edge}(X, Z), \mathsf{path}(Z, Y),\\ &\mathsf{edge}(a, b) \Leftarrow,\\ &\mathsf{edge}(b, c) \Leftarrow\}.\end{aligned}$$

Then, the Herbrand Universe $HU_{\Gamma_{\mathsf{GDL}}}$ is defined as

$$HU_{\Gamma_{\mathsf{GDL}}} = \{a, b, c\}$$

The Herbrand base is the predicate symbols over Herbrand universe defined as follows

$$\begin{aligned}HB_{\Gamma_{\mathsf{GDL}}} = \{&\mathsf{edge}(a,a), \mathsf{edge}(a,b), \mathsf{edge}(a,c),\\ &\mathsf{edge}(b,b), \mathsf{edge}(b,a), \mathsf{edge}(b,c),\\ &\mathsf{path}(a,a), \mathsf{path}(a,b), \mathsf{path}(a,c),\\ &\mathsf{path}(b,b), \mathsf{path}(b,a), \mathsf{path}(b,c),\\ &\mathsf{path}(c,c), \mathsf{path}(c,a), \mathsf{path}(c,b)\}.\end{aligned}$$

Since $\mathcal{I} \subseteq HB_{\Gamma_{\mathsf{GDL}}}$ then for the GDL program Γ_{GDL} its two interpretations are as follows

$$\begin{aligned}\mathcal{I}_1 &= \{\mathsf{edge}(b,b), \mathsf{path}(a,b), \mathsf{path}(c,b)\}.\\ \mathcal{I}_2 &= \{\mathsf{edge}(a,b), \mathsf{edge}(b,c), \mathsf{path}(b,b)\}.\end{aligned}$$

For the given GDL program Γ_{GDL}, the two Herbrand models are

$$\begin{aligned}M_1 &= \{\mathsf{edge}(a,b), \mathsf{edge}(b,c), \mathsf{path}(a,b), \mathsf{path}(b,c), \mathsf{path}(a,c)\}.\\ M_2 &= \{\mathsf{edge}(a,b), \mathsf{edge}(b,c), \mathsf{edge}(c,c), \mathsf{path}(a,b),\\ &\quad \mathsf{path}(b,c), \mathsf{path}(a,c), \mathsf{path}(b,a), \mathsf{path}(a,a)\}.\end{aligned}$$

Then, the least Herbrand model of program Γ_{GDL} would be

$$LM(\Gamma_{\mathsf{GDL}}) = \{\mathsf{edge}(a,b), \mathsf{edge}(b,c), \mathsf{path}(a,b), \mathsf{path}(b,c), \mathsf{path}(a,c)\}.$$

Apt et al. state in [8] that every definite logic program has a least model. Thus, every definite GDL program has a least model which can be computed using the fixed point operator which is explained in the following section.

3.3.3 *Fixed point Computation*

Van Emden and Kowalski [67] proposed an elegant way to compute the least model of logic programs without negation. Their idea was to use a natural closure operator and equate the models of a program Γ with the fixed-point operator, which is simpler to analyze. This operator is usually called T and defined as monotonic operator $T : 2^X \rightarrow 2^X$ where X is an arbitrary set and 2^X the power-set of X ordered by set inclusion. The following Knaster-Tarski theorem which is often known a Tarski's fixed point theorem [61] provides the base of this fixed point computation.

Theorem 41 (Knaster-Tarski Theorem)**.** For each set X a monotonic operator $T : 2^X \rightarrow 2^X$ has a least fixed point that is also the least pre-fixed point of T.

Proof. Lets assume that T is the monotonic operator on the partially order set $(\mathcal{B}, \leq)$. The least fixed point of T is a value $x \in \mathcal{B}$ that is the least among all fixed points of T, denoted as $lfp(\mathsf{T})$, and is characterized as $lfp(\mathsf{T}) = \mathsf{T}(lfp(\mathsf{T}))$. A least pre-fixed point of T, denoted by $lpp(\mathsf{T})$, is characterized as $\mathsf{T}(lpp(\mathsf{T})) \leq lpp(\mathsf{T})$ which is a value $x \in \mathcal{B}$ that is the least among all its pre-fixed points. From the following, it is simply entailed that a least fixed point is also a pre-fixed point. For all $x \in \mathcal{B}$, we have

$$lpp(\mathsf{T}) \leq x \Leftarrow \mathsf{T}(x) \leq x$$
$$lfp(\mathsf{T}) \leq x \Leftarrow x = \mathsf{T}(x)$$

$\square$

Let T_P is a monotonic operator like T whose domain and its domain and range is restricted to Herbrand Base. T_P is called the immediate consequence operator and it is used to compute the least fixed point of definite GDL programs.

Definition 42 (Immediate Consequence Operator)**.** [67] Let Γ_{GDL} be a definite GDL program and $\mathcal{I}$ be a Herbrand interpretation then the transformation operator $\mathsf{T}_\mathsf{P} : HB_{\Gamma_{\mathsf{GDL}}} \rightarrow HB_{\Gamma_{\mathsf{GDL}}}$ is

$$\mathsf{T}_\mathsf{P}(\mathcal{I}) = \{h | h \Leftarrow e_1 \wedge \cdots \wedge e_k \in \mathsf{ground}(\Gamma_{\mathsf{GDL}}) \wedge \mathcal{I} \vDash e_i \text{ for all } 1 \leq i \leq k\}$$

For definite GDL program Γ_{GDL}, Nilsson and Maluszynski [49] have shown that there exists an interpretation $\mathcal{I}$ such that $\mathsf{T}_\mathsf{P}(\mathcal{I}) = \mathcal{I}$ and the interpretation $\mathcal{I}$ is also the same as least Herbrand model $LM(\Gamma_{\mathsf{GDL}})$. $LM(\Gamma_{\mathsf{GDL}})$ could be constructed from an infinite sequence of iterations:

$$\emptyset, \mathsf{T}_\mathsf{P}(\emptyset), \mathsf{T}_\mathsf{P}(\mathsf{T}_\mathsf{P}(\emptyset)), \mathsf{T}_\mathsf{P}(\mathsf{T}_\mathsf{P}(\mathsf{T}_\mathsf{P}(\emptyset))), \ldots$$

Nilsson and Maluszynski [49] present a more standard notation which is used to denote the elements of the sequence of interpretations constructed from a logic program in the following way

$$\mathsf{T}_\mathsf{P} \uparrow 0 = \emptyset$$
$$\mathsf{T}_\mathsf{P} \uparrow (n+1) = \mathsf{T}_\mathsf{P}(\mathsf{T}_\mathsf{P} \uparrow n(\mathcal{I}))$$
$$\mathsf{T}_\mathsf{P} \uparrow \omega = \bigcup_{n=0}^{\infty} \mathsf{T}_\mathsf{P} \uparrow n$$

Example 43. For Example 40, the T_P is computed as follows

$$\begin{aligned}
\mathsf{T}_\mathsf{P} \uparrow 0 &= \emptyset \\
\mathsf{T}_\mathsf{P} \uparrow 1 &= \{\mathsf{edge}(a,b), \mathsf{edge}(b,c)\} \\
\mathsf{T}_\mathsf{P} \uparrow 2 &= \{\mathsf{edge}(a,b), \mathsf{edge}(b,c), \mathsf{path}(a,b), \mathsf{path}(b,c), \mathsf{path}(a,c)\} \\
&\vdots \\
\mathsf{T}_\mathsf{P} \uparrow \omega &= \{\mathsf{edge}(a,b), \mathsf{edge}(b,c), \mathsf{path}(a,b), \mathsf{path}(b,c), \mathsf{path}(a,c)\}
\end{aligned}$$

With this immediate consequence operator ($\mathsf{T_P}$) the following properties holds:

- $\mathsf{T_P}$ has a unique fixed point.
- $\mathcal{I}_1 \subseteq \mathcal{I}_2$ implies $\mathsf{T_P}(\mathcal{I}_1) \subseteq \mathsf{T_P}(\mathcal{I}_2)$.
- $\mathcal{I}$ is model of Γ_{GDL} iff $\mathsf{T_P}(\mathcal{I}) \subseteq \mathcal{I}$.

The Knaster-Tarski theorem's importance derives from the fact that for a definite GDL program Γ_{GDL}, the operator $\mathsf{T_P}$ is monotonic. A monotonic operator $\mathsf{T_P}$ has a least fixed point. The *lfp*($\mathsf{T_P}$) is computed iteratively by applying the operator $\mathsf{T_P}$ starting with the bottom element $\emptyset$ until a fixpoint is reached [8]. The least fixpoint contains all the ground atoms which are logical consequences of Γ_{GDL}. Thus, $\mathsf{T_P}$ provides an effective mechanism to compute models for a GDL program. Przymusinski [50] states that the idea behind least model semantics is to minimize the positive information to facts that are explicitly implied by the the GDL program Γ_{GDL} and making everything else false. This matches to the idea of closed world assumption [19].

Proposition 44. [25] The least-fixed point(*lfp*) of $\mathsf{T_P}$ coincides with the minimal Herbrand model of definite logic programs (i.e., programs without negation)

$$\mathit{lfp}(\mathsf{T_P}(\mathcal{I})) = \mathit{LM}(\Gamma_{\mathsf{GDL}})$$

where $\mathcal{I}$ is a Herbrand interpretation of the given program Γ_{GDL}.

Proof. Van Emden and Kowalski provide a detailed proof of this proposition in [67]. □

It is explained in [50] that the least model semantics is well defined only for the definite logic programs and not applicable to the normal logic programs. To define the semantics for normal GDL programs, we consider a syntactically restricted class of programs called the stratified GDL programs. The perfect model semantics is defined later on for this more expressive class of programs.

3.3.4 *Stratified Logic Programs*

In normal logic programs, the body of the rules also contain negative literals. It is important to realize that the normal logic programs are more expressive than definite logic programs [8].

A normal GDL program may have more than one minimal Herbrand model and computing the least model using immediate consequence operator $\mathsf{T_P}$ causes the following problems.

