Embedding a Card Game Language into a General Game Playing Language

Jakub KOWALSKI [a],
[a] *Institute of Computer Science, University of Wrocław, Poland*

Abstract. We make a link between a specialized context free language expressing the rules of variety of card games, called CGDL, and the most known general-purpose game description language GDL-II. We present a systematic translation from CGDL to GDL-II, prove that the translation is correct, and analyze the complexity of resulting code in both theoretical and empirical way.

Keywords. General Game Playing, knowledge representation, Game Description Language, Card Game Description Language

1. Introduction

Artificial intelligence was always using games as test problems and demonstration of its advancement. Defeating human supremacy in popular games such as chess (*Deep Blue*), checkers (*Chinook*) or Jeopardy (*Watson*) were great achievements of AI. However, to meet the basic purpose of AI – solving general problems, a different approach should be taken. Instead of improving specialized programs which are able to play only one game, the general approach to game playing consists of developing programs which can play every game from some wide game class defined by a formal description language.

The most popular approaches in this field are related to *General Game Playing* (GGP) competition [5] and a special first order logic language based on KIF called *Game Description Language* (GDL) [10]. It has enough power do describe all turn-based, finite and deterministic n-player games with full information. Playing a game given by such a description requires not only developing a move choosing algorithm, but also implementing a reasoning approach to understand the game rules in the sense of computing legal moves, computing the state update function, and computing the goal function. Many approaches were created in this field including implementations of game playing mechanisms [2,6,7] and improving effectiveness of reasoning engines [1,8,9]. Recently, an extension of GDL called GDL-II (from *GDL with Incomplete-Information*) was proposed [15]. This language removes some restrictions of GDL and allows to describe also nondeterministic games with hidden information, but requires developing new playing techniques [13].

A card game description language developed recently in [3,4], which we call for brevity CGDL, has a unique feature – it allows one to use genetic programming to evolve game rules and so, to create new games. CGDL is a high level language with a lot of domain specific commands, to describe n-player, standard deck card games with possibility of bets (like in poker). In this paper we define a direct translation from CGDL lan-

guage into GDL-II. Then, we compare both languages as representatives of two different approaches to game description: very general one, but with difficult semantic influencing efficiency of playing algorithms, and narrow one with compact and flexible description. As GDL-II was compared with other game classes to ensure it has enough expressive power [11,12,16], no translation between other game description languages focused on practical gaming aspects is presented. We implement our translation and compare the complexity of CGDL and GDL-II both theoretically and empirically. We also distinguish features, which can possibly be transferred from CGDL into GDL/GDL-II to improve this, currently the most widely used, general game description language.

2. Preliminaries

2.1. GDL-II

GDL-II [15] is, as its predecessor GDL, strictly declarative language using logic programming-like syntax very similar to Prolog. It can describe any finite, synchronous, turn-based, n-player game. Every game description contains the declaration of players roles, the initial game state, the legal moves, the state transition function with players' percepts, the terminating conditions, and the declaration of goal function.

Language does not provide any predefined functions including arithmetic expressions or game-domain specific structures such as a board or a card deck. Every function and declaration must be defined explicitly from scratch, and the only keywords used to define game are (symbols beginning with `?` are variables):

`(role ?r)`	`?r` is a player
`random`	random player (Nature, casino, ...)
`(init ?f)`	fact `?f` is true in the initial state
`(true ?f)`	fact `?f` is true in the current state
`(legal ?r ?a)`	in the current state `?r` can perform action `?a`
`(does ?r ?a)`	`?r` performed action `?a` in the previous state
`(next ?f)`	`?f` will be true in the next state
`(sees ?r ?p)`	player `?r` will perceive `?p` in the next state
`terminal`	current state is terminal
`(goal ?r ?n)`	player `?r` score is `?n`

To be considered as *valid*, a GDL-II game specification must be *stratified* and *allowed*. This, and other syntactic restrictions ensures that the game have a unique standard model with only a finite number of true positive instances, so all deductions in Definition 1 are finite and decidable. For details we must refer to [10].

