A General Approach of Game Description Decomposition for General Game Playing

Aline Hufschmitt(✉), Jean-Noël Vittaut, and Jean Méhat

LIASD - University of Paris 8, Saint-Denis, France
{alinehuf,jnv,jm}@ai.univ-paris8.fr
http://www.ai.univ-paris8.fr

Abstract. We present a general approach for the decomposition of games described in the *Game Description Language* (GDL). In the field of *General Game Playing*, the exploration of games described in GDL can be significantly sped up by the decomposition of the problem in sub-problems analyzed separately. Our program can decompose game descriptions with any number of players while addressing the problem of joint moves. This approach is used to identify perfectly separable subgames but can also decompose serial games composed of two subgames and games with compound moves while avoiding, unlike previous works, to rely on syntactic elements that can be eliminated by simply rewriting the GDL rules. We tested our program on 40 games, compound or not, and we can decompose 32 of them successfully in less than 5 s.

1 Introduction

Despite incentives from Genesereth and Björnsson [3] to encourage the development of GGP players able to discern structure of compound games and therefore to dramatically decrease search cost, not much research exists in this area.

Cox et al. [2] prove conditions under which a global game represents multiple, simultaneous independent sub-games, but the practical implementation of a GGP player using decomposition presents two major issues: the first is to detect and decompose a compound game, the second is to combine local subgame solutions into a global one.

Cerexhe et al. [1] provide a systematic approach for single player games to solve this second difficulty which they refer to as the *composition problem*. However, identifying and decomposing games is not within the scope of their paper.

Günther et al. [5,6] propose a decomposition approach for single player games by building a dependency graph between fluents and actions: the connected parts of the graph represent the different subgames. Potential preconditions, positive and negative effects between fluents and actions are used to build this dependency graph while action-independent fluents are isolated in a separate subgame to prevent them from blocking the decomposition.

Zhao et al. [10,11] propose a similar approach for multiplayer games using partially instantiated fluent and action terms. Serial games and games with compound actions are handled separately.

T. Cazenave et al. (Eds.): CGW 2016/GIGA 2016, CCIS 705, pp. 165–177, 2017.
DOI: 10.1007/978-3-319-57969-6_12

These approaches present different shortcoming we will details below such as a heavy reliance on certain syntactic structures in game descriptions.

We propose a more general approach to decompose games with any number of players while addressing the problem of joint moves, compound moves and serial games without relying on syntactic elements that can be eliminated by simply rewriting the GDL rules. The result of our decomposition can be used to solve the game by an approach like the one of Cerexhe et al. [1]; it is a non-trivial problem outside the scope of this paper.

We begin (Sect. 2) with a brief introduction of the *Game Description Language* and the different types of compound games that can be found on the different online servers and that our approach can decompose. Then we present the different aspects of our method to handle these different types of games (Sect. 3). We present results on 40 games, compound or not (Sect. 4). Finally, we conclude and present future work (Sect. 5).

2 Preliminaries

We present here some details about the *Game Description Language* and the different types of compound games that our approach can decompose.

2.1 The Game Description Language

We assume familiarity of the reader with the General Game Playing [4] as well as with the Game Description Language (GDL) [7]. A GDL game description takes the form of a set of assertions and of logical rules which conclusion describes: the transition to the next position (*next* predicate); the legality of actions (*legal*); the game termination (*terminal*); and the score (*goal*). The rules are expressed in terms of actions (*does*) and fluents (*true*) describing the game state.

Rule premises can also include *auxiliary predicates*, specific to the game description itself, which truth is defined by rules also using *true* and *does* premises. In the rest of this article, we will refer to *auxiliary predicates*, exclusively defined in terms of fluents (*true*) (*does* never appear in their premises), which have an important role in our decomposition approach (Sects. 3.3, 3.5).

2.2 Types of Compound Games

Among games available on the different *General Game Playing* servers (http://games.ggp.org) different types of compound games can be identified. The types we distinguish represent specific issues for the decomposition and are not directly related to the formal classification proposed by (Cerexhe et al. [1]).

For example, ***Parallel games*** like *Dual Connect 4* or *Double Tictactoe Dengji* are composed of two subgames played in parallel that can be *synchronous* or *asynchronous*, but this difference has no influence on the decomposition approach to use. Decomposing these games does not present any particular difficulty.

