Space-Consistent Game Equivalence Detection in General Game Playing

Haifeng Zhang(✉), Dangyi Liu, and Wenxin Li

Peking University, Beijing, China
{pkuzhf,ldy,lwx}@pku.edu.cn

Abstract. In general game playing, agents play previously unknown games by analyzing game rules which are provided in runtime. Since taking advantage of experience from past games can efficiently enhance their intelligence, it is necessary for agents to detect equivalence between games. This paper defines game equivalence formally and concentrates on a specific scale, space-consistent game equivalence (SCGE). To detect SCGE, an approach is proposed mainly reducing the complex problem to some well-studied problems. An evaluation of the approach is performed at the end.

1 Introduction

According to human experience, exploiting equivalence between a new problem and a studied problem provides a bridge for knowledge transfer, which efficiently enhances the understanding of the new problem. Therefore, for the aim of artificial intelligence, it is important to enable computer to recognize equivalence. Particularly, as a typical application of AI, it is necessary for game-playing agents to grasp the ability of detecting equivalence between games.

The main work of this paper is to discuss classification of game equivalence, define concepts of it formally and propose an approach to detect it. Since detecting the general equivalence between games is difficult, a narrowed scale of game equivalence, *space-consistent game equivalence*, is defined firstly. Then, an approach is proposed for agents to automatically detect space-consistent game equivalence, which intends to achieve an acceptable efficiency by defining a grounded rule graph and transferring the complex problem to the well-studied problems, i.e. graph isomorphism and SAT.

This paper discusses game equivalence in the domain of General Game Playing (GGP) [2], which sets up a framework for agents to play previously unknown games by being provided game rules in runtime. This framework obliges agents to take over the responsibility of analyzing game rules from human beings. The games in GGP are turn-based, synchronized and of complete information, which are described in the Game Description Language (GDL) [5].

The work of this paper can be applied to knowledge transfer between equivalent or similar games. For example, [4] introduces a method of value function transfer for speeding up reinforcement learning, based on the technique of game equivalence detection. It can also be applied to detect symmetry of games, as [8] does.

T. Cazenave et al. (Eds.): CGW 2015/GIGA 2015, CCIS 614, pp. 165–177, 2016.
DOI: 10.1007/978-3-319-39402-2_12

The following section provides background on GGP and introduces definitions of game. Section 3 discusses game equivalence and its narrowness. Section 4 introduces the proposed approach to detect game equivalence, which is evaluated in Sect. 5. Section 6 concludes the work of this paper.

2 General Game Playing

In the domain of General Game Playing, games are modeled as finite state machines. In this paper, the definitions of game derive from [9].

Definition 1 (Game). *Let Σ be a countable set of ground (i.e., variable-free) symbolic expressions (terms), S a set of states, and A a set of actions. A (discrete, synchronous, deterministic) game is a structure (R, s_0, T, L, u, G), where*

- *$R \subseteq \Sigma$ finite (the roles);*
- *$s_0 \in S$ (the initial state);*
- *$T \subseteq S$ finite (the terminal states);*
- *$L \subseteq R \times A \times S$ finite (the legality relation);*
- *$u : (R \to A) \times S \to S$ finite (the update function);*
- *$G \subseteq R \times \mathbb{N} \times S$ finite (the goal relation).*

Here, $A \subseteq \Sigma$ and $S \subseteq 2^{\Sigma}$. The legality relation $(r, a, s) \subseteq L$ defines action a to be a legal action for role r in state s. The update function u takes an action for each role and (synchronously) applies the joint actions to a current state, resulting in the updated state. The goal relation $(r, n, s) \subseteq G$ defines n to be the utility for role r in state s.

In General Game Playing, rules of games are described in the GDL, which is a Prolog-like language using prefix syntax. Some keywords of the GDL are defined in Table 1. As a demonstration of the GDL, the rules of Tic-tac-toe are provided in Listing 1.1.