Example 45. Given a GDL program $\Gamma_{\mathsf{GDL}} = \{\mathfrak{a} \Leftarrow \neg\mathfrak{b}.\}$. $\mathcal{I} = \{\mathfrak{a}\}$ and $\mathcal{J}\{\mathfrak{b}\}$ are two distinct minimal models of Γ. $\mathcal{I}$ and $\mathcal{J} =$ are models of Γ but $\mathcal{I} \cap \mathcal{J} = \emptyset$. $\mathsf{T_P}(\emptyset) = \mathcal{I}$ and $\mathsf{T_P}(\mathcal{J}) = \emptyset$. $\mathsf{T_P}$ is not monotonic because $\emptyset \subseteq \mathcal{J}$ but $\mathcal{I} \not\subseteq \emptyset$.

The following is the only property which holds for the operator T_P.

Lemma 46. [8]

$$\mathcal{I} \text{ is model of } \Gamma \text{ iff } T_P(\mathcal{I}) \subseteq \mathcal{I}$$

Thus, we need to syntactically restrict normal GDL programs. A stratified program imposes certain syntactic restrictions on its syntax thus disallowing certain combination of recursion and negation. In a stratified program a negation is only applied to an already known predicate and variable occurring in a negative literal must occur with a positive literal of the same body rule. A way to check the stratification of a GDL program Γ_{GDL} is by constructing its dependency graph $DG(\Gamma)$.

Definition 47 (Dependency Graph)**.** The dependency graph of a program Γ_{GDL} is a directed, labeled graph, denoted by $DG(\Gamma_{GDL})$, whose nodes are the atoms that occur in Γ_{GDL} where there is a positive edge $\phi \xrightarrow{+} \psi$ where $\phi \in H(r)$ and $\psi \in B^+(r)$ for every rule $r \in \Gamma$ and a negative edge $\phi \xrightarrow{-} \psi$ where $\phi \in H(r)$ and $\psi \in B^-(r)$ for every rule $r \in \Gamma_{GDL}$.

Proposition 48. A GDL program Γ_{GDL} is said to be *stratified* if there is no cycle in the dependency graph $DG(\Gamma_{GDL})$ involving a negation.

To compute the stratification more formally, we need the following definition.

Definition 49 (Stratified Program)**.** A GDL program Γ_{GDL} is *stratified* iff there exists a function $strat \colon \mathcal{L}_{GDL} \to \mathbb{N}^+$ such that there is mapping for every rule $r \in \Gamma$ to a value $v \in \mathbb{N}$ with

1. $strat(h) = v$ for all $h \in H(r)$,
2. $strat(e) \leq v$ for all $e \in B^+(r)$,
3. $strat(e) < v$ for all $e \in B^-(r)$,

Eiter et al. [20] states that a stratification function, denoted by *strat*, can be efficiently found for a logic programs if it exists. The function *strat* induces a decomposition of a GDL program Γ_{GDL} into subsets of rules $\Gamma_0, \dots, \Gamma_k$ which are called the *stratum* of Γ_{GDL}. In particular, definite GDL programs are always stratified. We present the following example to further illustrate the GDL program stratification.

Example 50. Consider the following normal GDL program given by:

$$\begin{aligned}\Gamma_{GDL} = \{&path(X,Y) \Leftarrow \neg unconnected(X,Y),\\ &unconnected(X,Y) \Leftarrow point(X), point(Y), \neg edge(X,Y),\\ &edge(X,Y) \Leftarrow edge(X,Y), edge(Y,Z),\\ &edge(a,b), edge(a,c),\\ &edge(a,a), edge(b,b), edge(c,c),\\ &point(a), point(b), point(c)\}.\end{aligned}$$

The program stratification is as follows

$$\begin{aligned}\Gamma_1 = \{&\mathsf{point}(a), \mathsf{point}(b), \mathsf{point}(c), \mathsf{edge}(a,b), \mathsf{edge}(b,c),\\ &\mathsf{edge}(X,Y) \Leftarrow \mathsf{edge}(X,Y), \mathsf{edge}(Y,Z)\}.\\ \Gamma_2 =& \{\mathsf{unconnected}(X,Y) \Leftarrow \neg\mathsf{edge}(X,Y)\}.\\ \Gamma_3 =& \{\mathsf{path}(X,Y) \Leftarrow \neg\mathsf{unconnected}(X,Y)\}.\end{aligned}$$

GDL programs required that all rules should be stratified which implies that the GDL program completion is consistent because it does not contain negative recursion. In the following section, we provide the perfect model semantics for stratified GDL programs.

3.3.5 *Perfect Model Semantics*

The perfect model is defined as follows.

Definition 51 (Perfect Model)**.** [20] A Herbrand interpretation $\mathbf{M}$ is a *perfect model* of a GDL program Γ_{GDL} iff for a stratification $\Gamma_0, \ldots, \Gamma_k$ of Γ_{GDL}, $\mathbf{M} \cap HB_{\Gamma_i^*} \in MM(\Gamma_i^{*\mathbf{M}_i})$ where $\Gamma_i^* = ground(\Gamma_i)$, $\mathbf{M}_0 = \emptyset$ and $\mathbf{M}_i = \mathbf{M} \cap HB_{\Gamma_{i-1}^*}$ for $i = 1, \ldots, k,$.

The set of all perfect models of Γ_{GDL} is denoted by $PM(\Gamma)$. Perfect models are always minimal models. For definite GDL programs these semantics coincide with least model semantics.

Proposition 52. [51] Let Γ_{GDL} be a definite GDL program then $MM(\Gamma_{\mathsf{GDL}}) = PM(\Gamma_{\mathsf{GDL}})$.

Proposition 53. [51] Let Γ_{GDL} be a stratified GDL program then Γ_{GDL} has unique perfect model.

Example 54. [20] Consider the following stratified logic program:

$$\begin{aligned}\Gamma = \{&S(a) \Leftarrow .\\ &A(a,b) \Leftarrow .\\ &S(b) \Leftarrow \neg B(a,b).\\ &S(a) \Leftarrow A(a,b).\\ &S(b) \Leftarrow A(b,b).\}\end{aligned}$$

The minimal models of Γ are $\mathbf{M} = \{S(a), S(b)\}$ and $\mathbf{M}' = \{S(a), B(a,b)\}$, so $MM(\Gamma) = \{\{S(a), S(b)\}, \{S(a), B(a,b)\}\}$. From models $\mathbf{M}$ and $\mathbf{M}'$ only $\mathbf{M}$ is the unique perfect model $PM(\Gamma) = \{\{S(a), S(b)\}\}$.

Apt et al. [8] proposed an elegant way of computing the perfect model of any given stratified program. The perfect model semantics is based on the idea of constructing an iterated least model of a GDL program by applying the iterated closed world assumption [19]. We define the powers of monotonic operator T, defined in Theorem 41, in the following way

$$\begin{aligned}&\mathsf{T}\uparrow 0(\mathcal{I}) = \mathcal{I}\\&\mathsf{T}\uparrow(n+1)(\mathcal{I}) = \mathsf{T}(\mathsf{T}\uparrow n(\mathcal{I}))\cup\mathsf{T}\uparrow(n)(\mathcal{I})\\&\mathsf{T}\uparrow\omega(\mathcal{I}) = \bigcup_{n=0}^{\infty}\mathsf{T}\uparrow n(\mathcal{I})\end{aligned}$$

$\mathsf{T}\uparrow(n+1)(\mathcal{I})$ should not be confused with $\mathsf{T}^n(\mathcal{I})$, which stands for the n fold application of T. Given a stratified program Γ with its stratification $\Gamma_{GDL} = \Gamma_1\cup\cdots\cup\Gamma_n$. Let $\mathsf{T}_{P_1},\ldots,\mathsf{T}_{P_n}$ is a finite family of operators of T_P, defined in Definition 42, for each stratum $\Gamma_1\ldots\Gamma_n$ as

$$\begin{aligned}&M_1 = \mathsf{T}_{P_1}\uparrow\omega(\emptyset)\\&M_2 = \mathsf{T}_{P_2}\uparrow\omega(M_1)\\&\qquad\vdots\\&M_n = \mathsf{T}_{P_n}\uparrow\omega(M_{n-1})\end{aligned}$$

Theorem 55. [8] In above if we set $M_{\Gamma_{GDL}} = M_n$, then

- $\mathsf{T}_P(M_{\Gamma_{GDL}}) = M_{\Gamma_{GDL}}$, thus $M_{\Gamma_{GDL}}$ is supported model of Γ_{GDL}.
- $M_{\Gamma_{GDL}}$ is the unique perfect model of Γ_{GDL}.

Theorem 56. [8] If Γ_{GDL} is a stratified GDL program then $\mathsf{comp}(\Gamma_{GDL})$ is consistent.

Proof. Apt et al. provide a detailed proof in [8]. □

In GDL, we require all rules to be stratified which implies that the program completion is consistent. In [7] model of $\mathsf{comp}(\Gamma_{GDL})$ is characterized as a least fixed point of T_P also hold for stratified logic program. As a corollary to the consistency of completion, for any stratified GDL program Γ_{GDL} is as follows.

Corollary 57. [7] In a GDL program Γ_{GDL}, if $\mathsf{comp}(\Gamma_{GDL})$ is consistent then the unique perfect model $PM(\Gamma_{GDL})$ is also a model of $\mathsf{comp}(\Gamma_{GDL})$.

The perfect models of a stratified program is also a model of Clark's completion and thus the perfect model semantics are stronger than Clark's completion [50].

Theorem 58. [64] Let $\mathcal{I}$ is the unique perfect model of a given GDL program Γ_{GDL} then

$$\mathsf{comp}(\Gamma_{GDL}) \vDash \phi \text{ if } \phi\in\mathcal{I}$$

The perfect models for stratified GDL programs is the most suited declarative semantics. Przymusinski states that if a logic program is *stratified* and *safe* then it guarantees variable-free facts and finite derivablility [52]. As described earlier that GDL programs are both safe and stratified which concludes that GDL programs are finitely derivable and has variable-free facts as it is also stated in [62]. In the following section, we provide more intuitive transition-based semantics of Game Description Language (GDL).

3.4 GDL TRANSITION-BASED SEMANTICS

In an earlier work, van der Hoek et al. [65] outlined a transition-based semantics for GDL using ATL (Alternating-time Temporal Logic) [65, 64] which restricted the GDL to propositional logic with some further constraints on its syntax. These restrictions are necessary in order to perform an induction on reachable states [65, 64]. Schiffel and Thielscher [58] later on provide more comprehensive game based transition model for GDL programs. In [58, 57] multiagent environments are expressed in GDL which conversely provides semantics for GDL. It described in [64, 65, 62] that computing a game based transition model of a GDL program makes the informal description of Game Description Language (GDL) mathematically precise because it provides a precise meanings to the Predefined GDL atoms 17 and Fluents atoms 16 which were informally described earlier in Table 1. We had used reachability game structure introduced in the preceding chapter in Definition 7 to define a game-based model for any concurrent reachability game specified in GDL. In our work, we had specified concurrent reachability game in GDL which in turn be used to provide the transition-based semantics for GDL in terms of a transition-based game model.