Let G be a valid GDL-II game description. It contains a finite number of function symbols and constants, which determines the set of possible ground terms Σ. Although Σ can be infinite, syntactic restrictions ensure that all sets needed to compute game flow (roles, legal moves, reachable states, etc.) are finite subsets of Σ [10]. Let $S = \{f_1, \ldots, f_k\}$ be a state of the game, denoted as the set of predicates which are true in the current position. Then we define a *base state* as $S^{\texttt{true}} \stackrel{\text{def}}{=} \{\texttt{(true } f_1\texttt{)}, \ldots, \texttt{(true } f_k\texttt{)}\}$. Let us also denote the joint move $A^{\texttt{does}} \stackrel{\text{def}}{=} \{\texttt{(does } r_1\ a_1\texttt{)}, \ldots, \texttt{(does } r_n\ a_n\texttt{)}\}$ if players $r_1, \ldots, r_n$ took actions $a_1, \ldots, a_n$. We can now introduce

Definition 1 *[15] The semantics of a valid GDL-II n player game specification G with a set of ground terms Σ is given by a state transition system composed as follows.*

- $R = \{r \in \Sigma : G \models \texttt{(role } r\texttt{)}\}$ *(player names);*
- $s_0 = \{f \in \Sigma : G \models \texttt{(init } f\texttt{)}\}$ *(initial state);*
- $t = \{S \in 2^{\Sigma} : G \cup S^{true} \models \texttt{terminal}\}$ *(terminal states);*
- $l = \{(r, a, S) : G \cup S^{true} \models \texttt{(legal } r\ a\texttt{)}\}$*, for all* $r \in R$*,* $a \in \Sigma$ *and* $S \in 2^{\Sigma}$*;*
- $u(A, S) = \{f : G \cup A^{does} \cup S^{true} \models \texttt{(next } f\texttt{)}\}$*, for all joint moves* $A : (R \mapsto \Sigma)$ *and states* $S \in 2^{\Sigma}$ *(state update);*
- $\mathcal{I} = \{(r, A, S, p) : G \cup A^{does} \cup S^{true} \models \texttt{(sees } r\ p\texttt{)}\}$*, for all* $r \in R \setminus \{\texttt{random}\}$*,* $A : (R \mapsto \Sigma)$*,* $S \in 2^{\Sigma}$ *and* $p \in \Sigma$ *(players' percepts);*
- $g = \{(r, n, S) : G \cup S^{true} \models \texttt{(goal } r\ n\texttt{)}\}$*,* $r \in R$*,* $n \in \{0, \ldots, 100\}$*,* $S \in 2^{\Sigma}$.

The execution model works as follows. Starting from the initial state s_0, in every state S every player $r \in R$ selects one legal action a such that $(r, a, S) \in l$. The `random` player choose his moves randomly with uniform probability. The joint move is applied to the state update function $u(A, S)$ to obtain a new state S'. In S', every role $r \in R \setminus \{\texttt{random}\}$ perceives every p that satisfies $(r, A, S, p) \in \mathcal{I}$. If the current state is terminal, i.e. $S \in t$, then every player obtains a score by relation $(r, n, S) \in g$ and the game ends.

The partial GDL-II game description is set as an example in Listing 2. For a more detailed language specification and full games examples we refer to [15,16].

2.2. CGDL

CGDL introduced in [4] is a context free language designed to define a rich subset of possible card games and allow to perform genetic operations to create new or evolve existing games [3]. The language domain is narrowed to n player, standard deck card games with a possibility of coin bets. All language constructions are strictly domain-dependent and use concepts of a card, number, suit, token, etc. These concepts, same as arithmetic and boolean operators, are defined a priori and used without explicit declaration.

A valid CGDL game is defined as follows. P is the number of players. Every player i has his private hand location (named Hi) and two areas for placing coins: private $Ki0$ with player's coins, and public $Ki1$ for placing bets. T is the number of virtual table locations, where cards can be placed face up. The set of game rules is organized into sequentially ordered stages, containing rules. Every stage is played in a round-robin order until all players are *out* of the game or decide to end the current stage (status set to *done*).