However, in some synchronous parallel games like *Asteroids Parallel* each player's action is a ***compound moves*** corresponding to two simultaneous actions played in each subgames. These create a strong connection between subgames and represent a specific difficulty for decomposition.

Serial Games like *Blocker Serial* are composed of two *sequential* subgames i.e. the second starts when the first is completed. As the two games are linked together, identifying the boundary between them is a specific issue for decomposition.

Multiple Games like *Multiple Buttons And Lights* are composed of several subgames, only one of them being involved in the score calculation or the game termination. The other subgames only increase the size of the game tree to explore. Identifying those *useless* subgames allows to avoid unnecessary calculations. Note that in the game *Incredible*, *contemplate* actions are detected as *noop* actions by our decomposition program and does not constitute a *useless* subgame.

Games using a Stepper to ensure finite games like *Eight Puzzle* may be considered as compound games (*synchronous*). In these games, different descriptions of a position can vary only by the value of the *stepper* (step counter). To allow a programmed player to exploit these near-perfect transpositions, it is necessary to operate a game decomposition to separate the stepper from the game itself. This stepper is then an *action independent subgame*.

Some Impartial Games, like *Nim* starting with several piles of objects, may also be considered as compounds games (*asynchronous*) as they can be decomposed in several subgames, one for each pile, each of them being an impartial game [10]. Identifying that these subgames are impartial, subsequently allows to use known techniques for the resolution of the global game.

3 Method

Our approach is based on Günther's idea [5] and consists in using a dependency graph between actions and fluents, and then to identify the connected parts of the graph representing the subgames. As nothing in the GDL specification prohibits the use of completely instantiated rules or prevents that fluents or actions be reduced to simple atoms, we identify relations between totally instantiated fluents f and actions a and rely neither on their predicates names nor their arguments.

For the analysis of these relations, we use the following definitions:

Definition 1. *Let F be the set of all the instantiated fluents f appearing in $true(f)$ or $\neg true(f)$.*

Definition 2. *R being the set of all the roles r and O the set of all options o of these roles, let $A \subset R \times O$ be the set of all the instantiated player actions $a = (r, o)$.*
O_r is the set of all the possible options of role r.

Definition 3. *Let C be the set of all the possible conjunctions of atoms of the form $true(f)$, $\neg true(f)$, $does(r, o)$ or $\neg does(r, o)$.*

3.1 Grounding and Creation of a Logic Circuit

To instantiate completely the rules (grounding), we carry out a fast instantiation using Prolog with tabling [9] and use these instantiated rules to build a logic *circuit* similar to a *propnet* [8]. Conclusions of *legal*, *next*, *goal* or *teminal* rules are the outputs of the circuit and only depends on fluents (*true*) and actions (*does*) at the inputs.

It is possible, according to the GDL specifications, to produce a description with fully developed rules using no auxiliary predicate at all. However, these predicates, like *column1*, *diagonal2* or *game1over* in *Tictactoe*, may be necessary for some specific stages of our process of decomposition (Sects. 3.3, 3.5). To ensure that these auxiliary predicates will be available even when not specified in the GDL description, we proceed to a factorization of the conjunctions, disjunctions and use De Morgan's laws to reduce the number of negations in the circuit. As a perfect factorization is an NP-hard problem, our program uses a greedy approach where the first common factor is used. Factorization and application of De Morgan's laws are iterated until the circuit reaches a minimum size.

We identify the needed auxiliary predicates as these are represented by internal logic gates of the circuit, depending only on input fluents and representing important expressions in the logic of the game i.e. these expressions are used several times, several logic gates use their outputs.

After the factorization, the GDL description is a set of formulas under disjunctive normal form of which atoms are fluents, actions, and auxiliary predicates. In the following we say that these formulas are under *DNF* form.

Other stages of the decomposition process need a description of the game under canonical form. By recursively replacing auxiliary predicates by their expression we obtain a new set of formulas in disjunctive normal form describing the same game where all the auxiliary predicates have been eliminated. In the following we say that these formulas are under *DNFD* form.

3.2 Building a Dependency Graph

To build our dependency graph and to identify the different subgames, we start with a set of vertices which are the fully instantiated actions and fluents. We then identify different relations between these fluents and actions that we define below. For each of these relations we add an edge between the involved actions and fluents vertices. These relations correspond to preconditions or effects of the actions.