Table 1. GDL Keywords

(*role* r)	r is a player
(*init* p)	Proposition p holds in the initial state
(*true* p)	Proposition p holds in the current state
(*legal* r a)	Player r has legal action a in the current state
(*does* r a)	Player r does Action a
(*next* p)	Proposition p holds in the next state
terminal	The current state is terminal
(*goal* r n)	Utility of player r in current terminal state is n

```
1 (role xplayer) (role oplayer)
2 (init (cell 1 1 b)) (init (cell 1 2 b))...(init (cell 3 3 b))
3 (init (control xplayer))
4 (<= (legal ?w (mark ?x ?y)) (true (cell ?x ?y b)) (true (control ?w)))
5 (<= (legal xplayer noop) (true (control oplayer)))
6 (<= (legal oplayer noop) (true (control xplayer)))
7 (<= (next (cell ?m ?n x)) (does xplayer (mark ?m ?n)) (true (cell ?m ?n b)))
8 (<= (next (cell ?m ?n o)) (does oplayer (mark ?m ?n)) (true (cell ?m ?n b)))
9 (<= (next (cell ?m ?n ?w)) (true (cell ?m ?n ?w)) (distinct ?w b))
10 (<= (next (cell ?m ?n b)) (does ?w (mark ?j ?k)) (true (cell ?m ?n b)) (or (distinct ?m ?j)
       (distinct ?n ?k)))
11 (<= (next (control xplayer)) (true (control oplayer)))
12 (<= (next (control oplayer)) (true (control xplayer)))
13 (<= (row ?m ?x) (true (cell ?m 1 ?x)) (true (cell ?m 2 ?x)) (true (cell ?m 3 ?x)))
14 (<= (column ?n ?x) (true (cell 1 ?n ?x)) (true (cell 2 ?n ?x)) (true (cell 3 ?n ?x)))
15 (<= (diagonal ?x) (true (cell 1 1 ?x)) (true (cell 2 2 ?x)) (true (cell 3 3 ?x)))
16 (<= (diagonal ?x) (true (cell 1 3 ?x)) (true (cell 2 2 ?x)) (true (cell 3 1 ?x)))
17 (<= (line ?x) (or (row ?m ?x) (column ?m ?x) (diagonal ?x))
18 (<= open (true (cell ?m ?n b)))
19 (<= (goal xplayer 100) (line x))
20 (<= (goal xplayer 50) (not (line x)) (not (line o)) (not open))
21 (<= (goal xplayer 0) (line o))
22 (<= (goal oplayer 100) (line o))
23 (<= (goal oplayer 50) (not (line x)) (not (line o)) (not open))
24 (<= (goal oplayer 0) (line x))
25 (<= terminal (or (line x) (line o) (not open)))
```

Listing 1.1. Rules of Tic-tac-toe

Here, the symbol <= is the implication operator. Tokens starting with a question mark are variables. The first line declares two roles of the game. Lines 2–3 define the initial state. Lines 4–6 define legal actions for roles. In order to describe an asynchronous turn-based game, an extra action noop is provided to players during their opponents' turns. Lines 7–12 define the update function. For example, Line 7 implies that (cell 1 1 x) holds in the next state if xplayer does the action (mark 1 1) and (cell 1 1 b) holds in the current state. Lines 13–18 define several auxiliary propositions describing properties of the current state. It is convenient to use these propositions in the following rules. Lines 19–24 define the goal relation of the game, while Line 25 defines the terminal states.

Except the keywords and logical words, which are printed italic, all tokens are game-specific and can be replaced by other tokens without changing the meaning of the game. Auxiliary propositions and variables are used for convenience and compactness, which can be eliminated without changing the meaning of the game.

Provided a GDL description, a game is defined as follows.

Definition 2 (Game for GDL). *Let D be a valid GDL game description, whose signature determines the set of ground terms Σ. The game for D is the game (R, s_0, T, L, u, G), where*

- $R = \{r \in \Sigma | D \models (role\ r)\}$
- $s_0 = \{p \in \Sigma | D \models (init\ p)\}$
- $T = \{s \in S | D \cup s^{true} \models terminal\}$
- $L = \{(r, a, s) \in R \times A \times S | D \cup s^{true} \models (legal\ r\ a)\}$

- $u(j : R \to A, s) = \{p \in \Sigma | D \cup j^{does} \cup s^{true} \models (next\ p)\}$
- $G = \{(r, n, s) \in R \times \mathbb{N} \times S | D \cup s^{true} \models (goal\ r\ n)\}$

Here, S is defined as 2^P where $P = \{p|\ (true\ p) \in \Sigma\}$, A as $\{a|\ (legal\ r\ a) \in \Sigma\}$, s^{true} as $\{(true\ p)\ | p \in s\}$ and $(j : R \to A)^{does}$ as $\{(does\ r\ j(r))\ | r \in R\}$.