Every rule has a form of $modifier$ `if` $condition$ `then` $action$. The current player can perform any *action* from the set of current stage rules only if the rule *condition* is satisfied and the *modifier* limitations are met. Possible rule *modifiers* are: *computer* – must be played by the computer at the beginning of a stage, *mandatory* – must be played by a player at the beginning of a stage (after computer rules applied), *once* – can be applied only once, *optional* – no restrictions. A partial list of possible *conditions* contains

- λ – no condition to satisfy, always true;
- `sum`, LA, R, LB – sums values of cards in both locations LA and LB, evaluates to true if the restriction $R \in \{<, >, =, \leq, \geq, \lambda\}$ is satisfied;
- `tokens`, KA, R, KB – compares number of tokens in KA and KB using R;
- `have`, C – check if the player's hand contains given card combination C, e.g. the king of hearts, three diamonds and one heart, etc.

Set of available *actions* is a superset of

- `pifr`, LA, A, F – draw a given amount A of cards from a location LA to player's hand, cards are openly visible if face F is *up*;
- `bet`, R, KA – bet an amount of tokens which, when compared to the number of tokens in the location KA, satisfies the restriction R;
- `done` / `out` – the player status is set to *done*/*out*;
- `win` – the player instantly wins the game;
- `give`, P, A – *computer* only, give A tokens to the set of players P;
- `deal`, P, A – *computer*, deal A cards from the deck to players P.

Additional special abbreviations are: HA – hands of all players, KA bets of all players, HX the hand of the current player, KX bets of the current player, `<allplayers>` rule is multiplied for all players. Rules can also define a value mapping for every card and some combination of cards (called plays). Game is over when all players are *out*, some player performs `win` action or the last stage is over. Player's score is calculated by points for every possessed token (`t`), card (`c`) and "not `out`" bonus (`s`). If player ends by choosing `win` action he got 100 and other 0. As an example, rules in Listing 1 describe a codification of game *Blackjack*. For more detailed language specification we must refer to [3,4].

3. Translation

In this section we present details of our translation. We constructed the function $\mathcal{F}$ which takes a CGDL game description $\mathcal{G}$ and returns the same game encoded in GDL-II. As the CGDL language is created by a context-free grammar, the construction is grammar based. For every game description subtree, we compute all GDL rules necessary to encode that subtree. Afterwards we remove duplicate definitions of the predicates.

The specification of the translation is provided in as much detail as we can in such limited space. As an example we translate CGDL codification of the game *Blackjack* from Listing 1 to equivalent GDL-II rules partially presented in Listing 2. References to line numbers (if not stated otherwise) refers to Listing 2.

Listing 1: CGDL codification of the game *Blackjack*.

```
1  [SETTINGS] P=3, T=0
2  [STAGES]
3    Stage 0
4  COMPUTER deal, <allplayers>, 2
5  COMPUTER give, <allplayers>, 99
6    Stage 1
7  MANDATORY if λ then bet, λ , λ
8    Stage 2
9  OPTIONAL if λ then pifr, D, 1, up
10 OPTIONAL if λ then done
11   Stage 3
12 MANDATORY if sum, HX, >, 21  then out
13 MANDATORY if sum, HX, <=, 21  then done
14   Stage 4
15 MANDATORY if sum, HX, >, HA  then gain, KA
16 [RANKING] 2:2, ..., King:10, Ace:11, Ace:1
17 [POINTS] t=1, c=0, s=0
```