Unfortunately, GDL does not explicitly describe action effects unlike STRIPS or PDDL languages used for planning domains. A fluent being *false* by default, an action present in a *next* rule can have an effect or not. For example, let us consider the legal actions $does(r, a)$, $does(r, b)$ and $does(r, c)$, in the rule $next(f) :\!-\ \neg true(f) \wedge (does(r, a) \vee does(r, b))$. a and b have an effect if the rule means *"The cell will contain a pawn if r does one of the 2 actions moving a pawn in it"* and c has an effect if it means *"the boat will sink if r does anything else than action c (bailing)"*. A similar example can be found for any *next* rule

with an action (in a negation or not) and regardless of the value of the fluent f and its presence or not in the rule premises.

It is thus possible to produce GDL descriptions in which the actions present in a *next* rule body belong to another subgame than the fluent in the rule head. We can only address this using heuristics similar to those of Günther [6].

They propose to consider that an action a has a negative effect on a fluent f if this action does not keep the fluent true i.e. if $next(f)$ does not contain $true(f) \wedge does(a)$ in its premises. However in a game like Double Tictactoe, there is no rule like this to indicates that actions of a subgame do not change the value of the other subgame fluents. Consequently, fluents of a subgame can be considered as negative effects of the second subgame actions and the decomposition fails.

In our approach we use slightly different heuristics which work well for existing composed games to find potential effects of actions:

Definition 4. *The fluent f is a **potential negative effect** of the action $a = (r, o)$ if $next(f)$ under DNFD has a clause where $\neg does(r, o)$ appears.*

*The fluent f is a **potential positive effect** of the action $a = (r, o)$ if $next(f)$ under DNFD has a clause containing the $does(r, o)$ literal and not containing the $true(f)$ literal.*

In case of joint moves from several players, it is necessary to identify if the action of each player is responsible of the observed effect on the rule conclusion to avoid linking unrelated action with the conclusion.

To solve this problem Zhao et al. [11] propose to compare the arguments used in a *next* rule head with the ones used in the moves (*does*). For example, in the following rule from *Blocker Serial*, we can see that the action from *crosser* is the only one that is likely to affect the conclusion:

$$next(cell2(\mathbf{XC}, \mathbf{YC}, crosser)) :- distinctcell(\mathbf{XC}, \mathbf{YC}, XB, YB) \wedge does(crosser, mark2(\mathbf{XC}, \mathbf{YC})) \wedge does(blocker, mark2(XB, YB)).$$

However, GDL specification allows to use completely instantiated rules and simple atoms to represent fluents and moves. For example, we can replace the previous rule by some instantiated rules:

$$next(f) :- does(crosser, o1) \wedge does(blocker, o2).$$
$$next(f) :- does(crosser, o1) \wedge does(blocker, o3).$$
$$...$$

With fluents like f and moves like $does(r, o)$, their approach is no longer able to deal with joint moves.

To identify which action has an effect without relying on syntactic elements, we compare, for each player, the different actions used in conjunction with the same fluents and actions of other players in the clauses of each *next* rule.

Suppose that $next(f) \leftarrow C_f$ is in DNFD. Let us consider a specific option o' for player r'. We consider the set $E(o')$ of the different options of the role r when r' choose the o' option:

$$E(o') = \{\ \ o \in O_r\ \ |\ \ \exists c \in C_f, \exists b \in C, \\ c = does(r, o) \wedge does(r', o') \wedge b\ \ \}$$

We define $E(o)$ the same way by exchanging the role of (r, o) with (r', o').

If all the options of the r are present in conjunction with the same action of r': these options have probably no effect i.e. the result is the same regardless of the option chosen. On the contrary, if a single option of r is present, it is probably responsible for the observed effect. We then use the following heuristics:

Definition 5. *The action $a = (r, o) \in A$ is **potentially responsible for an effect** on f if:*

- $card(E(o')) = 1$, *or*
- $E(o') \subsetneq O_r$ *and* $card(E(o)) \neq 1$

For example, in the game *BlockerSerial*, the term $next(cell1(2, 3, crosser))$ is *true* if *blocker* choose any option but $mark1(2, 3)$ and *crosser* choose the $mark1(2, 3)$ option. All the options of *blocker* are not represented but, as *crosser* has a single possible option, its action is considered responsible for the effect while actions of *blocker* are not linked to the $cell1(2, 3, crosser)$ fluent.