As described in [63], we begin with a GDL game description Γ_{GDL} which implicitly determines a set of ground terms $\{t_1, \ldots, t_m\} \subseteq T_{GDL}$ and compute a game model in form of concurrent reachability structure The GDL terms T_{GDL} constitute the symbolic base in the declarative semantics for Γ_{GDL} as described in [57]. In the following section, we compute a game based transition model using reachability game structure for any game specified in GDL.

3.4.1 *Game Model Computation*

Let $Q = 2^{T_{GDL}}$ is the possible finite subsets, encoding every possible state in the game (i.e., set of fluents holds at particular state). We define a labeling function $L : Q \rightarrow 2^{T_{GDL}}$. It is described in [63] that from a given GDL program, the set of players and actions are immediately read off but the legal moves, update relation, termination and goal states are only defined with respective to the current state. In order to use the *concurrent reachability structure* as a game model $\mathcal{G}$ for a concurrent game specified in GDL, we need the following constructs:

Definition 59 (Current State)**.** The function $\mathcal{C} : Q \to 2^{\mathcal{A}_{FL}}$ maps a state to a set of GDL fluent atoms. Let q is any given state of $\mathcal{G}$ then

$$\mathcal{C}(q) = \{\mathtt{true}(t) : t \in L(q)\}$$

Determining a state transition requires the encoding of the current state along with rules representing a joint move from players in the game. To reach a successor state in the game, each player takes an action and the synchronous application of the joint actions by all players in the games at any current state results in the updated state.

Definition 60 (Action Execution)**.** The function $\mathcal{D} : \gamma^n \to 2^{\mathcal{A}_{PD}}$ maps a joint move (i.e., action tuple) to a set of GDL fluent atoms. Let $\langle \alpha_1, \ldots, \alpha_n \rangle$ is any given joint action then

$$\mathcal{D}(\langle \alpha_1, \ldots, \alpha_n \rangle) := \{\mathtt{does}(i, \alpha_i) : i \in \{1, \ldots, n\}\}$$

In a concurrent reachability game specified encoded as a GDL program Γ_{GDL}, the derivable instances of $\mathtt{role}(r)$, $\mathtt{init}(t)$ from Γ_{GDL} define the players and initial state respectively. The instances of $\mathtt{legal}(s, t)$ derivable from the GDL program Γ_{GDL} define all legal actions for player with respect to a current state in the game. In the same way, the rule for $\mathtt{terminal}$ and $\mathtt{goal}(v, r)$ define the terminal and goal state relative to the encoding of a given state as described in [58]. We compute a game model from a GDL program using the following definition.

$\mathcal{L}_{GDL}$ represents the class of Game Description Language (GDL)

Definition 61. The function $\zeta : \mathcal{L}_{GDL} \to \mathcal{G}$ computes a game model using reachability game structure for any given GDL program Γ_{GDL}. Let $\Gamma_{GDL} := \Upsilon_{role} \cup \Upsilon_{init} \cup \Upsilon_{glob} \cup \Upsilon_{next} \cup \Upsilon_{legal} \cup \Upsilon_{goal} \cup \Upsilon_{terminal}$ is a GDL specification of a program partitioned w.r.t. to rules types and ζ constructs a reachability game structure $\mathcal{G}$ from given GDL specification in the following way:

$$\mathcal{G} = (\langle \mathcal{P}, Q, q_0, \gamma, \xi, \delta, \tau \rangle, \mathcal{R}(r))$$

where

- $\mathcal{P} = \{r \in T_{GDL} : \mathtt{role}(r) \Leftarrow\in \Upsilon_{role}\}$,
- $Q = 2^{T_{GDL}}$ is set of the states,
- $q_0 = \{t \in T_{GDL} : \mathtt{init}(t) \Leftarrow\in \Upsilon_{init}\}$,
- $\gamma = \{\alpha \in T_{GDL} : \mathtt{legal}(r, \alpha) \Leftarrow\in \Upsilon_{legal}$ for every $r \in \mathcal{P}\}$,
- $\xi(r, q) = \{\alpha \in \gamma : \Gamma_{GDL} \cup \mathcal{C}(q) \vDash \mathtt{legal}(r, \alpha)\}$, for a given state $q \in Q$ and player $r \in \mathcal{P}$,
- $\delta(q, \langle \alpha_1, \ldots, \alpha_n \rangle) = \{t \in T_{GDL} : \Gamma_{GDL} \cup \mathcal{C}(q) \cup \mathcal{D}(\langle \alpha_1, \ldots, \alpha_n \rangle) \vDash \mathtt{next}(t)\}$, for a given state $q \in Q$ and a joint move $\langle \alpha_1, \ldots, \alpha_n \rangle$ where $\alpha_i \in \xi(i, q)$ for $i \in \mathcal{P}$,

- $\tau = \{q \in Q : \Gamma_{GDL} \cup \mathcal{C}(q) \vDash \mathtt{terminal}\}$,
- $\mathcal{R}(r) = \{q \in \tau : \Gamma_{GDL} a \cup \mathcal{C}(q) \vDash \mathtt{goal}(r, 100)\}$, for a player $r \in \mathcal{P}$.

For $q, q' \in Q$, we write $q \xrightarrow{\langle \alpha_1,\dots,\alpha_n \rangle} q'$ if $q' \in \delta(q, \langle \alpha_1, \dots, \alpha_n \rangle)$ where $\alpha_i \in \xi(i, q)$ for each player $i \in \mathcal{P}$. The above definition provides a transition based semantics by which a GDL description is interpreted as an abstract n-player game.

Lemma 62. Let us consider a GDL program Γ_{GDL} and its game model $\zeta(\Gamma_{GDL})$ where $q_i, q_{i+1} \in Q$. If $q_i \xrightarrow{\langle \alpha_1^i,\dots,\alpha_n^i \rangle} q_{i+1}$ then $\Gamma_{GDL} \cup \mathcal{C}(q_i) \cup \mathcal{D}(\langle \alpha_1^i, \dots, \alpha_n^i \rangle) \vDash \Gamma_{GDL} \cup \mathcal{C}(q_{i+1})$ where $\alpha_j^k \in \xi(j, q_i)$ for each player $j \in \mathcal{P}$.

Proof. By definition 61, it clear that every legal move at state q_i is obtained from standard the model $\Gamma_{GDL} \cup \mathcal{C}(q_k) \vDash \mathtt{legal}(j, \alpha_i^k)$ and each player $\mathcal{P}$ in $\zeta(\Gamma_{GDL})$ is a derivable instance because $\Gamma_{GDL} \vDash \mathtt{role}(j)$. Thus, the transition function $q_{i+1} \in \delta(q_i, \langle \alpha_1^i, \dots, \alpha_n^i \rangle)$ where the executed moves are represented using facts of form $\mathcal{D}(\langle \alpha_1^i, \dots, \alpha_n^i \rangle)$ combined with game description Γ_{GDL} is a model of the successor state $\Gamma_{GDL} \cup \mathcal{C}(q_{i+1})$. □

Theorem 63. Consider a GDL description of program Γ_{GDL} and its game model $\zeta(\Gamma_{GDL}) \equiv (\langle \mathcal{P}, Q, q_0, \gamma, \xi, \delta, \tau \rangle, \mathcal{R}(r))$. In a play $\pi = q_0 \xrightarrow{\langle \alpha_1^0,\dots,\alpha_n^0 \rangle} q_1 \xrightarrow{\langle \alpha_1^1,\dots,\alpha_n^1 \rangle} \dots q_{m-1} \xrightarrow{\langle \alpha_1^{m-1},\dots,\alpha_n^{m-1} \rangle} q_m$ such that $q_m \in \mathcal{R}(r)$ if and only if $\Gamma_{GDL} \cup \mathcal{C}(q_0) \cup \mathcal{D}(\langle \alpha_1^0, \dots, \alpha_n^0 \rangle) \vDash \Gamma_{GDL} \cup \mathcal{C}(q_1) \cup \mathcal{D}(\langle \alpha_1^1, \dots, \alpha_n^1 \rangle) \vDash \dots \Gamma_{GDL} \cup \mathcal{C}(q_{m-1}) \cup \mathcal{D}(\langle \alpha_1^{m-1}, \dots, \alpha_n^{m-1} \rangle) \vDash \Gamma_{GDL} \cup \mathcal{C}(q_m)$ such that $\Gamma_{GDL} \cup \mathcal{C}(q_m) \vDash \mathtt{goal}(r, 100)$.

Proof. In Lemma 62, it is shown that the a successor states correspond to the models of the GDL program Γ_{GDL}. By applying an induction on the length of the given player π, it can be easily shown that if $\zeta(\Gamma_{GDL})$ has a terminal state which belongs to goal-hood. If this is the case then GDL program Γ_{GDL} contains the derivable instance of the goal fact if the same joint moves that are performed in game model are carried over as moves to the GDL program. □

Corollary 64. Consider a GDL program Γ_{GDL} and its corresponding model $\zeta(\Gamma_{GDL})$ then a state q_m of $\zeta(\Gamma_{GDL})$ is in the winning play $q_m \in \mathcal{W}_r$ if and only if $\Gamma_{GDL} \cup \mathcal{C}(q_m) \vDash \mathtt{goal}(r, 100)$ for any given player r.

Let us consider the guard and intruder example, we introduced in the preceding chapter and write its GDL specification.