Listing 2: Partial GDL-II code of the translated CGDL game *Blackjack* (in infix notation).

```
 1(role random) (role player1) ... (role player3)
...
26(init (Stage ShuffleDeck COMPUTER))
27(init (Shuffled Top))
28((init (UnShuffled ?c)) ⇐ (card ?c ?num ?suit))
29(init (ActionAvailable 0 s0a0 random COMPUTER))...
...
161((next (Token ?l2 ?n3))
162  ⇐ (true (Token ?l2 ?n1)) ∧ (movecoin ?n2 ?l1 ?l2) ∧ (asum ?n1 ?n2 ?n3))
163((next (Token ?l1 ?n3))
164  ⇐ (true (Token ?l1 ?n1)) ∧ (movecoin ?n2 ?l1 ?l2) (asub ?n1 ?n2 ?n3))
165((next (ActionAvailable ?stage ?id1 ?p ?type))
166  ⇐ (true (ActionAvailable ?stage ?id1 ?p ?type))
167  ∧ (does ?player (action ?id2 ?vis ?cond ?act)) ∧ (distinct ?id1 ?id2))
...
314((legal ?player ?act) ⇐ (tmplegal ?player ?act))
315((tmplegal ?player (action s4a0 visible (sum HX gt HA) (gain KA)))
316  ⇐ (true (Stage 4 MANDATORY)) ∧ (true (CurrentPlayer ?player))
317  ∧ (true (ActionAvailable 4 s4a0 ?player MANDATORY))
318  ∧ (not (true (PlayerStatus ?player aDONE)))
319  ∧ (not (true (PlayerStatus ?player aOUT)))
320  ∧ (handlocation ?player ?hand) ∧ (rsum ?hand ?n) ∧ (handlocation ?p1 ?h1)
321  ∧ (rsum ?h1 ?n1) ∧ (handlocation ?p2 ?h2) ∧ (rsum ?h2 ?n2)
322  ∧ (distinct ?p1 ?player) ∧ (distinct ?p2 ?player) ∧ (distinct ?p2 ?p1)
323  ∧ (or (bgt ?n ?n1) (true (PlayerStatus ?p1 aOUT)))
324  ∧ (or (bgt ?n ?n2) (true (PlayerStatus ?p2 aOUT))))
...
432((movecoin ?n ?bloc1 ?hloc) ⇐ (betlocation ?player ?hloc ?bloc)
433  ∧ (does ?player (action ?id ?vi ?cond (gain KA)))
434  ∧ (betlocation ?p ?hloc1 ?bloc1) (rKA ?n))
...
471((sees ?player (deltacoins ?n ?loc1 ?loc2)) ⇐ (movecoin ?n ?loc1 ?loc2))
472  ∧ (does ?p (action ?id visible ?cond ?action))
...
487(terminal ⇐ endstage ∧ (true (Stage ?n ?t))
488  ∧ (_stagesorder (Stage ?n ?t) ∧ (Stage EndGame none)))
489((goal ?player 100) ⇐ (true (Won ?player)))
490((goal ?player 0) ⇐ (true (Won ?p))∧(role ?player)∧(not(true (Won ?player))))
...
557((rKA ?s3) ⇐ (true (Token K11 ?n1)) ∧ (true (Token K21 ?n2))
558  ∧ (true (Token K31 ?n3)) ∧(asum ?n1 ?n2 ?s2) ∧ (asum ?s2 ?n3 ?s3))
559((rsum ?loc ?s12) ⇐ (location ?loc) ∧ (numberofcards ?loc 2)
560  ∧ (hold2cards ?loc ?c1 ?c2) ∧ (card ?c1 ?num1 ?suit1)
561  ∧ (value ?num1 ?val1) ∧ (card ?c2 ?num2 ?suit2)
562  ∧ (value ?num2 ?val2) ∧ (asum 0 ?val1 ?s1) ∧ (asum ?s1 ?val2 ?s12))
...
694(location D) ... (location H3)
695(handlocation player1 H1) ... (handlocation player3 H3)
696(betlocation player1 K10 K11) ... (betlocation player3 K30 K31)
697(card 2OfHearts 2 Hearts) ... (card AceOfSpades Ace Spades)
698(value 2 2) ... (value Ace 11) (value Ace 1)
699(stagesorder (Stage ShuffleDeck COMPUTER) (Stage 0 COMPUTER)) ...
700  (stagesorder (Stage 4 MANDATORY) (Stage EndGame none))
...
819(aplus1 0 1) ... (aplus1 100 101)
820((asum ?n 0 ?n) ⇐ (aplus1 ?n ?m))
821((asum ?n1 ?n3 ?n5) ⇐ (aplus1 ?n2 ?n3)
822  ∧ (aplus1 ?n4 ?n5) ∧ (asum ?n1 ?n2 ?n4))
823((bleq ?n ?n) ⇐ (aplus1 ?n ?m))
824((bleq ?n1 ?n3) ⇐ (aplus1 ?n1 ?n2) ∧ (bleq ?n2 ?n3))
```