Even if this approach sometimes put aside actions related to the conclusion, we did not observe any over-decomposition. At least one of the actions is indeed related to the conclusion and edges between fluents and actions added in the dependency graph to represent preconditions relations are redundant with those added for effect relations.

Therefore a fluent is a ***potential effect*** of an action if this action has a *potential positive or negative effect* on this fluent and if this action is *potentially responsible for this effect* in presence of joint moves. From the potential effect of actions we can deduce fluents that are action-independent, such as *step* or *control* fluents, and actions that are fluent-independent such as *noop* actions:

Definition 6. *A fluent f is **action-independent** if it is not the potential effect of any action a. An action a is **fluent-independent** if no fluent f is the potential effect of this action.*

Then we can identify fluents that are potential preconditions of an action in the same subgame and create a link in the graph between them:

Definition 7. *The fluent f is a **potential precondition in the same subgame** of the action $a = (r, o)$ if:*

- *a is not fluent-independent, and*
- *f is not action-independent, and*
- *one of the two following conditions holds:*
  - *$legal(r, o)$ under DNFD has a clause where $true(f)$ or $\neg true(f)$ appears, or*
  - *it exist f' which is a potential effect of a, such that $next(f')$ under DNFD has a clause containing $does(r, o) \wedge true(f)$ or $does(r, o) \wedge \neg true(f)$.*

An action-independent fluent can be present in the premises of all *legal* rules, it is then a precondition of all actions but belongs to another subgame which is action-independent.

3.3 Subgoal-Predicates to Fix Over-Decomposition

Edges between actions and fluent vertices corresponding to preconditions or effects of these actions may not be sufficient to connect all the elements of a subgame. For instance, in a subgame like *Tictactoe*, an action has an effect on a cell and the state of this cell is a precondition to this action. However, no link exists through actions between fluents describing different cells.

In the game *Double Tictactoe* given as an example by Zhao et al. [11] the *auxiliary predicates line1/1* or *line2/1* are present in the premises of some *legal* rules. All the fluents in the premises of these predicates are then preconditions of the corresponding actions and create a link between the cells of each subgame. However, in games like *Tictactoe Parallel*, *Connect4* or *Rainbow* no such predicate is present in the *legal* rules and an over-decomposition occurs.

The logic link between elements of a subgame is in the goal to reach and this goal is usually a condition for the termination of the global game. We need to distinguish an auxiliary predicate corresponding to a subgoal in one subgame from one corresponding to different subgoals from different subgames because the second one can prevent the decomposition. To address this problem of over-decomposition we use the following heuristic to identify potential subgoal-predicates corresponding to only one subgame:

Definition 8. *Let g be the maximum possible score of r. An auxiliary predicate b is a* ***potential subgoal-predicate*** *if:*

- *terminal depends on the logical value of b, and*
- *$goal(r, g)$ under DNF has a clause where b appears.*

or

- *All the roles play in different subgames, and*
- *$goal(r, g)$ under DNF has a clause where b appears, and for all roles $r' \neq r$, $goal(r', g')$ under DNF has no clause where b appears.*

In games like *Dual Rainbow* or *Dual Hamilton*, subgoal-predicates appear only in the premises of *goal* rules. Since these games are composed of single player subgames, an auxiliary predicate present in the *goal* rule of a single player involves only this player and therefore only one subgame.

The first part of the definition holds in the games where the victory in one of the subgames terminates the game as it is generally the case in compound games. Otherwise, the subgames may be connected by the use of a misidentified subgoal-predicate.

Once a subgoal-predicate is identified, we add edges in our dependency graph between fluents that appear in a same clause in its formulas under DNFD.

3.4 Compound Moves and Meta-Action Sets

A compound move is composed of two or more actions related to different subgames. For example, the compound move *legal*(*ship*,*do*(*clockcounter*)) in the

game *Asteroid Parallel* corresponds to a *clock*wise move in a first subgame and a *counter*clockwise move in a second subgame. Such an action creates a link between the different subgames and can interfere with the decomposition process.

To detect compound moves, Zhao et al. [11] use the same approach as that applied to the problem of joints move. For example, in the following rule from *Tictactoe Parallel* we can see that only the first two arguments of the action have an effect on the rule conclusion: $next(cell1(\mathbf{X1}, \mathbf{Y1}, o)) :- does(oplayer, mark(\mathbf{X1}, \mathbf{Y1}, X2, Y2))$. Once again, the rule has just to be rewritten to defeat detection: $next(f) :- does(oplayer, o)$.