3 Game Equivalence

Two games looking different in rules may be identical in nature. [7] points out that Tic-tac-toe is identical to Number Scrabble[1]. In fact, filling the numbers of Number Scrabble into the cells of Tic-tac-toe as Fig. 1 reveals the mapping between them.

2	9	4
7	5	3
6	1	8

Fig. 1. Mapping between Tic-tac-toe and Number Scrabble. Picking a number corresponds to marking a cell, and collecting three numbers summing up to 15 corresponds to drawing a line.

Essentially, two games are equivalent exactly if the state machines described them are identical. Corresponding to the definition of game, game equivalence is defined as follows.

Definition 3 (Game Equivalence). *Game $\Gamma = (R, s_0, T, L, u, G)$ and Game $\Gamma' = (R', s'_0, T', L', u', G')$ (Σ and Σ' are their grounds sets, S and S' state sets, A and A' action sets, respectively) are equivalent iff there is a bijection set $\sigma = (\sigma^R : R \leftrightarrow R', \sigma^S : S \leftrightarrow S', \sigma^A : A \leftrightarrow A')$ s.t.*

- $\sigma^S(s_0) = s'_0$
- $(\forall t)\ t \in T \Leftrightarrow \sigma^S(t) \in T'$
- $(\forall r, a, s)\ (r, a, s) \in L \Leftrightarrow (\sigma^R(r), \sigma^A(a), \sigma^S(s)) \in L'$
- $(\forall j : R \to A, \forall s_{cur}, s_{next} \in S)$
  $u(j, s_{cur}) = s_{next} \Leftrightarrow u'(j', \sigma^S(s_{cur})) = \sigma^S(s_{next})$, *where* $j' : R' \to A'$ *satisfies* $j'(\sigma^R(r)) = \sigma^A(j(r))$
- $(\forall r, n, s)\ (r, n, s) \in G \Leftrightarrow (\sigma^R(r), n, \sigma^S(s)) \in G'$

The bijection set σ is called a game equivalence between Γ and Γ'.

[1] Number Scrabble is a game for two players taking turns to pick numbers from a pool of 1–9, whose goals are collecting three numbers summing up to 15 before the opponent achieving it.

Previous works successfully detect some kinds of game equivalence. [4] proposes a rule graph to detect game equivalence caused by rules reordering and tokens scrambling. Based on it, [8] enhances the rule graph to handle arguments reordering. However, more kinds of equivalence exist, such as:

- auxiliary propositions elimination, e.g. replacing (<= (p_0) (p_1)) (<= (p_1) (p_2)) by (<= (p_0) (p_2));
- logical conversion, e.g. replacing (<= (consequence) (not (or ($condition_1$) ($condition_2$)))) by (<= (consequence) (not ($condition_1$)) (not ($condition_2$)));
- arguments re-encoding, e.g. replacing (*true* (cell 1..3 1..3 x)) by (*true* (cell 1..9 x)).

In general, game equivalence is caused by the uncertainty of transformation from a state machine to a GDL description. A state machine can be transformed into different but equivalent propositional nets, which can be further transformed into different but equivalent GDL descriptions. Figure 2 demonstrates a reasonable sequence of transformation steps.

According to Fig. 2, each kind of game equivalence is caused by one of the steps. As to the mentioned ones, rules reordering is caused by Step 8, tokens scrambling and arguments reordering are caused by Step 7, auxiliary propositions elimination is caused by Step 5, arguments re-encoding is caused by Step 3, and logical conversion is caused by Step 2.