Constants define relations which hold throughout the entire game (line 1 and 694). Predicate `role ?role` declares set of players `player1`, ...`player`P, `random`; `location ?loc` defines set of possible card locations: `D` for the deck, `H1`, ..., `H`P for players' hands and `T0`, ..., `T`(T–1) for table locations; `handlocation ?player ?hand` maps `player`i to his hand location `H`i. Predicate `tokenlocation ?loc` defines all token locations `K10`, ..., `K`P`0`, `K11`, ..., `K`P`1` and `betlocation ?player ?privloc ?betloc` maps these locations from player `player`i to private token location `K`i`0` and bet location `K`i`1`. List of all cards is stored in `card ?card ?number ?suit`. The card value mapping is stored in predicate `value ?card ?n`. Round-robin ordering of P players is simply defined as `playersorder ?prev ?next` relation. Predicate `stagesorder ?prev ?next` is constructed based on ordering detected in CGDL game codification.

Definition 2 *The game state is the minimal set of data necessary to distinguish that two game positions are different. CGDL game state consists of: the contents of the deck and all table and token locations; the number of current stage and player; the status of every player and the information about available computer/mandatory/once actions.*

A game state is covered by a constant number of GDL-II predicates forming $S^{\texttt{true}}$ sets and storing information necessary to encode the CGDL state according to Definition 2.

- `Stage ?id ?type` – identifier of the current stage and a type of the sub-stage containing information about the allowed actions type (*computer*, *mandatory*, *optional*); one such predicate is true at the time;
- `ActionAvailable ?stagenumber ?actionID ?player ?type` – stores all actions which can be performed by the players, if a non-optional action was made it is removed from this set and cannot be used again;
- `Token ?location ?amount` – for every `tokenlocation` stores the number of tokens in this location;
- `Table ?location ?card` – for every `tablelocation` except the deck stores the cards in this location;
- `Deck ?nextcard ?prevcard` – contains cards in the deck arranged in an order (special constant `Top` marks top of the deck);
- `CurrentPlayer ?player` – identifies the current player if it is not the dealer's turn, maximum one such predicate is true at the time;
- `PlayerStatus ?player ?status` – for every player remembers his status if necessary; status can be `aDONE` if the player decided to end the current stage, `aOUT` if he is out of the game or `mDONE` if he has no mandatory moves but is not *done*;
- `Won ?player` – true if some player performed the `win` action;
- `UnShuffled ?card` – a special predicate used at the beginning of the game to remember cards which are not yet shuffled into the deck;
- `Shuffled ?card` – a special game-beginning predicate to remember the last card chosen by the dealer to be placed at the bottom of the deck.

Definition 3 *A state is called **technical** in one of the following cases: the predicate `Stage ShuffleDeck COMPUTER` is true – which means that the virtual dealer prepares random ordering of cards in the deck; or the `CurrentPlayer` status is "out", "done" or he has no actions with fulfilled conditions but it is his turn in round-robin ordering – then every player makes the `NOOP` move and the `CurrentPlayer` shifts to the next player in order (which may lead to a next technical state).*

The initial $\mathcal{F}(\mathcal{G})$ game state s_0 (line 26) is technical, because of necessity to randomize the card deck. Every game begins with `(Stage ShuffleDeck COMPUTER)` $\in S^{\texttt{true}}$ and all cards identifiers marked as `UnShuffled`. For the next 52 turns the `random` player choose every still `UnShuffled` card with the uniform probability and put on the bottom of the actual deck. After this, `Stage` changes to the first stage from the original $\mathcal{G}$ game. Also possible players' actions are put into the `AvailableAction` predicate.

Managing legal actions can be divided into several cases. When `CurrentPlayer` is set and there is some non-`COMPUTER Stage`, the player can choose legal action from existing `tmplegal ?player (action ?id ?visibility ?condition ?action)` relations. Such relation corresponds to exactly one rule from $\mathcal{G}$. Variables `?condition` and `?action` matches CGDL rule condition and action; `?id` is an artificial identifier matching the id from `AvailableActions` relation, and `?visibility` serves to determine $\mathcal{I}$ function.