In games with compound moves, the set of all actions is a combination of the sets of all actions of each subgame. Then in a game composed of two subgames, for each action in the first subgame, there is N compound moves corresponding to this action combined to the N possible actions in the second subgame. To identify the different parts of compound moves, we distribute actions into meta-action sets. An action can belong to one or several meta-action sets which depend only on a role r, a fluent $f \in F$ and two clauses $c \in C$ and $c' \in C$.

Definition 9. *An action* $a = (r, o)$ ***belongs to the meta-action set*** $P(r, f, c, c')$ *if:*

- f *is a potential effect of* a*, and*
- $next(f)$ *under DNFD has a clause* $(does(r, o) \wedge c)$*, and*
- *if* c' *is empty,* $legal(r, o)$ *must always be true, or if* c' *is not empty, it contains only action-dependent literals and appears in at least one clause of* $legal(r, o)$ *under DNFD.*

Therefore a meta-action set is a group of actions with an identical effect on a fluent of a particular subgame, the same preconditions in the corresponding *next* rule and at least one precondition in common in their *legal* rules.

For example, in the game *Blocks World Parallel* we can find the meta-action set $\{does(robot, do(\mathbf{stack}stack, \mathbf{a}, \mathbf{b}, *, *)), does(robot, do(\mathbf{stack}unstack, \mathbf{a}, \mathbf{b}, *, *))\}$[1] corresponding to the action $stack(a, b)$ in the first subgame. These actions have an effect in common on $true(on1(a, b))$, same preconditions $\{true(table\,1(a)), true(clear1(b)), true(clear1(a))\}$ in the $next(on1(a, b))$ clauses and are always legal.

In a game with compound actions, each action is placed in M meta-action sets corresponding to M effects. If a game contains no compound action but some actions with an identical effect in the same situation, these actions are grouped in the same meta-action set. And finally, if all actions in a game have a different effect, each one constitutes a meta-action singleton. The use of meta-action sets is then compatible with all games.

In our dependency graph, we then encapsulate all actions into meta-action sets to avoid compound actions from connecting different subgames. The links

[1] The * represents different possible values, the whole meta-action set contains 12 compound moves.

between actions and fluents are replaced by links between action sets and fluents i.e. in the dependency graph, edges are added between a meta-action set and its effect f and preconditions $f' \in c \cup c'$.

3.5 Serial Games

In serial games an auxiliary predicate describing the terminal situation of the first subgame determines the legality of all actions of the second subgame. Consequently, it creates links between first subgame fluents and second subgame actions. We must detect it and avoid these links to separate both subgames.

Zhao [10] uses a separate special detection: the desired auxiliary predicate must be *false* to authorize the first subgame actions and *true* to authorize the second ones, like *game1over* in *Tictactoe Serial*:

$$legal(PLAYER, mark1(X,Y)) :- \neg game1over \wedge ... .$$
$$legal(PLAYER, mark2(X,Y)) :- game1over \wedge ... .$$

with $game1over$ depending on $line1(x) \vee line1(o) \vee \neg open1$. However, someone can defeat this approach by simply rewriting the first subgame *legal* rules with a different precondition: $legal(PLAYER, mark1(X,Y)) :- ongoing1 \wedge ... .$ with $ongoing1$ depending on $\neg line1(x) \wedge \neg line1(o) \wedge open1$.

To generalize the approach of Zhao [10], we consider that a *pivot* between two serial subgames is composed of two auxiliary predicates that can be the negation of each other or two completely different predicates. We use our circuit representing the game to test the influence of each auxiliary predicate detected during the circuit creation on the actions legality and look for a couple of predicates that parts the fluent-dependent actions in two groups.

If such a couple of auxiliary predicates is found, then it is a *pivot* and the latter predicates are directly used as action preconditions instead of the fluents included in them. In our dependency graph, fluents of the first subgame are then encapsulated in these auxiliary predicates to ensure that they will not connect the different subgames with direct links to actions (meta-action sets) of the second subgame. This approach works for existing games that are limited to two serial subgames.