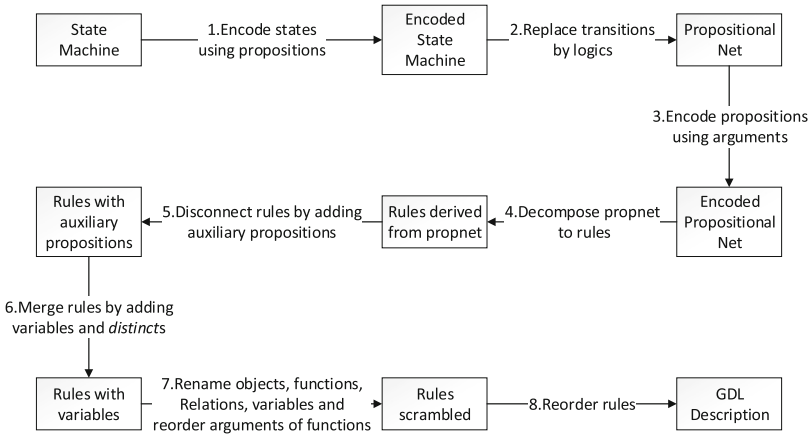

Fig. 2. Transformation steps from state machine to GDL rules. Each step can yield different targets from a single source, except for Step 4 which decomposes a propnet into certain rules.

This paper considers the steps after the encoding state machine in Fig. 2. The encoded state machines derived from a particular state machine share the same propositions. These propositions form a state space, which is also shared by the encoded state machines. To describe it, space-consistent game equivalence is defined.

Definition 4 (Space-Consistent Game Equivalence (SCGE)). *A game equivalence* $\sigma = (\sigma^R, \sigma^S, \sigma^A)$ *is a space-consistent game equivalence for two games* Γ *and* Γ' *iff there is a bijection* $\sigma^P : P \leftrightarrow P'$ *(where* $P = \{p | p \in s, s \in S\}$*,* P' *likewise, i.e.* P *and* P' *contains propositions forming the states) satisfying* $(\forall s \in S)\ p \in s \Leftrightarrow \sigma^P(p) \in \sigma^S(s)$*. Here,* S *is the set of states of* Γ*.* P *and* P' *are called state spaces. The SCGE* σ *can be written as* $(\sigma^R, \sigma^P, \sigma^A)$*, since* σ^S *can be determined by* σ^P*.*

SCGE for GDL is defined as follows, which rewrites the definition of SCGE in the context of GDL without changing the meaning.

Definition 5 (Space-Consistent Game Equivalence for GDL). *Let* D *and* D' *be valid GDL game descriptions, whose signatures determine the sets of ground terms* Σ *and* Σ' *respectively. A space-consistent game equivalence* $\sigma = (\sigma^R : R \leftrightarrow R', \sigma^P : P \leftrightarrow P', \sigma^A : A \leftrightarrow A')$ *for* D *and* D' *satisfies:*

- $D \models p^{init} \Leftrightarrow D' \models (\sigma^P(p))^{init}$
- $D \cup s^{true} \models terminal \Leftrightarrow D' \cup (\sigma^S(s))^{true} \models terminal$
- $D \cup s^{true} \models (legal\ r\ a) \Leftrightarrow D' \cup (\sigma^S(s))^{true} \models (legal\ \sigma^R(r)\ \sigma^A(a))$
- $D \cup s^{true} \cup \{(does\ r\ j(r))\ |r \in R\} \models (next\ p) \Leftrightarrow D' \cup (\sigma^S(s))^{true} \cup \{(does\ r\ j'(r))\ |r \in R'\} \models (next\ \sigma^P(p))$
- $D \cup s^{true} \models (goal\ r\ n) \Leftrightarrow D' \cup (\sigma^S(s))^{true} \models (goal\ \sigma^R(r)\ n)$

Here, p^{init} *is defined as (init p),* s^{true} *as* $\{(true\ p)\ |p \in s\}$ *and* $\sigma^S(s)$ *as* $\{\sigma^P(p) | p \in s\}$*.* $j : R \to A$ *and* $j' : R' \to A'$ *satisfy that* $(\forall r \in R)\ j'(\sigma^R(r)) = \sigma^A(j(r))$*.*

SCGE covers the kinds of game equivalence caused by Step 2 and after in Fig. 2. It narrows the concept of game equivalence by building a bijection between P and P' which determines the bijection between S and S', instead of building bijection between S and S' directly. For an example of space-inconsistent game equivalence which is caused by Step 1, replacing (*true* (cell 1 1 b)) in Tic-tac-toe by (not (or (*true* (cell 1 1 o)) (*true* (cell 1 1 x))) doesn't change the game, but reduces the state space of the game.