In our example, the rule from CGDL codification line 15 is translated into the `tmplegal` rule in GDL-II code line 315. Initial queries are checking stage, action availability and player's status. Then queries matching *condition* are set. In our example these are restricting sum of values of player's cards against the values of all other players cards.

Another case occurs when no `tmplegal` relation is true, due to not fulfilled conditions or exhaustion of `AvailableActions`. Then player can make only special `NOOP` move which does not change the game state. This move is also used in the following cases: for player which as not marked as `CurrentPlayers`; for player who is marked as current but he is actually *done* or *out* and for all players when there is a `COMPUTER` type stage.

For managing changes in cards and tokens possession two special predicates were introduced: `movecard` and `movecoin`. They are filling the gap between performed actions $A^{\texttt{does}}$ and state update function u. As the main idea between both predicates is similar we present here only a `movecoin` example (line 432). If relation `movecoin ?n ?from ?to` holds, this means that `?n` coins are added to `?to` coin location and subtracted from `?from` location. Multiple such relations can hold at the same time, but then their `?n` and `?to` arguments are the same. CGDL limitations implies that moving coins from multiple locations forces them to be empty, so in such cases `?n` is always set to sum of tokens in all `?from` locations. It is safe due to natural number arithmetic (subtracting from 0 is 0).

Updating the current state given players' moves, i.e. defining u function is the most complex part of the translation which, due to the limited space, we cannot describe in details (partial `next` code is shown in line 161). Every base predicate has its own set of updating rules, mostly using additional helper predicates. Sketch of the mechanics looks as follows. End of a stage occurs when all players have adequate statuses (`aOUT`, `aDONE` or `mDONE`), there is no player with any `ActionAvailable` left for the stage, or this is last turn of `ShuffleDeck` stage. If `endstage` holds, stage changes to the next stage in `stagesorder`. Similarly `CurrentPlayer` changes when `endplayer` holds. This depends on the player's actions performed, status and availability of the current stage actions.

Content of `Table`, `Deck` and `Token` is updated based on `movecard` and `movecoin` semantics. This requires several cases to examine, especially concerning taking cards from the deck without destroying its structure. If in the last turn, the player performed non-`OPTIONAL` action with some identifier, it is removed from the `ActionAvailable` set. Updating `Shuffled` and `UnShuffled` predicates takes place during the `ShuffleDeck` stage. If `does random (?shuffle ?card)` holds, then `?card` is subtracted from `UnShuffled` set and remembered as last `Shuffled` card. The predicate `Won` becomes true when the `win` action was performed.

Simulating CGDL requires natural number arithmetic for the numbers not greater then 100 (upper bound for a goal value). If some card value or play value extends that number, we can use the optional translation parameter to increase this maximum. Every created GDL-II description contains the following arithmetic and boolean functions: `aplus1`, `asum`, `asub`, `amult`, `blt`, `bgt`, `bleq`, `bgeq`, `beq`, `bneq` (line 819).

Rules defining `tmplegal` contain queries to helper predicates encoding CGDL restrictions and conditions. For example `rKA ?n` holds if `?n` is the cumulative bet of all players (line 557) and `rsum ?loc ?n` holds if values of cards in `?loc` sum up to `?n`. The function `rsum` is particularly complex and requires other helper predicates: `numberofcards ?loc ?n` which holds if in `?loc` there exactly `?n` cards, `holdcards_arity ?loc ?n` satisfied when `?loc` contains at least `?n` cards, and the predicate family `hold`n`cards ?loc ?card1 ... ?card`n which hold if there are n different cards in `?loc`. The complexity of the solution rises from a need of reasoning about the number of satisfied predicates. Example of `rsum` rule for $n = 2$ is shown in line 559.

The predicate `sees ?player ?percept` defines $\mathcal{I}$ relation, i.e. predicates perceptible by given player. Every player has a full knowledge about his hand, his private coin location, the current stage, current player and all players' statuses. Table locations, bet locations and last performed action identifier are visible to all players. Details of the action (e.g. what cards player took from the deck) are perceived only by the player who made the action or by everyone if action visibility is set to `visible` (line 471).