EXAMPLE: GUARD AND INTRUDER: We use again the Example 1 introduced in the preceding chapter. The GDL specification for this example is presented by the following program which is sub-form of the GDL specification presented in [58].

```
% **************************************************
% Guard & Intruder Example GDL source:
% **************************************************
 role(ag_1)⇐
 role(ag_2)⇐

 init(at(ag_1,1,1))⇐
 init(at(ag_2,1,5))⇐

 legal(R,stay) ⇐ true(at(R,X,Y))
 legal(ag_2,exit)⇐ ¬terminal ∧ true(at(ag_1,1,1))
 legal(R,move(D))⇐ ¬terminal ∧ true(at(R,U,V)) ∧
 adjacent(U,V,D,X,Y)

 adjacent(X,Y_1,north,X,Y_2) ⇐ co(X) ∧ succ(Y_1,Y_2)
 adjacent(X,Y_1,south,X,Y_2) ⇐ co(X) ∧ succ(Y_2,Y_1)
 adjacent(X_1,Y,east,X_2,Y) ⇐ co(Y) ∧ succ(X_1,X_2)
 adjacent(X_1,Y,west,X_2,Y) ⇐ co(Y) ∧ succ(X_2,X_1)
 co(1)⇐
 co(2)⇐
 co(3)⇐
 co(4)⇐
 co(5)⇐
 succ(1,2)⇐
 succ(2,3)⇐
 succ(3,4)⇐
 succ(4,5)⇐

 next(at(R,X,Y))⇐ does(R,stay) ∧ true(at(R,X,Y))
 next(at(R,X,Y))⇐ does(R,move(D)) ∧ true(at(R,U,V))
 ∧ adjacent(U,V,D,X,Y) ∧ ¬capture(R)
 next(at(ag_2,X,Y))⇐ true(at(ag_2,X,Y))
 ∧ capture(ag_2)

 capture(ag_2)⇐ true(at(ag_2,X,Y)) ∧ true(at(R,U,V))
 ∧ does(ag_2,move(D_1)) ∧ does(R,move(D_2))
 ∧ adjacent(X,Y,D_1,U,V) ∧ adjacent(U,V,D_2,X,Y)

 terminal⇐ true(at(ag_1,X,Y)) ∧ true(at(ag_2,X,Y))
 terminal⇐ ¬remain
 remain⇐ true(at(ag_2,X,Y))

 goal(R,0)⇐ role(R) ∧ ¬terminal
 goal(R,0)⇐ role(R) ∧ distinct(R,ag_2) ∧ terminal
 ∧ ¬remain
 goal(R,100)⇐ role(R) ∧ distinct(R,ag_2) ∧ terminal
 ∧ true(at(ag_2,X,Y))
 goal(ag_2,0)⇐ terminal ∧ true(at(ag_2,X,Y))
 goal(ag_2,100)⇐ terminal ∧ ¬remain
```

In keeping with Prolog convention, logical variables begin with a capital letter where constants, functions, and predicates begin with a lowercase letter. The game model of the above mentioned GDL program Γ_{GDL} is $\mathcal{G}$ is

$$\mathcal{G} = (\langle \mathcal{P}, Q, q_0, \gamma, \xi, \delta, \tau \rangle, \mathcal{R}_{Ag_2})$$

where

- $\mathcal{P} = \{\mathsf{ag\_1}, \mathsf{ag\_2}\}$.
- $Q = \{\mathtt{at(ag\_1,X\_1,Y\_1),at(ag\_2,X\_2,Y\_2)}\} \cup \{remain\}$ $\cup \{captured(\mathtt{ag\_2})\}$ where $\mathtt{X\_1,Y\_1,X\_2,Y\_2} \in \{1, \dots, 5\}$.
- $q_0 = \{\mathtt{at(ag\_1,1,1),\ at(ag\_2,5,5)}\}$.
- $\gamma = \{north, south, east, west, stay, exit\}$.
- For each player $i \in \mathcal{P}$, the legal actions are:
  - $\xi(i, q) = \{stay\}$ for any state $q \in Q \setminus \tau$.
  - $\xi(i, q) = \{north\}$ if $q = \{\mathtt{at(ag\_1,X\_1,Y\_1),at(ag\_2,X\_2,Y\_2)}\}$ where $\mathtt{Y\_1,Y\_2} < 5$.
  - $\xi(i, q) = \{south\}$ if $q = \{\mathtt{at(ag\_1,X\_1,Y\_1),at(ag\_2,X\_2,Y\_2)}\}$ where $\mathtt{Y\_1,Y\_2} > 0$.
  - $\xi(i, q) = \{east\}$ if $q = \{\mathtt{at(ag\_1,X\_1,Y\_1),at(ag\_2,X\_2,Y\_2)}\}$ where $\mathtt{X\_1,X\_2} < 5$.
  - $\xi(i, q) = \{west\}$ if $q = \{\mathtt{at(ag\_1,X\_1,Y\_1),at(ag\_2,X\_2,Y\_2)}\}$ where $\mathtt{X\_1,X\_2} > 0$.
  - $\xi(\mathsf{ag\_2}, q) = \{exit\}$ if $q = \{\mathtt{at(ag\_2,1,1),at(ag\_1,X,Y)}\}$ where $\mathtt{X} \neq 1$ and $\mathtt{Y} \neq 1$
  - $\xi(\mathsf{ag\_2}, q) = \{exit\}$ if $q = \{\mathtt{at(ag\_2,5,1),at(ag\_1,X,Y)}\}$ where $\mathtt{X} \neq 5$ and $\mathtt{Y} \neq 1$.
- $\delta(q, \langle \alpha_{\mathsf{ag\_1}}, \alpha_{\mathsf{ag\_2}} \rangle) = \{\mathtt{at(ag\_1,X\_1,Y\_1-1),at(ag\_2,X\_2,Y\_2+1)}\}$ if $q = \{\mathtt{at(ag\_1,X\_1,Y\_1),at(ag\_2,X\_2,Y\_2)}\}$, $\alpha_{\mathsf{ag\_1}}$=$south$ and $\alpha_{\mathsf{ag\_2}}$=$north$.
- $\delta(q, \langle \alpha_{\mathsf{ag\_1}}, \alpha_{\mathsf{ag\_2}} \rangle) = \{\mathtt{at(ag\_1,X\_1+1,Y\_1),at(ag\_2,X\_2+1,Y\_2)}\}$ if $q = \{\mathtt{at(ag\_1,X\_1,Y\_1),at(ag\_2,X\_2,Y\_2)}\}$, $\alpha_{\mathsf{ag\_1}}$=$east$ and $\alpha_{\mathsf{ag\_2}}$=$west$.
- $\delta(q, \langle \alpha_{\mathsf{ag\_1}}, \alpha_{\mathsf{ag\_2}} \rangle) = \{\neg remain, \mathtt{at(ag\_2,X\_2,Y\_2)}\}$ if $q = \{\mathtt{at(ag\_1,1,5),at(ag\_2,X\_2,Y\_2)}\}$, $\alpha_{\mathsf{ag\_1}}$=$exit$ and $\alpha_{\mathsf{ag\_2}}$=$stay$
- $\tau = \{escaped\}$ if $q = \{\neg remain, \mathtt{at(ag\_1,X,Y)}\}$ where $\mathtt{X,Y} \in \{1, \dots, 5\}$
- $\tau = \{captured(\mathtt{ag\_2})\}$ if $q = \{\mathtt{at(ag\_1,X\_1,Y\_1),at(ag\_2,X\_2,Y\_2)}\}$ where $\mathtt{X\_1\ =\ X\_2}$ and $\mathtt{Y\_1\ =\ Y\_2}$,
- $\mathcal{R}(\mathsf{ag\_2}) = \{goal(\mathtt{ag\_2,100})\}$ if $q = \{\mathtt{at(ag\_1,X,Y)}\}$ where $\mathtt{X,Y} \in \{1, \dots, 5\}$

- $\mathcal{R}$(ag_1) = {goal(ag_1,100)} if q = {at(ag_1,X_1,Y_1),at(ag_2,X_2,Y_2)} where X_1 = X_2 and Y_1 = Y_2.

The above example makes it clear that the reachability game structure for guard and intruder example is a game-based transition model for its GDL specification. Thus, we can compute more formal and intuitive transition-based models for GDL programs.

Human beings, who are almost unique in having the ability to learn from the experience of others, are also remarkable for their apparent disinclination to do so.

— Douglas Adams (1952 – 2001)

4 SOLVING CONCURRENT GAMES

In the preceding chapter, we had computed game based transition models for concurrent reachability games specified in GDL using the reachability game structure. In these game models, the players actions were defined in terms of primitive action relations (i.e. we assume legal moves on players behalf). To win a game there is a need to determine a winning strategy for any given player in a concurrent reachability game. In this chapter, we provide an algorithm for computing the winning strategy and a general game playing program for solving the reachability question in two-player concurrent reachability games for any given player. In the following, we present an algorithm for computing winning strategy for a player in concurrent reachability game.

4.1 WINNING STRATEGY COMPUTATION

It is described earlier that for closed systems the reachability question is as simple as solving a graph reachability problem [35]. It is in contrast to the game-based model which represents an open system as a two-player concurrent game and the question of reachability is to determine that whether any given player has a strategy to selects actions against it opponents in such a way that the play reaches a state which is in its goal states set. This problem is difficult to solve because opponents actions also influence the play of a game. Therefore, we need an algorithm for computing the winning strategy for any given player. In order to solve the reachability question in a concurrent game specified in GDL, we constitute a general game playing program. This program implements the winning strategy algorithm and solves the reachability question in two-player concurrent reachability games.

There is a growing literature on solving reachability games [40, 14, 15, 13, 31]. Hansen et al. [32] state two best-known algorithms, *value iteration* and *strategy iteration*, for approximately solving two-player concurrent reachability games. Both of these algorithms were derived from similar algorithms for solving Markov decision processes [33] and discounted stochastic games [59] and based on mixed strategies[1]. In

1 A mixed strategy is an assignment of a probability to each deterministic strategy. A deterministic strategy provides a complete definition of how a player will play a game.

this report, we only concentrate on deriving a winning strategy for any given player in a two-player concurrent reachable game.

In order to derive the winning regions and a winning strategy for any given player, we need the following constructs as a building block for our algorithms.

Definition 65 (Action sub-assignment)**.** For any player $\sigma \in \mathcal{P}$, an action sub-assignment $\mathfrak{I}_\sigma : Q \rightarrow 2^\gamma$ is a mapping that associates each state $q \in Q$ with a subset of legal moves for player σ at state q (i.e., $\mathfrak{I}_\sigma(q) \subseteq \xi(\sigma, q)$).