For solving the whole problem of game equivalence detection, comparing state machines directly is a method with a very high complexity. However, comparing propositional nets covers the kinds of game equivalence caused by Step 3 and after, whose complexity is logarithmic to the corresponding state machines in most cases. Since logical conversion is a quite common situation of game equivalence, Step 2 should be also taken into consideration. This is the significance of SCGE detection.

4 Space-Consistent Game Equivalence Detection

Based on the definition of SCGE for GDL, a brute force approach to detect it is enumerating all σs mapping roles, actions and propositions of states, then

checking whether all pairs of mapped terms are equivalent to each other. Specifically speaking, it consists of three phases. The first phase is generating the logical implications between keyword-propositions, which is related to the propositional net. The second phase is enumerating all possible σs mapping R to R', P to P' and A to A' so that all keyword-propositions are mapped in accordance. The third phase is verifying whether mapped keyword-propositions are equivalent to each other by comparing the logical implications generated in the first phase.

The brute force approach takes exponential time due to bijection enumeration and logical implication comparison. Therefore, Space-Consistent Game Equivalence Detection Approach (SCGEDA, or GEDA for short) is proposed. It transfers the problem to two well-studied problems, i.e. graph isomorphism and boolean satisfiability, to achieve the state-of-the-art efficiency.

The GEDA consists of three phases as the brute force approach does:

- rule grounding, which is to generate all logical implications between grounded keyword-propositions;
- graph building and mapping, which is to build a dependency graph of keyword-propositions and inspect graph isomorphisms to map keyword-propositions;
- logical equivalence verifying, which is to verify whether the mapped logical implications are equivalent.

In addition, an analysis of complexity and some efficient improvements are to be introduced.

4.1 Rule Grounding

The aim of this phase is to transfer GDL rules to equivalent rules that only contain logical implications of keyword-propositions. An example of grounded rule is displayed as follows:

(<= (*goal* xplayer 100)
  (*or* (*and* (*true* (cell 1 1 x)) (*true* (cell 1 2 x)) (*true* (cell 1 3 x)))
    (*and* (*true* (cell 2 1 x)) (*true* (cell 2 2 x)) (*true* (cell 2 3 x)))
    (*and* (*true* (cell 3 1 x)) (*true* (cell 3 2 x)) (*true* (cell 3 3 x)))
    (*and* (*true* (cell 1 1 x)) (*true* (cell 2 1 x)) (*true* (cell 3 1 x)))
    (*and* (*true* (cell 1 2 x)) (*true* (cell 2 2 x)) (*true* (cell 3 2 x)))
    (*and* (*true* (cell 1 3 x)) (*true* (cell 2 3 x)) (*true* (cell 3 3 x)))
    (*and* (*true* (cell 1 1 x)) (*true* (cell 2 2 x)) (*true* (cell 3 3 x)))
    (*and* (*true* (cell 1 3 x)) (*true* (cell 2 2 x)) (*true* (cell 3 1 x))))).

To achieve it, several procedures are taken.

1. Calculate ranges of arguments, such as
   (*true* (cell {1,2,3} {1,2,3} {x,o})).
2. Replace variables by constants according to ranges of arguments, e.g. replace
   (<= (diagonal ?x)
     (*true* (cell 1 1 ?x)) (*true* (cell 2 2 ?x)) (*true* (cell 3 3 ?x)))
   by

(<= (diagonal x)
(*true* (cell 1 1 x)) (*true* (cell 2 2 x)) (*true* (cell 3 3 x)))
(<= (diagonal o)
(*true* (cell 1 1 o)) (*true* (cell 2 2 o)) (*true* (cell 3 3 o))).

(The consistency of variables is ensured. If a *distinct*-proposition is contained in a rule, its logical value is computed and applied to the rule during this procedure.)