Definition of the `terminal` relation depends on several rules checking if: all players are *out*, some player made `win` action or `Stage EndGame none` is reached (line 487). Computing the g function is divided into two cases. If player won using the `win` action, he got 100 and all other players (including `random`) got 0. Otherwise the player's score is computed according to the CGDL game specification using `t`, `c`, `s` values.

4. Translation Properties

We state the main properties of the translation, concerning its correctness and complexity.

Definition 4 *Let S be a state of CGDL game $\mathcal{G}$. Then a state S' of game $\mathcal{F}(\mathcal{G})$ is called **corresponding**, i.e. $S \doteq S'$ if: both states have the same content and ordering of deck and every hand location, table location and token location; the actual number and type of stage; the current player; player's statuses and the set of available actions.*

Theorem 1 *For every CGDL game $\mathcal{G}$ presented translation $\mathcal{F}$ satisfies the following.*

1. *The game $\mathcal{F}(\mathcal{G})$ meet all syntactic requirements of valid GDL-II description.*
2. *The irst non-technical state of $\mathcal{F}(\mathcal{G})$ is corresponding to the first state of $\mathcal{G}$ assuming identical deck ordering.*
3. *For every non-technical game states $S \doteq S'$ there is an isomorphism between joint legal actions from both states.*
4. *For every non-technical game states $S \doteq S'$ and joint actions $A \doteq A'$, for all sequences of joint moves $\langle A', A'_2, \ldots, A'_k \rangle$ such that $S_i = u(A'_i, \ldots u(A', S') \ldots)$ and S_k is non-technical but for all $i < k$, S_i is technical, S_k is corresponding to state S after applying actions A. Such sequence always exists.*
5. *If $S \doteq S'$, S is terminal iff. $S' \in t$, goal values match for corresponding players.*

Table 1. Results of example games transformation. Visualize dependence between complexity of CGDL description (number of players, table locations, stages and rules) and resulting GDL-II code. As measurement we took number of rules and predicates used to express the game. The number of predicates and rules for base predicates are shown separately.

Game	CGDL code				GDL-II code			
					predicates		rules	
	P	T	stages	rules	base	all	base	all
Uno	2	1	2	7	10	61	38	245
Uno	3	1	2	8	10	61	39	254
Uno	3	2	2	8	10	61	39	256
Blackjack	3	0	5	12	10	63	43	361
Blackjack	3	1	5	12	10	63	43	363
Poker	3	2	13	30	10	68	61	426
Poker	4	2	13	32	10	68	63	437

Proofs of theorems are omitted due to space reasons.
We implemented a program which applies the translation function for given CGDL game description. To measure practical complexity of resulting code we provided a series of experiments using *Poker*, *Blackjack* and *Uno* games from [4] (with slightly different changes).The result of the experiments are shown in Table 1. Sizes of translated games gives a picture of complexity of their rules, which affects speed of computing GDL game states. The data also show that a number of predicates is independent on the number of players and table locations, a number of base predicates rules depends linear on the number of players and both these values have influence on overall number of rules. We state that theoretical complexity of a translated game is described by

Theorem 2 *Let $\mathcal{G}$ be a P player CGDL game description of the length N (where length is the sum of the number of stages, rules, and card/plays value mapping entries) with T table locations. Then the number of rules in the GDL-II game $\mathcal{F}(\mathcal{G})$ is $O(P + T + N)$ and the number of predicates can be bounded by a constant.*

5. Summary

The quality of solutions for General Game Playing problems heavily depends on a language which describes a game. Although the target should be to describe as many games as possible, very general languages causes two major problems. First is that providing a good playing algorithm is much harder if the type of a game is unknown. Second, that understanding and maintaining the game description is also far more complicated. Other extreme case is a language which can describe only certain types of strictly declared games, but it is high level and uses domain-dependent constructions. In this case maintaining a game description and implementing better playing algorithms is simpler.

In this paper we studied the relation between both of these approaches. On the one hand we took the most popular and the most general first order logic GDL-II language, which can describe any finite, turn-based, n-player game. On the other hand we took recently developed Card Game Description Language which is a high level language to describe card games with bets and allows a genetic manipulation on the game structure.