Definition 66. For any player $\sigma \in \mathcal{P}$, the function $Pre_\sigma : 2^Q \times 2^\gamma \rightarrow 2^Q$ is the set of states from which player σ surely reach to any set of states $U \subseteq 2^Q$ in one step

$$Pre_\sigma(U, \mathfrak{I}_\sigma) = \{q \in Q : \exists \alpha_\sigma \in \mathfrak{I}_\sigma(q) \forall \alpha_{\overline{\sigma}} \in \xi(\overline{\sigma}, q)\ \delta(q, \alpha_\sigma, \alpha_{\overline{\sigma}}) \subseteq U\}$$

$Pre_\sigma(U, \mathfrak{I}_\sigma)$ is the set of states from which player σ can be sure of entering U in one step, regardless of the move chosen by its opponent $\overline{\sigma}$, given that the player σ chooses a move according to action sub-assignments $\mathfrak{I}_\sigma$.

Definition 67. For any player $\sigma \in \mathcal{P}$, the function $Stay_\sigma : 2^Q \times 2^\gamma \rightarrow 2^\gamma$ is defined such that for all states $q \in Q$ and any subset of states $U \subseteq 2^Q$

$$Stay_\sigma(U, \mathfrak{I}_\sigma)(q) = \{\alpha_\sigma \in \mathfrak{I}_\sigma(q) : \forall \alpha_{\overline{\sigma}} \in \xi(\overline{\sigma}, q)\ \delta(q, \alpha_\sigma, \alpha_{\overline{\sigma}}) \subseteq U\}$$

$Stay_\sigma(U, \mathfrak{I}_\sigma)$ is the largest action sub-assignment for player σ that guarantees that the game stay in U for at least one round, regardless of the move chosen by player $\overline{\sigma}$, given that the player σ chooses a move according to action sub-assignments $\mathfrak{I}_\sigma$.

As described in earlier chapters that a concurrent reachability game is represented by a reachable game structure. In the following, we present an algorithm to derive the winning region for any given player in a two-player concurrent reachability game. This algorithm is quite similar to the one described in [3].

Algorithm 68. Algorithm to compute winning regions for player σ:

```
Input: (Two-player) reachability game structure $\mathcal{G}$ = $(\mathcal{A}, \mathcal{R}_\sigma)$)
Output: Winning region $\mathcal{WR}_\sigma$ for player $\sigma$
begin:
      Initialization: Let $U_0$ = $\mathcal{R}_\sigma$;
         Repeat For $k \geq 0$:
              let $U_{k+1}$ = $U_k \cup Pre_\sigma(U_k, \mathfrak{I}_\sigma)$;
         Until $U_{k+1}$ = $U_k$;
      Return: $U_k$;
end;
```

The following theorem provides us an algorithm to compute the winning strategy for any given player σ in a concurrent reachability game. This work is also inspired from similar work done in [3].

Theorem 69. Given a reachability game structure $\mathcal{G} = (\mathcal{A}, \mathcal{R}_\sigma)$, we can compute winning strategy strategy w.r.t. to winning region $\mathcal{WR}_\sigma$ in the following way:

1. Assume that Algorithm 68 terminates at iteration m and let $U_0, \dots, U_m$ be the set of states computed during the execution of the algorithm. Lets us define $\zeta : U_m \setminus \mathcal{R}_\sigma \to \gamma$ as

   $$\zeta(q) = \{Stay_\sigma(U_{h(q)-1}, \xi(\sigma, q))(q) : \text{ for all states } q \in U_m \setminus \mathcal{R}_\sigma\}.$$

   where function $h : U \setminus \mathcal{R}_\sigma \to \mathbb{N}$ is defined for each $q \in U_m \setminus \mathcal{R}_\sigma$ as

   $$h(q) = min\{j \in \{1, \dots, m\} : q \in U_j\}$$

   Let f^*_σ be a memoryless deterministic strategy for player σ that at every state $q \in U_m \setminus \mathcal{R}_\sigma$ deterministically choose a move from $\zeta(q) \neq \emptyset$. At other states in, f^*_σ is defined arbitrarily. Then f^*_σ is the winning strategy of player σ w.r.t. to its winning region $\mathcal{WR}_\sigma$.

Proof. Let us consider that the algorithm terminates at iteration m and let $U_0, \dots, U_m$ be the set of states computed during the execution of the algorithm and assume that f^*_σ is the winning strategy for player σ.

" $\Rightarrow$ " For $q \in U_m$, consider any $f_{\overline{\sigma}} \in \Pi_\sigma$ where Π_σ is the set of strategies of player σ . Let us consider any play $\pi \in \chi(q_i, f^*_\sigma, f_{\overline{\sigma}})$ where $0 \leq i \leq |\pi|$. From the definition of f^*_σ, it is immediate to see that for all $j \geq 0$, if $q_j \in U_m/\mathcal{R}_\sigma$ then $q_{j+1} \in U_m$ and either $q_{j+1} \in \mathcal{R}_\sigma$ or $h(q_j) > h(q_{j+1})$. Thus, π is the winning player $\pi \in \mathcal{W}_\sigma$.

" $\Leftarrow$ " In other direction, if $q \notin U_m$, then all $\alpha_\sigma \in \xi(\sigma, q)$ there is a $\alpha_{\overline{\sigma}} \in \xi(\overline{\sigma}, q)$ such that $\delta(q, \alpha_\sigma, \alpha_{\overline{\sigma}}) \notin U_m$. Hence all $q \notin U_m$ and all strategies $f_\sigma \in \Pi_\sigma$ there is a play $\pi \in \chi(q, f_\sigma, f^*_{\overline{\sigma}})$ such that $\pi \notin \mathcal{W}_\sigma$. $\square$

We want to enumerate over states in a play. For it we incorporate time as positive integer values in states. GDL programs are required to be temporalized in a way that is common for the encoding of temporal domains as logic programs [36, 21, 28]. Thielscher [62] suggests that in order to directly reason about the evolution of game states using any logic programming `time` should be encoded explicitly within the rules of GDL program which eliminates the need to apply rules repeatedly in order to determine the legal moves and their effects during a play at every state of the game as specified earlier in transition-based semantics of GDL. In the following section, we defined a temporal extension of GDL programs.

4.1.1 *GDL Temporal Extension*

Here, we define a temporal mapping which extend any given GDL program by replacing predicate names of some atoms and extending others with an additional time argument. We consider the time domain as a non-negative set of integers denoted by $\mathbb{Z}^+$. The auxiliary function $\mathsf{Pr} : \mathcal{A}_{\mathsf{GDL}} \to$ *Pred* which returns the predicate name of GDL atoms.

Definition 70 (Temporal mapping)**.** The function $\rho : \mathcal{A}_{\mathsf{GDL}} \to \mathcal{A}_{\mathsf{GDL}}$ maps each GDL atom to an another GDL atom in the following way

- If $\mathsf{Pr}(\phi) \notin \{role, distinct, next, init\}$ then just add T as an additional argument to ϕ. T represents the variable whose domain is $\mathbb{Z}^+$.
- If $\mathsf{Pr}(\phi) \in \{init\}$ then replace it with predicate $true$ and add 0 as an additional argument to ϕ,
- If $\mathsf{Pr}(\phi) \in \{next\}$ then replace it with predicate $true$ and add an additional argument $T + 1$ to ϕ,

The temporal extension of any given GDL program Γ is defined by

$$\rho(\Gamma) = \{r : \text{ apply } \rho(\phi) \text{ to each } \phi \in H(r) \cup B(r) \text{ for every rule } r \in \Gamma\}$$

This mapping can be made more efficient by omitting the time argument from atoms in $\mathcal{A}_{\mathsf{NR}}$.

GDL logic-based semantics remains unchanged even after its temporal extension because of the seamless renaming of the GDL atoms. Now only a few changes are required in GDL transition-based semantics to accommodate GDL temporal extension as described in [63]. We need the following action function to prove the correctness of GDL temporal extension in case of transition-based semantics presented in the last chapter.

Definition 71 (Action Execution)**.** The function $\mathcal{D} : \gamma^n \times \mathbb{Z}^+ \to 2^{\mathcal{A}_{\mathrm{PD}}}$ maps a joint move (i.e., action tuple) to a set of GDL facts. Let $\langle \alpha_1, \dots, \alpha_n \rangle$ is any given joint action then

$$\mathcal{D}(\langle \alpha_1, \dots, \alpha_n \rangle, T) :\overset{\mathrm{def}}{=} \{does(i, \alpha_i, T) : i \in \mathcal{P}\}$$

The correctness of mapping of a GDL program onto an answer set program with time is provide as follows. The following theorem is also described in [63] but for single player games but we had proved it for multi-players games.

Theorem 72. [63] Let Γ be a GDL program description with a model $\mathcal{G} = (\langle \mathcal{P}, Q, q_0, \gamma, \xi, \delta, \tau \rangle, \mathcal{R}(r))$. For any finite sequence of legal joint moves and states we have that

$$q_0 \xrightarrow{\langle \alpha_1^0, \dots, \alpha_n^0 \rangle} q_1 \dots q_{m-1} \xrightarrow{\langle \alpha_1^{m-1}, \dots, \alpha_n^{m-1} \rangle} q_m$$

if and only if the perfect model $\mathcal{M}$ of

$$\rho(\Gamma_{\mathrm{GDL}}) \cup \mathcal{D}(\langle \alpha_1^0, \ldots, \alpha_n^0 \rangle, 0) \cup \cdots \cup \mathcal{D}(\langle \alpha_1^{m-1}, \ldots, \alpha_n^{m-1} \rangle, m-1)$$

satisfies the following:

- $q_i = \{t \in \mathcal{T}_{\mathrm{GDL}} : \mathcal{M} \vDash \mathsf{true}(t, i)\}$ for each $1 \leq i \leq m$ and
- $\mathcal{M} \vDash \mathsf{legal}(r, \alpha_r, i)$ for each player $r \in \mathcal{P}$ and each $1 \leq i \leq m$.

Proof. We proceed by doing an induction on m.