3. Eliminate auxiliary propositions stage by stage, e.g. replace line by row, column and diagonal before replacing row, column and diagonal by *true*.
4. Remove non-state-relative propositions and *role*-propositions from premises of rules, because their values are always true.
5. Remove the rules which use auxiliary propositions as consequences, because they are no longer of use.
6. Merge the rules which use the same propositions as consequences so that one proposition acts as the consequence in only one rule. For example, the rule with (goal xplayer 100) printed above is merged from eight partial ones with (goal xplayer 100).

After rule grounding, all rules are in the form of
(<= (consequence) Func(condition1, condition2, condition3...)),

where consequence and conditions are keyword-propositions. Keywords in consequence include *role, init, next, legal, goal* and *terminal*, while *true* and *does* are the keywords in conditions. Particularly, *role*- and *init*-propositions depend on no propositions as conditions, *next*-propositions depend on *true*- and *does*-propositions and the remaining consequences only depend on *true*-propositions. Func is a logical function connecting conditions by *and, or* and *not*, which is called the *reasoning function* of the consequence.

After this phase, rules of equivalent games are normalized except the reasoning functions.

4.2 Graph Building and Mapping

In this phase, the grounded rules excluding the reasoning functions are modeled as a so-called ground graph, which is mainly a dependency graph of keyword-propositions. Thus, the number of enumerated bijections between propositions is determined by the number of isomorphisms between the graphs, which is much smaller than completely enumeration.

Definition 6 (Ground Graph). *A ground graph $G = (V, E, l)$ for grounded rules GR is a directed labeled graph with the following properties:*

- *($\forall p$, p is a keyword-proposition appearing in GR with a keyword k as its predicate) $p \in V$ and $l(p) = k$;*
- *($\forall n \in \mathbb{N}, n \in [0, 100]$) $n \in V$ and $l(n) = n$;*
- *($\forall v_s, v_t \in V, r \in GR, v_s$ is a condition of r and v_t is the consequence of r) $(v_s, v_t) \in E$;*

- $(\forall p^{init}, p^{true} \in V)\ (p^{init}, p^{true}) \in E$;
- $(\forall p^{next}, p^{true} \in V)\ (p^{next}, p^{true}) \in E$;
- $(\forall a^{does,r}, r^{role} \in V)\ (a^{does,r}, r^{role}) \in E$;
- $(\forall a^{legal,r}, r^{role} \in V)\ (a^{legal,r}, r^{role}) \in E$;
- $(\forall r^{goal,n}, r^{role} \in V)\ (r^{goal,n}, r^{role}) \in E$;
- $(\forall a^{does,r}, a^{legal,r} \in V)\ (a^{does,r}, a^{legal,r}) \in E$;
- $(\forall r^{goal,n}, n \in V)\ (r^{goal,n}, n) \in E$;

Here, p^{init}, p^{true}, p^{next}, $a^{does,r}$, r^{role}, $a^{legal,r}$ *and* $r^{goal,n}$ *express (init p), (true p), (next p), (does r a), (role r), (legal r a), (goal r n) respectively.* $(\forall n \in \mathbb{N}, n \in [0, 100])$ *represents all possible utilities in a valid GDL description.*

Thus, a ground graph has two types of nodes, which are proposition-nodes and integer-nodes. It also has two types of edges, which are logical-dependency-edges and consistency-maintaining-edges. It only reserves logical dependencies of propositions and discards reasoning functions. Figure 3 displays a brief structure of a ground graph.

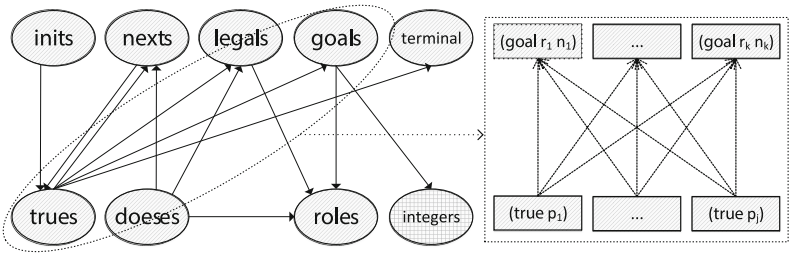

Fig. 3. Brief structure of ground graph. The solid ellipses stand for sets of nodes, while the solid rectangles stand for particular nodes.