Unfortunately, we cannot generalize this approach and identify a *pivot* in case of more than two serial subgames without risking an over-decomposition of games with movable parts. In a *pivot*, each auxiliary predicate is necessary to allow the legality of some actions and may prevent the legality of other actions. If a third subgame is present, its actions are not affected by both auxiliary predicates. In a game with movable pawns, an auxiliary predicate may be used to describe the state of a cell; this predicate may allow the legality of some moves from this cell, prevent some moves to this cell and does not concern other moves of the game, consequently it may be confused with a part of a *pivot*. Therefore, if we try to identify *pivots* for more than two serial subgames with a generalization of this approach, a game with movable pawn may be over-decomposed, each cell being a small serial subgame leading to the next ones.

3.6 Multiple Games and Useless Subgames

Some subgames are involved in the calculation of the score or can cause the end of the game when some position is reached. A subgame may also be played to allow another subgame to start in the case of serial subgames.

Definition 10. *Let V_S be the set of vertices of a connected part of the dependency graph representing a subgame S. S is considered* ***useful*** *if:*

- *S is played before another subgame in a serial game and is necessary to start it, or*
- *it exists $f \in F \cap V_S$ such that terminal depends on the logical value of $true(f)$, or*
- *it exists $f \in F \cap V_S$ such that $goal(r, g)$ depends on the logical value of $true(f)$.*

In multiple games, all the subgames that are not identified as *useful* can be ignored and remain unexplored. However, a *useless* action (*noop*) can be sometime strategically useful to avoid a *zugzwang* in another subgame. Actions of these subgames can then be flagged as *noop* actions, be considered equivalently *useless*, and only one of them need to be explored (if legal) for each position of the game.

4 Experiments

We evaluated our decomposition program on a panel of 40 descriptions of games, compound or not, from the servers of Dresden, Stanford and Tiltyard. We took all the available compound games except for the redundant ones. We added the original version of games commonly used as subgames and a representative panel of games with different characteristics (movable parts, steppers, asymmetry, impartiality) and complexity. The experiments were run on one core of an Intel Core i7 2,7 GHz with 8Go of 1600 MHz DDR3.

For each game, we measured the mean time necessary for each stage of the decomposition on a set of 100 decomposition tests. To limit the duration of the experiments, a decomposition test was aborted after 60 min. The longest stages of the decomposition are grounding the rules, factorizing the circuit and calculating completely developed disjunctive normal forms (DNFD). The column 5 of Table 1 indicates the total time needed to decompose each game and shows that the DNFD calculation can be very time consuming.

We try to compute DNF without developing the auxiliary predicates identified during the circuit construction. As we can see it in column 6, the time saved is really significant and allows the successful decomposition of 32 games among 40 in less than 5 s. The major part of the total time necessary for the decomposition using DNF corresponds to the rules grounding and circuit factorization.

Unfortunately, the use of partially developed DNF presents a shortcoming: if a rule containing variables is already instantiated in the original GDL description of a game and if some of these instances only are expressed in terms of auxiliary predicates, actions may occur in conjunction with different but equivalent

premises: a group of fluents or an equivalent auxiliary predicate. The factorization of the circuit should restore auxiliary predicates in all rules instances but as we use a greedy approach (Sect. 3.1), it is not guaranteed. Therefore, meta-action sets detection may be hindered. Nevertheless, this case is sufficiently specific to successfully use the auxiliary predicates in DNF, in most cases.

For *Hex* and *Blocker Parallel*, the time required to compute the grounded rules, the factorization and the DNFs still remains too large. The factorization does not allow to sufficiently reduce the complexity of *Hex* and, in *Blocker Parallel*, the presence of compound actions combined with joint moves for both players brings a large number of combinations.

Note that *LeJoueur* of Jean Noël Vittaut, which won the 2015 Tiltyard Open, is on average 8.5 times faster to ground and factorize the three most complex games (*Breakthrough*, *Hex* and *Blocker Parallel*). This indicates the potential scope for improving these steps.

Table 1 also shows the total number of subgames discovered for each of the 40 games and among them, the ones that are action-dependent and action-independent. The figures in parenthesis indicate the number of discovered subgames considered as useless.

Games at the top of the table are composed of only one action-dependent subgame and sometimes a stepper detected as a useful action-independent subgame. The useless action-independent subgame detected for games like *Breakthrough* or *Sheep and Wolf* corresponds to the *control* fluents which indicate the active player in an alternate moves game and does not represent a playable game per se.