- Let us consider the case when $m = 0$. In this case, the $\mathsf{init}(t)$ got replaced with $\mathsf{true}(t, 0)$ in $\rho(\Gamma)$. By considering game model construction shown in Definition [61], it correspond to the initial state q_0.
- Consider the case when $m > 0$, then $\mathsf{legal}(r, \alpha_r)$ got replaced with $\mathsf{legal}(r, \alpha_r, i)$ in $\rho(\Gamma_{\mathrm{GDL}})$. By Definition [61], $\mathsf{legal}(r, \alpha_r, i)$ still corresponding to the legal moves $\xi(r, i)$ at that state.
- By induction, $\mathsf{true}(t)$, $\mathsf{next}(t)$ and $\mathsf{does}(i, \alpha)$ are replaced with $\mathsf{true}(t, m)$, $\mathsf{true}(t, m+1)$ and $\mathsf{does}(i, \alpha, m)$ in in $\rho(\Gamma)$. The game model construction shown in Definition [61] still remain intact.

□

In order to solve the reachability question in a concurrent game specified in GDL, we constitute a general game playing program which is presented in the following section.

4.2 GENERAL GAME PLAYING PROGRAM

The general game playing program implements the winning strategy algorithm and solves the reachability question in two-player concurrent reachability games. This game player is implemented in Answer Set Programming (ASP) [47] because it is a declarative programming language oriented towards solving difficult search program primarily NP-hard problems [47]. General game players [38, 30, 62] developed using answer set programming has shown promising results for solving single player games. Another benefit of using answer set programming is that the GDL specifications can be mapped onto a answer set program efficiently [62]. The general game playing program presented in this chapter is an multi-player program based on Thielscher [62] single player general game player. Our general game player solves the reachability question for any given player in two-player concurrent reachability games.

Guo [30] and Thielscher [62] use Answer Set Programming (ASP) to develop a single-player game playing program which performs a depth restricted search to determine the winning position and define the mapping of GDL specification onto an answer set program in such a way that the models of the program coincides with the solutions in single player game environment.

We describe the rules for our *general game playing program* which encode the winning strategy for any given player ν in game Γ, denoted by $\kappa(\Gamma, \nu)$ which are specified as follows.

1. Each player has to perform a single legal move in each round (time point) unless a terminal position has been reached.

   ```
   1 {does(P, M, T): moves(M)} 1 :- role(P),
     actiontime(T),not terminal(T).
   0 {does(P, M, T): moves(M)} 1 :-role(P),
     actiontime(T),terminal(T).
   ```

2. Each player only plays a legal move.

   ```
   :- does(P, M, T), not legal(P, M, T), role(P),
      moves(M), actiontime(T).
   ```

3. The terminal position determines the intended goal value of given player ν.

   ```
   :- terminal(T), not terminal(T-1), not goal(v,100,T).
   :- terminal(1), not goal(v,100,1).
   ```

   Here, ν is assumed to be the player (i.e., $\mathtt{role}(\nu)$) for which the game player is choosing moves and 100 stands for the goal value its trying to achieve.

4. Determine the best move move at each round (time point) in the game and then progress the play by playing only that move for the given player ν.

   ```
   1{does(v,M,T): in(move_domain(Move),T)}1:- goal(v,100,T + 1).
   :- does(v,M,T), not in(M,T), not terminal(T).
   ```

The `move_domain(M)` is computed by generating a dependency graph of given game description and its method is described in [62]. Thielscher [62] describes a similar set of rules in [62] in case of single player games. In following, we prove it for two-player concurrent games.

4.2.1 *Correctness of Solving Program*

Thielscher proves in his paper [62], the correctness of the above mentioned solving method in single players games. The correctness of the solving method for two-player games is given by two following theorems.

Theorem 73. [62] Consider a GDL description Γ with semantics $\mathcal{G} = (\langle \mathcal{P}, Q, q_0, \gamma, \xi, \delta, \tau \rangle, \mathcal{R}(\nu))$. If Π is an answer set for $\rho(\Gamma) \cup \kappa(\Gamma, \nu)$ where ν is the given player then there exists $m \geq 0$ such that

$$\mathsf{does}(\nu, \langle \alpha^0_\sigma, \alpha^0_{\overline{\sigma}} \rangle, 0), \dots, \mathsf{does}(\nu, \langle \alpha^{m-1}_\sigma, \alpha^{m-1}_{\overline{\sigma}} \rangle, m-1)$$

are exactly the positive instances of predicate does in Π then there are states $q_1, \dots, q_m$ such that

- $q_0 \xrightarrow{\langle \alpha^0_\sigma, \alpha^0_{\overline{\sigma}} \rangle} q_1 \dots q_{m-1} \xrightarrow{\langle \alpha^{m-1}_\sigma, \alpha^{m-1}_{\overline{\sigma}} \rangle} q_m$,
- $q_m \in \tau$ and
- $(\nu, 100, q_m) \in \mathcal{R}(\nu)$.

Proof. Since $\rho(\Gamma) \cup \kappa(\Gamma, \nu)$ contains only rules with does in the head, the program $\kappa(\Gamma, \nu)$ ensures the existence of finite set of sequence of actions $\{\langle \alpha^0_\sigma, \alpha^0_{\overline{\sigma}} \rangle, \dots, \langle \alpha^{m-1}_\sigma, \alpha^{m-1}_{\overline{\sigma}} \rangle\}$ that are the instances of does in Π. From Theorem [72] and rules in the program $\kappa(\Gamma, \nu)$ it follows that there are states $q_1, \dots, q_{m-1}$ such that

$$q_0 \xrightarrow{\langle \alpha^0_\sigma, \alpha^0_{\overline{\sigma}} \rangle} q_1 \dots q_{m-1} \xrightarrow{\langle \alpha^{m-1}_\sigma, \alpha^{m-1}_{\overline{\sigma}} \rangle} q_m \text{ where } q_m \in \tau$$

The reachability algorithm in $\kappa(\Gamma, \nu)$ ensures that $(\nu, 100, q_m) \in \mathcal{R}(\nu)$. $\square$

Theorem 74. [62] Consider a GDL description Γ with semantics $\mathcal{G} = (\langle \mathcal{P}, Q, q_0, \gamma, \xi, \delta, \tau \rangle, \mathcal{R}(\nu))$. Let $q_0, \dots, q_m$ is a sequence of states such that

- $q_0 \xrightarrow{\langle \alpha^0_\sigma, \alpha^0_{\overline{\sigma}} \rangle} q_1 \dots q_{m-1} \xrightarrow{\langle \alpha^{m-1}_\sigma, \alpha^{m-1}_{\overline{\sigma}} \rangle} q_{m-1}$,
- $q_m \in \tau$ and
- $(\nu, 100, q_m) \in \mathcal{R}(\nu)$.

Then $\rho(\Gamma) \cup \kappa(\Gamma, \nu)$ admits an answer set Π in which

$$\mathsf{does}(\nu, \langle \alpha^0_\sigma, \alpha^0_{\overline{\sigma}} \rangle, 0), \dots, \mathsf{does}(\nu, \langle \alpha^{m-1}_\sigma, \alpha^{m-1}_{\overline{\sigma}} \rangle, m-1)$$

are exactly the positive instances of predicate does.

Proof. From Theorem [72] and the fact that the only answer set of $\rho(\Gamma)$ coincides with the perfect model and given that the sequence $q_0, \dots, q_{m-1}$ does not contain terminal state. It follows that there is an answer set for $\rho(\Gamma)$ that are the instances of predicate does. The Π is the answer set for the program $\rho(\Gamma) \cup \kappa(\Gamma, \nu)$ because

- $\alpha^k_i \in \xi(r, q_k)$ for each $k \in \{1, m-1\}$ and for every player $r \in \mathcal{P}$,
- $q_m \in \tau$ and
- $(\nu, 100, q_m) \in \mathcal{R}(\nu)$.

$\square$

EXAMPLE: GUARD & INTRUDER

The following ASP program $\kappa(\Gamma, ag_2)$ plays the Guard and Intruder game according to the winning strategy of intruder (ag_2). In keeping

```
% **************************************************
% ASP Solving Program
% **************************************************
  move_domain(stay).
  move_domain(exit).
  move_domain(north).
  move_domain(south).
  move_domain(east).
  move_domain(west).
  time_bound(100).

  1 {does(P, Move, T) : move_domain(Move)} 1 :-
   role(P), actiontime(T), not terminal(T).
  0 {does(P, Move, T) : move_domain(Move)} 0 :-
   role(P), actiontime(T), terminal(T).
  :- does(P,M,T), not legal(P,M,T), role(P),
   role(P), moves(M), actiontime(T).
  :- terminal(1), not goal(ag_2,100,1).
  1 { does(ag_2,M,T): in(move_domain(Move),T)} 1 :-
   goal(ag_2,100, T + 1).
  :- does(ag_2,M,T), not in(M,T), not terminal(T).
```

up with Prolog convention, logical variables begin with a capital letter; constants, functions, and predicates begin with a lowercase letter. In General Game Playing (GGP) competitions for the ease of computation KIF (Knowledge Interchange Format) [26] is used as an official language to represent GDL (Game Description Language) specifications. We translate KIF into an an answer set logic program using a method described in [30] which replace atoms like $\mathrm{distinct}(t_1, t_2)$ with $t_1 \mathrel{!}= t_2$ and symbols like "$\Leftarrow$" and "$\wedge$" with "$:-$" and "," respectively. The following ASP program is obtained from Guard and Intruder Example ??.

```
% ************************************************
% Solving Program for Guard \& Intruder Example
% ************************************************
 role(ag_1).
 role(ag_2).

 true(at(ag_1,1,1),0).
 true(at(ag_2,1,5),0).

 legal(R,stay,T) :- true(at(R,X,Y),T).
 legal(ag_2,exit,T) :- not terminal(T),

 true(at(ag_1,1,1),T).
 legal(R,move(D),T) :- not terminal(T),
 true(at(R,U,V),T),

 adjacent(U,V,D,X,Y).
 adjacent(X,Y_1,north,X,Y_2):- co(X,T),succ(Y_1,Y_2).
 adjacent(X,Y_1,south,X,Y_2):- co(X,T),succ(Y_2,Y_1).
 adjacent(X_1,Y,east,X_2,Y):- co(Y,T),succ(X_1,X_2).
 adjacent(X_1,Y,west,X_2,Y):- co(Y,T),succ(X_2,X_1).
 co(1).
 co(2).
 co(3).
 co(4).
 co(5).
 succ(1,2).
 succ(2,3).
 succ(3,4).
 succ(4,5).

 true(at(R,X,Y),T+1):- does(R,stay,T),
 true(at(R,X,Y),T).
 true(at(R,X,Y),T+1):- does(R,move(D)),
 true(at(R,U,V),T),
 adjacent(U,V,D,X,Y), not capture(R,T).
 true(at(ag_2,X,Y),T+1):- true(at(ag_2,X,Y),T),
 capture(ag_2,T).
 capture(ag_2,T):- true(at(ag_2,X,Y),T),
 true(at(R,U,V),T),
 does(ag_2,move(D_1),T), does(R,move(D_2),T),
 adjacent(X,Y,D_1,U,V), adjacent(U,V,D_2,X,Y).
 terminal(T):- true(at(ag_1,X,Y),T),
 true(at(ag_2,X,Y),T).
 terminal(T):- not remain(T).
 remain(T):- true(at(ag_2,X,Y),T).

 goal(R,0,T):- role(R), not terminal.
 goal(R,0,T):- role(R), R != ag_2, terminal(T),
 not remain(T).
 goal(R,100,T):- role(R), R != ag_2, terminal,
 true(at(ag_2,X,Y),T).
 goal(ag_2,0,T):- terminal(T), true(at(ag_2,X,Y),T).
 goal(ag_2,100,T):- terminal(T), not remain(T).
```

Programs must be written for people to read,
and only incidentally for machines to execute.