After two ground graphs are built, they are tested for isomorphism. An isomorphism between directed labelled graphs $iso : V \leftrightarrow V'$ satisfies (1) $(\forall v \in V)\ l(v) = l'(iso(v))$; (2) $(\forall v_s, v_t \in V)\ (v_s, v_t) \in E \Leftrightarrow (iso(v_s), iso(v_t)) \in E'$.

Therefore, according to the definition of ground graph, an isomorphism between two ground graphs satisfies that (1) proposition-nodes map to proposition-nodes containing the same predicates and integer-nodes map to integer-nodes with the same value; (2) for two mapped propositions, their logically dependent propositions are also mapped; Since *init*-propositions are mapped, the initial states are equivalent; the consistencies between *next*- and *true*-propositions, *does*-, *legal*-, *goal*-propositions and *role*-propositions, *does*- and *legal*-propositions respectively are maintained; the mapped *goal*-propositions have the same utility.

After this phase, if an isomorphism is built, the two games may be equivalent. The remaining uncertainty is the reasoning functions of each proposition ,which is to be considered in the next phase.

Since a game may have symmetries [8], there may be several game equivalences between two games. In general, detecting one of them is sufficient for applications such as knowledge transfer. However, in this phase all isomorphisms of ground graphs need to be found, because any of them may cause an equivalence between games. Thus, for each isomorphism, the following phase is applied.

4.3 Logical Equivalence Verifying

In this phase, the unnormalized part of grounded rules, the reasoning functions, is handled.

By rule grounding, reasoning functions of all keyword-propositions are clear. By the last phase, mappings between keyword-propositions of two games are provided, so the reasoning functions are mapped in accordance. Moreover, propositions as conditions are also mapped. In other words, variables of reasoning functions are mapped. So, the actual problem is verifying the logical equivalence of two mapped logical functions, provided the consistent variable list. For example, there are two grounded rules (<= (p1) Func(p2, p3)) and (<= (q1) Func2(q2, q3)) of two games respectively, px maps qx respectively, then the problem is checking if Func(x, y) equals Func2(x, y).

For solving this problem, the naive approach that compares the truth tables of two logical functions takes exponential time. However, the problem can be transferred to the well-studied boolean satisfiability problem (SAT) to achieve a state-of-the-art efficiency. For example, testing whether logical functions f_1 and f_2 are equivalent can be transferred to testing whether $(((not\ f_1)\ and\ f_2)\ or\ (f_1\ and\ (not\ f_2)))$ is unsatisfiable. By using a SAT solver, the equivalence of two reasoning functions can be judged. So the remaining work is to verify the equivalence of all mapped reasoning functions in sequence with the SAT solver. Only if the verification is passed, the two games are equivalent and the SCGE σ can be obtained from the isomorphism of ground graphs.

4.4 Complexity

Let n be the number of reasoning functions, l the number of terms in the longest reasoning function.

For the first phase, the complexity is $O(nl)$, since the cost of grounding process is linear to the length of results.

The complexity of the second phase is at most NP-complete about n, since graph isomorphism is a special case of the NP-complete subgraph isomorphism problem [3].

The bottleneck is the third phase, which costs $O(m * n * NP - complete(l))$, where m denotes the number of maps generated by the second phase and $NP - complete(l)$ is the complexity of SAT problem [3].

The overall complexity is high. However, the approach is more efficient in practice than in theory.

Table 2. ConnectFour series games tested with their modified versions. Node No. and Edge No. express the scale of ground graph. Fun is the number of logical functions to be verified. Bij is the number of bijections generated by Phase 2 with heuristic grouping. Retry is the number of bijections verified by Phase 3 to find the first equivalence.