Useless subgames in multiple games are correctly identified. We remark that for *Multiple Tictactoe*, the number of useless subgames is particularly large because these subgames have been over-decomposed as no auxiliary predicate creates a link between their cells.

For the game of *Nim*, our program has detected an action-independent subgame not involved in the end of the game (it is not a stepper) while it is the only subgame useful for the calculation of the score: this is an important clue indicating that this game is impartial.

Except for the special case of Chomp, all the detected subgames are the expected ones and correspond to what would have been obtained by a manual decomposition. Chomp is an example of a game on which the heuristics used for the action effects detection do not work properly. Other actions than eating the poisoned chocolate square have only implicit negative effects which are not detected. These actions are considered as *noop* actions and would be evaluated as equivalent during the game: this could not allow the player to prevent the fatal outcome. Fortunately, such a wrong detection of the action effects is visible in the resulting dependency graph as a huge proportion of fluents and actions are isolated vertices. So we can prevent this error from affecting the game solving.

Table 1. Result of the decomposition for a panel of 40 games descriptions from the servers of Dresden (D), Stanford (S) and Tiltyard (T) with comments on subgames (SG) found.

game	# total of SG	# SG with actions (# useless)	# action-indep. SG (# useless)	time (DNFD)	time (DNF)	comments
Hex (T)	*not decomposed after 1 hour*					
Blockerparallel (D)	*not decomposed after 1 hour*					
Asteroids (D)	2	1	1	<1sec	<1sec	
Blocks (D)	2	1	1	<1sec	<1sec	
EightPuzzle (T)	2	1	1	<2sec	<2sec	
Roshambo2 (D)	2	1	1	<1sec	<1sec	
Checkers (D)	3	1	1 (1)	>**1hr**	<12min	
Breakthrough (T)	2	1	(1)	<16min	<16min	
Sheep and wolf (D)	2	1	(1)	>**1hr**	<5sec	
Tictactoe (S)	2	1	(1)	≈1sec	<1sec	
Nineboardtictactoe (S)	2	1	(1)	>**1hr**	<2sec	SG = 9 Tictactoe together
Tictactoex9 (D)	2	1	(1)	>**1hr**	<5sec	SG = 9 Tictactoe together
Chomp (D)	2	1	(1)	<1sec	<1sec	⚠**Wrong decomposition**
Multiplehamilton (S)	3	1 (1)	1	<1sec	<1sec	SG = Hamilton (only right useful)
Multiplebuttonsandlights (S)	10	1 (9)	1	<1sec	<1sec	SG = group of buttons (only n^o5 useful)
Multipletictactoe (S)	75	1 (72)	1 (1)	<10sec	<1sec	SG = Tictactoe n^o5 (+ useless SG = cells)
Blockerserial (D)	2	2		<20min	<10min	SG = Blocker ×2
Dualrainbow (S)	2	2		≈1min	<8sec	SG = Rainbow ×2
Asteroidsparallel (D)	3	2	1	<1sec	<1sec	SG = Asteroid ×2
Blocksworldparallel (D)	3	2	1	≈1sec	≈1sec	SG = Blocks ×2
Dualhamilton (S)	3	2	1	<1sec	<1sec	SG = Hamilton ×2
Dualhunter (S)	3	2	1	<2sec	<2sec	SG = Hunter ×2
Incredible (D)	3	2	1	<1sec	<1sec	SG = Maze et Block
Asteroidsserial (D)	4	2	2	<1sec	<1sec	SG = Asteroid ×2
Blocksworldserial (D)	4	2	2	<1sec	<1sec	SG = Blocks ×2
Jointbuttonsandlights (S)	4	3	1	<1sec	<1sec	SG = 3 groups of buttons
LightsOnParallel (T)	5	4	1	<8min	<1sec	SG = 4 groups of lights
LightsOnSimul4 (T)	5	4	1	<8min	<1sec	SG = 4 groups of lights
LightsOnSimultaneous (T)	5	4	1	<8min	<1sec	SG = 4 groups of lights
Nim3 (D)	5	4	1	<2sec	<2sec	SG = 4 heaps + control useful for *goal*
Chinook (S)	6	2	2 (2)	<14sec	<14sec	SG = Checkers ×2
Double tictactoe dengji (D)	3	2	(1)	>**1hr**	<1sec	SG = Tictactoe ×2
SnakeParallel (T)	3	2	(1)	>**1hr**	<2sec	SG = Snake ×2
TicTacToeParallel (T)	3	2	(1)	>**1hr**	≈2sec	SG = Tictactoe ×2
Doubletictactoe (D)	4	2	(2)	>**1hr**	<1sec	SG = Tictactoe ×2
TicTacHeaven (T)	4	2	(2)	>**1hr**	<2sec	SG = 9 Tictactoe together + 1 isolated
TicTacToeSerial (T)	4	2	(2)	>**1hr**	<1sec	SG = Tictactoe ×2
ConnectFourSimultaneous (T)	4	2	(2)	>**1hr**	<1sec	SG = Connect4 ×2
DualConnect4 (T)	4	2	(2)	>**1hr**	<1sec	SG = Connect4 ×2
Jointconnectfour (S)	4	2	(2)	>**1hr**	<1sec	SG = Connect4 ×2