— H. Abelson and G. Sussman

5 IMPLEMENTATION & RESULTS

Kuhlmann et al., König et al. [41, 38] and Schiffel and Thielscher [56] develop successful general game players which use domain dependent heuristic functions to play a game intelligently. These domain dependent heuristic functions largely rely on a small set of generic features (piece-values, mobility, similarity etc) of a game [22]. We develop a general game playing program which implements the reachability algorithm, described in the preceding chapter, to compute the winning strategy for any given player in a concurrent reachability game. The implemented game player plays any multi-player game described in GDL and tries to win it; thus, solving the reachability question in concurrent reachability games. In this chapter, we briefly describe the implementation details of our general game player. We conclude this chapter by providing some essential results which evaluate the performance of our game player.

5.1 IMPLEMENTATION

Love et al. [45] state that the general game player must be a complete and fully autonomous program which accept a formal description of an arbitrary game specified in GDL and then plays that game effectively without any human interaction. We develop our general game player using JAVA and Answer Set Programming (ASP). The different components of the game player are explained as follows.

CONNECTION MANAGER

The connection manager is the interface between the game player and the Game manager (GM) [45]. Game manager is a program that mediates game plays between players by sending each player the game description described in GDL. The game manager requires the game player to be a running HTTP server. The game player supports the HTTP connection for both sending and receiving messages because the communication between game manager and game players should be done using HTTP [11]. A game begins when the player receives a message from the game manager announcing a new game. The requests from GM to the game player should be replied appropriately. The communication protocol is explained in [45]. The game player extract the game description from the input message and parsed it using the following parsing component.

GDL PARSER

The parser component in our game player translates the games rules specified in GDL onto an answer set logic program. GDL parser is also responsible to extend the given GDL game description into a temporalized extension which is explained in the preceding chapter.

REASONING ENGINE

Due the closeness of Answer set programming (ASP) in syntax and semantics with GDL, the generated answer set program is then used to reason about game-specific state-space manipulation (generation and execution of legal moves, detection of terminal and goal states) [62]. Our player can use any of answer set solvers (i.e., DLV [43], CLASP [24], SMODELS [48]) to obtain the models of the given answer set program. The main responsibility of this component is to generate legal moves for the players in a game and then use the following searching strategy to select the best move among the available moves.

SEARCHING STRATEGY

The searching strategy helps to determine the best playing move during a game play. The general game players [41, 56, 38] have been based on the traditional approach of using game-tree search which requires a heuristic evaluation function for encapsulating the game specific knowledge. This automatically generated evaluation function reply on a small set of generic features namely piece-values, mobility, similarity etc. In our game player, we had used Iterative deepening depth-first search (IDDFS) [54]. In this searching technique, a depth-limited search is run repeatedly, increasing the depth limit with each iteration until it reaches to the maximal value [54]. We had set this maximal value depending on the play clock of the game. In case the game player found the goal state, the player chooses the most promising move which helps it to win the game. Otherwise, we had used an evaluation function that is based on the cardinality criteria of legal moves set (i.e., The action is more favorable if player reaches to a state from the current state where the player has more legal moves). Although, evaluation functions based on the general criteria like piece-values, mobility, cardinality and similarity etc are applicable to a wider range of games but sometimes fail to capture very essential game properties because of the inaccuracy of the resulting heuristic.

5.2 RESULTS

The expressiveness of GDL is sufficient to express a large number of complete information single or multi-player turn-based and concurrent games. To evaluate the performance of our general game player, we had played several single and multi-player games and list the obtained results in the following section.

5.2.1 *Single Player Games*

We had played single player games namely the famous Maze [54], Peg Solitaire (4x4 board) and Towers of Hanoi (6 disks). Table 2 shows the results in case of single player games. Maze and Peg Solitaire games were

GAME NAME	ROUNDS	SOLVED
Maze	6	Yes
Peg Solitaire	31	Yes
Towers of Hanoi	67	N/A

Table 2: Single Player Games Results Table

solved but our game player unable to solve Towers of Hanoi because it has a very large search space.

5.2.2 *Multi-player Games*

In Table 3, we had provided the results obtained when playing multi-player games between our game player, called reachability player, and the legal player. The legal player plays randomly (i.e., choose moves from the set of its available moves at random). All the games, we had played are two-player games. Tic-Tac-Toe is a turn-based two-player game. Guard & Intruder and Rock-paper-scissors are two-player concurrent games.

GAME NAME	MATCHES	σ	$\overline{\sigma}$
Tic-Tac-Toe	10	10	0
Guard & Intruder	10	10	0
Rock-paper-scissors	10	3	7

Table 3: σ := Reachability Player $\overline{\sigma}$:= Legal Player

For Tic-Tac-Toe our game player played as a first player and wins every game against the legal player. For Guard and Intruder game, the reachability player played as an intruder and able to escape without being caught in all the games as again shown in Table 3. For the case of Rock-paper-scissors, surprisingly our player looses the competition because this game is best played using a random strategy (i.e., Randomly choosing a move) [13] and the deterministic strategy fails to obtain a good result in this game as showed in Table 3.

We had played again all the above games but this time switching the roles of our player and list the results in Table 4. In case of Tic-Tac-Toe, our reachability player played as a second player but still able to win all the games. In Guard and Intruder example, the legal player playing as guard catches the intruder in once instance because in this game playing as guard is an advantage.

GAME NAME	MATCHES	σ	$\overline{\sigma}$
Tic-Tac-Toe	10	0	10
Guard & Intruder	10	1	9
Rock-paper-scissors	10	6	4

Table 4: σ := Legal Player $\overline{\sigma}$:= Reachability Player

Deep Thought [38] is a successful general game player also based on answer set programming approach. Now, we play all the games against Deep Thought. Table 5 showed the results of the game where our reachability player is making first move in case of Tic-Tac-Toe and playing as an intruder for Guard & Intruder game.

GAME NAME	MATCHES	σ	$\overline{\sigma}$
Tic-Tac-Toe	10	8	2
Guard & Intruder	10	7	3
Rock-paper-scissors	10	6	4

Table 5: σ := Reachability Player $\overline{\sigma}$:= Deep Thought [38]

Table 6 shows results in which both players are the reachability players (We had run two different instances of our player). Table 6 showed that for Tic-Tac-Toe, playing first is an advantage and in Guard & Intruder game playing guard is an advantage over playing as an intruder.

GAME NAME	MATCHES	σ	$\overline{\sigma}$
Tic-Tac-Toe	10	7	3
Guard & Intruder	10	6	4
Rock-paper-scissors	10	4	5

Table 6: σ := Reachability Player $\overline{\sigma}$:= Reachability Player

6 CONCLUSION

In this report, we had specified concurrent reachability games in Game Description Language (GDL) and implemented a general game playing program using Answer Set Programming (ASP). This game player solves the reachability question for any given player in a concurrent reachability game. The searching technique used in this general game player could be applied to a wide range of games but for some games it could generate a highly inaccurate evaluation function. The game GO is an example to such a scenario [22]. In order to solve this problem, Finnsson and Björnsson proposed a simulation-based approach called Monte-Carlo Tree Search (MCTS) [22] which does not require a prior domain knowledge. For future investigations, a Monte-Carlo Tree Search (MCT) with Upper Confidence Bounds applied to Trees (UCT) [12] can be implemented in the game player because it eliminates the need of a heuristic evaluation function. Further research work can also be directed to study the relationship between the parity game [29] and symbolic planning [10] with Game Description Language (GDL) and General Game Playing (GGP).

BIBLIOGRAPHY

[1] Matrin Abadi, Leslie Lamport, and Pierre Wolper. Realizable and unrealizable concurrent program specifications. In *Proceedings of Sixteenth International Colloquium on Automata Languages and Programming*, volume 372, pages 1–17. Springer-Verlag, 1989.

[2] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*. Addison-Wesley, 1995. ISBN 9780201537710.

[3] Luca De Alfaro, Thomas A. Henzinger, and Orna Kupferman. Concurrent reachability games. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:564, 1998. ISSN 0272-5428.

[4] Luca De Alfaro, Thomas A. Henzinger, and Freddy Y. C. Mang. Detecting errors before reaching them. In *In CAV00, LNCS*, volume 1855, pages 186–201. Springer, 2000.

[5] Rajeev Alu, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:100, 1997. ISSN 0272-5428.

[6] R. Alur, P. Madhusudan, and Wonhong Nam. Symbolic computational techniques for solving games. *Electronic Notes in Theoretical Computer Science*, 89(4), 2003.

[7] Krzysztof R. Apt and M. H. van Emden. Contributions to the theory of logic programming. *J. ACM*, 29:841–862, July 1982.

[8] Krzysztof R Apt, Howard A. Blair, and Adrian Walker. *Towards a theory of declarative knowledge*, pages 89–148. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988. ISBN 0-934613-40-0.

[9] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008. ISBN 978-0-262-02649-9.

[10] Marco Bakera, Stefan Edelkamp, Peter Kissmann, and Clemens D. Renner. Solving μ-calculus parity games by symbolic planning. In *Model Checking and Artificial Intelligence: 5th International Workshop*, pages 15–33, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-00430-8.

[11] T. Berners-Lee and D. Connolly. *Hypertext Markup Language - 2.0*. RFC Editor, United States, 1995.