We constructed a translation from CGDL language into GDL-II, described details of this translation, proved its correctness and checked its complexity both theoretically and empirically. Although the size of translated game description is linear (in the sense of game rules), a complexity of simulating the game basing on pure GDL engine without compilation [9] or calling external code [14] can be computationally too hard to be performed in reasonable time, due to complicated form of queries. Our translation can be used as a benchmark tool for improving GDL players by comparison with CGDL playing algorithms. If some successful solutions could be transferred to more general approach it would be a step to reduce the gap between GDL-based and game-specific reasoners.

An interesting feature is a possibility of performing genetic operations on CGDL game code. Although CGDL was designed specifically for this case, we think it is also possible to develop this feature to GDL. GDL is a complicated language in terms of validity and semantics, but syntax rules are simple enough to make it feasible. In fact, artificially evolved GDL games could have interesting influence on creating GGP agents. They should from now on be able to play really *any* game, even codified in strange style and without common sense, not only well behaved adaptations of human games.

References

[1] Y. Björnsson and S. Schiffel. Comparison of GDL Reasoners. In *Proceedings of the IJCAI-13 Workshop on General Game Playing (GIGA'13)*, 2013.

[2] H. Finnsson and Y. Bjornsson. Simulation-based Approach to General Game Playing. In *AAAI*. AAAI Press, 2008.

[3] J. Font, T. Mahlmann, D. Manrique, and J. Togelius. Towards the automatic generation of card games through grammar-guided genetic programming. In *International Conference on the Foundations of Digital Games (FDG 2013)*, pages 360–363, 2013.

[4] J. M. Font, T. Mahlmann, D. Manrique, and J. Togelius. A Card Game Description Language. In *Applications of Evolutionary Computation*, volume 7835 of *LNCS*, pages 254–263. Springer, 2013.

[5] M. Genesereth, N. Love, and B. Pell. General game playing: Overview of the AAAI competition. *AI Magazine*, 26:62–72, 2005.

[6] S. Haufe, D. Michulke, S. Schiffel, and M. Thielscher. Knowledge-Based General Game Playing. *KI*, 25(1):25–33, 2011.

[7] P. Kissmann and S. Edelkamp. Symbolic Classification of General Multi-Player Games. In *European Conference on Artificial Intelligence (ECAI)*, volume 178 of *FAIA*, pages 905–906. IOS Press, 2008.

[8] P. Kissmann and S. Edelkamp. Instantiating General Games Using Prolog or Dependency Graphs. In *KI 2010: Advances in Artificial Intelligence*, volume 6359 of *LNCS*, pages 255–262. Springer, 2010.

[9] J. Kowalski and M. Szykuła. Game Description Language Compiler Construction. In *AI 2013: Advances in Artificial Intelligence*, volume 8272 of *LNCS*, pages 234–245. Springer, 2013.

[10] N. Love, T. Hinrichs, D. Haley, E. Schkufza, and M. Genesereth. General Game Playing: Game Description Language Specification. Technical report, Stanford Logic Group, 2008.

[11] J. Ruan and M. Thielscher. On the Comparative Expressiveness of Epistemic Models and GDL-II. In *Proceedings of the IJCAI-11 Workshop on General Game Playing (GIGA'11)*, 2011.

[12] S. Schiffel and M. Thielscher. Representing and Reasoning About the Rules of General Games With Imperfect Information. *Journal of Artificial Intelligence Research*, 49:171–206, 2014.

[13] M. Schofield, T. Cerexhe, and M. Thielscher. HyperPlay: A Solution to General Game Playing with Imperfect Information. In *AAAI Conference on Artificial Intelligence*. AAAI Press, 2012.

[14] X. Sheng and D. Thuente. Extending the General Game Playing Framework to Other Languages. In *Proceedings of the IJCAI-11 Workshop on General Game Playing (GIGA'11)*, 2011.

[15] M. Thielscher. A General Game Description Language for Incomplete Information Games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 994–999. AAAI Press, 2010.

[16] M. Thielscher. The General Game Playing Description Language is Universal. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1107–1112. AAAI Press, 2011.