5 Conclusion and Future Work

In this paper we presented a general approach for the decomposition of games described in the *Game Description Language* (GDL). Our program decomposes descriptions of games, compound or not, with any number of players while

addressing the problem of joint moves. It decomposes parallel games, games with compound moves and serial games composed of two subgames. It also identifies steppers, useless subgames in multiple games, and unlike previous works, without relying on syntactic elements that can be eliminated by simply rewriting GDL rules. We tested our program on 40 games, compound or not, and have decomposed 32 of them with success in less than 5 s which is a time compatible with GGP competition setups.

Using Meta-action sets is an efficient way to the problem raised by compound moves (Sect. 3.4). However, it requires the completely developed disjunctive normal form of the *next* rules which is computationally expensive. We are seeking another approach to avoid this need or to minimize its computation time. Beside this, we plan to eliminate the ad-hoc heuristics used to identify action effects (Sect. 3.2) and to avoid over-decomposition (Sect. 3.3). We will also address the problem of the decomposition of more than two sequential subgames.

Finally, using these decomposed games to solve the *composition problem* for any games with any number of players remains an open problem.

References

1. Cerexhe, T., Rajaratnam, D., Saffidine, A., Thielscher, M.: A systematic solution to the (de-)composition problem in general game playing. In: Proceedings of the European Conference on Artificial Intelligence (ECAI), pp. 195–200. IOS Press (2014)
2. Cox, E., Schkufza, E., Madsen, R., Genesereth, M.: Factoring general games using propositional automata. In: Proceedings of the IJCAI-09 Workshop on General Game Playing (GIGA 2009), pp. 13–20 (2009)
3. Genesereth, M., Björnsson, Y.: The international general game playing competition. AI Mag. **34**(2), 107–111 (2013)
4. Genesereth, M.R., Love, N., Pell, B.: General game playing: overview of the AAAI competition. AI Mag. **26**(2), 62–72 (2005)
5. Günther, M.: Decomposition of Single Player Games. Master's thesis, TU-Dresden, Germany (2007)
6. Günther, M., Schiffel, S., Thielscher, M.: Factoring general games. In: Proceedings of the IJCAI-09 Workshop on General Game Playing (GIGA 2009), pp. 27–33 (2009)
7. Love, N., Hinrichs, T., Haley, D., Schkufza, E., Genesereth, M.: General Game Playing: Game Description Language Specification. Technical report LG-2006-01, Stanford University, March 2008
8. Schkufza, E., Love, N., Genesereth, M.: Propositional automata and cell automata: representational frameworks for discrete dynamic systems. In: Wobcke, W., Zhang, M. (eds.) AI 2008. LNCS (LNAI), vol. 5360, pp. 56–66. Springer, Heidelberg (2008). doi:10.1007/978-3-540-89378-3_6
9. Vittaut, J., Méhat, J.: Fast instantiation of GGP game descriptions using prolog with tabling. In: ECAI 2014, pp. 1121–1122 (2014)
10. Zhao, D.: Decomposition of Multi-Player Games. Master's thesis, TU-Dresden, Germany (2009)
11. Zhao, D., Schiffel, S., Thielscher, M.: Decomposition of multi-player games. In: Nicholson, A., Li, X. (eds.) AI 2009. LNCS (LNAI), vol. 5866, pp. 475–484. Springer, Heidelberg (2009). doi:10.1007/978-3-642-10439-8_48