[12] Yngvi Björnsson and Hilmar Finnsson. Cadiaplayer: A simulation-based general game player. *IEEE Trans. Comput. Intellig. and AI in Games*, 1(1):4–15, 2009.

[13] Krishnendu Chatterjee, Marcin Jurdzinski, and Rupak Majumdar. On nash equilibria in stochastic games. Technical Report UCB/CSD-03-1281, EECS Department, University of California, Berkeley, Oct 2003.

[14] Krishnendu Chatterjee, Luca de Alfaro, and Thomas A. Henzinger. Strategy improvement for concurrent safety games. *CoRR*, abs/0804.4530, 2008.

[15] Krishnendu Chatterjee, Luca de Alfaro, and Thomas A. Henzinger. Termination criteria for solving concurrent safety and reachability games. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 09, pages 197–206, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

[16] Keith L. Clark. Negation as failure. In J. Nicolas H. Gallaire, J. Minker, editor, *In Logic and Databases*. Plenum Press, 1978.

[17] Tang Coe. Inside the pentium fdiv bug. *Journal of Software Tools*, 20:129–135, 1995.

[18] Luca de Alfaro. Game models for open systems. In *Verification: Theory and Practice'03*, pages 269–289, 2003.

[19] R. Eiter. On closed-world data bases. In H Garllaire and J. Minker, editors, *Logic and Data Bases*, pages 55–76. Plenum Press, Newyork, 1978.

[20] Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive datalog. *ACM Trans. Database Syst.*, 22(3):364–418, 1997. ISSN 0362-5915.

[21] Michael Ernst, Todd Millstein, and Daniel S. Weld. Automatic sat-compilation of planning problems. In *IJCAI-97, Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 1169–1176. Morgan Kaufmann, 1997.

[22] Hilmar Finnsson and Yngvi Björnsson. Simulation-based approach to general game playing. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 1*, pages 259–264. AAAI Press, 2008. ISBN 978-1-57735-368-3.

[23] Dov M. Gabbay, C. J. Hogger, and J. A. Robinson, editors. *Handbook of logic in artificial intelligence and logic programming (vol. 1)*. Oxford University Press, Inc., New York, NY, USA, 1993. ISBN 0-19-853745-X.

[24] M. Gebser, B. Kaufmann, and T. Schaub. The conflict-driven answer set solver *clasp*: Progress report. In E. Erdem, F. Lin, and T. Schaub, editors, *Proceedings of the Tenth International*

Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09), volume 5753 of *Lecture Notes in Artificial Intelligence*, pages 509–514. Springer-Verlag, 2009.

[25] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proceedings of International Logic Programming Conference and Symposium*, pages 1070–1080. MIT Press, 1988.

[26] Michael R. Genesereth and Richard E. Fikes. Knowledge interchange format, version 3.0 reference manual. Technical Report Logic-92-1, Stanford University, Stanford, CA, USA, 1992.

[27] Michael R. Genesereth, Nathaniel Love, and Barney Pell. General game playing: Overview of the aaai competition. *AI Magazine*, 26 (2):62–72, 2005.

[28] Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain, Hudson Turner, and Joohyung Lee Vladimir Lifschitz. Nonmonotonic causal theories. *Artificial Intelligence*, 153:2004, 2004.

[29] Eric Gradel, Wolfgang Thomas, and Thomas Wilke, editors. *Alternating Tree Automata and Parity Games*, 2002.

[30] Zong Hui Guo. Design and develop a general game playing system using answer set programming. Technical report, The University of Bath, 2008.

[31] Kristoffer Arnsfelt Hansen, Michal Koucky, and Peter Bro Miltersen. Winning concurrent reachability games requires doubly-exponential patience. In *Proceedings of the 2009 24th Annual IEEE Symposium on Logic In Computer Science*, pages 332–341, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3746-7.

[32] Kristoffer Arnsfelt Hansen, Rasmus Ibsen-Jensen, and Peter Bro Miltersen. The complexity of solving reachability games using value and strategy iteration. *CoRR*, abs/1007.1812, 2010.

[33] Roland A Howard. *Dynamic Programming and Markov Processes*, volume 3. MIT Press and Wiley, 1960.

[34] World Book Inc. *The World Book Encyclopedia.* Addison-Wesley, Chicago, Illinois, USA, 10th edition, 2001.

[35] Neil D. Jones. Space-bounded reducibility among combinatorial problems. *J. Comput. Syst. Sci.*, 11:68–85, August 1975.

[36] Henry Kautz and Bart Selman. Planning as satisfiability. In *Proceedings of the 10th European conference on Artificial intelligence*, ECAI 92, pages 359–363, New York, NY, USA, 1992. John Wiley & Sons, Inc. ISBN 0-471-93608-1.

[37] Robert M. Keller. Formal verification of parallel programs. *Commun. ACM*, 19:371–384, July 1976. ISSN 0001-0782.

[38] Arne König, Christian Drescher, Max Ostrowski, and Torsten Grote. General game player using answer set programming. Technical report, University of Postdam, Postdam, Germany, 2007.

[39] Saul A. Kripke. Semantical considerations on modal logic. *Acta Philosophica Fennica*, 16:83–94, 1963.

[40] Luca de Alfaro Krishnendu Chatterjee and Thomas A. Henzinger. Strategy improvement for concurrent reachability games. In *QEST 06*, September 2006.

[41] Gregory Kuhlmann, Kurt Dresner, and Peter Stone. Automatic heuristic construction in a complete general game player. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, pages 1457–62, 2006.

[42] Kenneth Kunen. Some remarks on the completed database. In *ICLP/SLP*, pages 978–992, 1988.

[43] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The dlv system for knowledge representation and reasoning. *ACM Trans. Comput. Logic*, 7:499–562, July 2006. ISSN 1529-3785.

[44] Nancy G. Leveson. An investigation of the therac-25 accidents. *IEEE Computer*, 26:18–41, 1993.

[45] Nathaniel Love, Michael Genesereth, and Timothy Hinrichs. General game playing: Game description language specification. Technical Report LG-2006-01, Stanford University, Stanford, CA, 2006.

[46] Yiu-Chung Mang. *Games in open systems verification and synthesis.* PhD thesis, Oxford University England, 2002. AAI3063469.

[47] Ilkka Niemelä. Answer set programming without unstratified negation. In *ICLP '08: Proceedings of the 24th International Conference on Logic Programming*, pages 88–92, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-89981-5.

[48] Ilkka Niemelä and Patrik Simons. Smodels - an implementation of the stable model and well-founded semantics for normal lp. In *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning*, LPNMR '97, pages 421–430, London, UK, 1997. Springer-Verlag. ISBN 3-540-63255-7.

[49] Ulf Nilsson and Jan Maluszynski. *Logic, programming and Prolog.* Wiley, 1990. ISBN 978-0-471-92625-2.

[50] Teodor Przymusinski. Well-founded and stationary models of logic programs. *Annals of Mathematics and Artificial Intelligence*, 12: 141–187, 1994.

[51] Teodor C. Przymusinski. Perfect model semantics. In *ICLP/SLP*, pages 1081–1096, 1988.

[52] Teodor C. Przymusinski. Every logic program has a natural stratification and an iterated least fixed point model (extended abstract). In *Proceedings of the Eighth Symposium of Principles of Database Systems*, pages 11–21. ACM Press, 1989.

[53] Teodor C. Przymusinski. Stable semantics for disjunctive programs. *New Generation Computing*, 9:401–424, 1991.

[54] Stuart J. Russell, Peter Norvig, John F. Candy, Jitendra M. Malik, and Douglas D. Edwards. *Artificial intelligence: a modern approach.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996. ISBN 0-13-103805-2.

[55] Stephan Schiffel and Michael Thielscher. Reconciling situation calculus and fluent calculus. In *AAAI'06: Proceedings of the 21st national conference on Artificial intelligence*, pages 287–292. AAAI Press, 2006. ISBN 978-1-57735-281-5.

[56] Stephan Schiffel and Michael Thielscher. Fluxplayer: a successful general game player. In *AAAI'07: Proceedings of the 22nd national conference on Artificial intelligence*, pages 1191–1196. AAAI Press, 2007. ISBN 978-1-57735-323-2.

[57] Stephan Schiffel and Michael Thielscher. Automated theorem proving for general game playing. In *IJCAI 09, Proceedings of the 21st international jont conference on Artifical intelligence*, pages 911–916, 2009.

[58] Stephan Schiffel and Michael Thielscher. A multiagent semantics for the game description language. In J. Filipe, A. Fred, and B. Sharp, editors, *International Conference on Agents and Artificial Intelligence (ICAART)*, pages 44–55, Porto, Portugal, 2009. Springer.

[59] Lloyd S. Shapley. Stochastic games. In *PNAS*, pages 1095–1100, USA, 1953. National Academy of Sciences.

[60] John C. Shepherdson. Negation in logic programming. *Foundations of Deductive Databases and logic programming*, pages 19–88, 1988.

[61] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285–309, 1955.

[62] Michael Thielscher. Answer set programming for single-player games in general game playing. In *Proceedings of the 25th International Conference on Logic Programming*, ICLP 09, pages 327–341, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-02845-8.

[63] Michael Thielscher and Sebastian Voigt. A temporal proof system for general game playing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Atlanta, 2010. AAAI Press.

[64] Wiebe Van, Der Hoek, Ji Ruan, and Michael Wooldridge. Strategy logics and the game description language. In *Workshop on Logic, Rationality and Interaction*, Beijing, China, 2007.

[65] Wiebe van der Hoek, Wojciech Jamroga, and Michael Wooldridge. A logic for strategic reasoning. In *AAMAS05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 157–164, New York, NY, USA, 2005. ACM. ISBN 1-59593-093-0.

[66] M. H. Van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *J. ACM*, 23:733–742, October 1976. ISSN 0004-5411.

[67] Martin H. Van Emden and Robert A. Kowalski. The semantics of predicate logic as a programming language. *J. ACM*, 23(4): 733–742, 1976. ISSN 0004-5411.

DECLARATION

I declare that the work in my master thesis was carried out in accordance with the regulations of the Technical University of Dresden, Germany and no part of the thesis has been submitted for any other degree. The work is original except where contributions of others are involved, every effort is made to indicate this clearly with due reference to the literature.

Dresden, Germany 19.09.2011

M. Fareed Arif