Game	Phase 1	Node No.	Edge No.	Phase 2	Fun	Bij	Retry	Phase 3
ConnectFour	0.107 s	217	2093	0.021 s	103	16	7	5.369 s
ConnectFourSuicide	0.100 s	217	2093	0.020 s	103	16	14	19.967 s
ConnectFourLarge	0.302 s	465	4717	0.144 s	223	64	5	8.986 s
ConnectFourLarger	1.841 s	1649	17533	9.322 s	807	1024	7	46.214 s

4.5 Improvements

There are several improvements which can be applied to the GEDA, listed by order of importance as follows.

Heuristically grouping nodes of ground graph. The number of isomorphisms of ground graphs can be huge. For example, the number of automorphisms of Tic-tac-toe's ground graph is 9!, since all 9 cells of the board are equivalent when discarding the information expressed by reasoning functions. However, the 9 cells can be grouped into 4 corner-cells, 4 border-cells and 1 center-cell by counting the numbers they are possible to form a line, which are 3, 2 and 4 respectively. This dramatically reduces the number of automorphisms to 4!4!1!. In general, analyzing the symmetry of the reasoning functions helps to group the elements of state space, so as the corresponding nodes of ground graph. Since the structure of reasoning function can be arbitrary, it is a heuristic grouping. However, it works for most occasions, because the symmetric structure is usually used by default.

Caching bad reasoning functions. Mapped reasoning functions have different possibilities to be equivalent for some reasons such as the different complexities. Caching the bad reasoning functions helps to prune early during verification.

Simplifying ground graph. Integer-nodes of ground graphs can be removed by adding a phase after graph mapping to verify the equivalence of utilities. Corresponding *true*- and *next*-proposition-nodes, *legal*- and *does*-proposition-nodes can be merged respectively. The *init*-proposition-nodes can be replaced by a single *init*-node.

Generating propositional net. Since grounded rules may need exponential space, it is more efficient to generate a propositional net and dynamically compute reasoning functions.

5 Evaluation

As introduced in Sect. 3, Tic-tac-toe is equivalent to Number Scrabble. The different part of Number Scrabble's rules is provided in Listing 1.2. The auxiliary propositions defined in Lines 3–4 represent the winning conditions. The *goal*- and *terminal*-propositions are dependent on the winning conditions. The state space

```
(sum15 1 5 9) (sum15 1 6 8) (sum15 2 4 9) (sum15 2 5 8)
(sum15 2 6 7) (sum15 3 4 8) (sum15 3 5 7) (sum15 4 5 6)
(<= (win x) (sum15 ?a ?b ?c) (true (cell ?a x)) (true (cell ?b x)) (true (cell ?c x)))
(<= (win o) (sum15 ?a ?b ?c) (true (cell ?a o)) (true (cell ?b o)) (true (cell ?c o)))
(<= (goal xplayer 100) (win x))
(<= (goal xplayer 50) (not (win x)) (not (win o)) (not open))
(<= (goal xplayer 0) (win o))
(<= (goal oplayer 100) (win o))
(<= (goal oplayer 50) (not (win x)) (not (win o)) (not open))
(<= (goal oplayer 0) (win x))
(<= terminal (or (win x) (win o) (not open)))
```

Listing 1.2. Partial rules of Number Scrabble

consists of (cell [1,9] {x,o,b}) and (control {xplayer,oplayer}), which is consistent with Tic-tac-toe. Therefore, the GEDA can work on it.

Table 3. Self-mapping numbers of some games. Brute Force enumerates all permutations of the elements of state space. GEDA+ stands for the GEDA with heuristic grouping. Goal stands for the number of symmetries of a game in nature.

Game	Brute force	GEDA	GEDA+	Goal
Tic-tac-toe	27!	9!	4!4!1!	8
Blocker	48!	16!	4!4!4!4!	4
Breakthrough	128!	2	2	2
Peg jumping	66!	8	8	8
Connect four	96!	8!	2!2!2!2!	2

By the phase of rule grounding, 68 grounded rules are generated for each game.

In the phase of graph building and mapping, two ground graphs are built. Each ground graph has 59 nodes with the improvement of graph simplification. To generate isomorphisms of them, NAUTYv2.5 [6] is applied. As mentioned above, 9! isomorphisms are found between them, which can be reduced to 4!4!1!2! by the improvement of nodes grouping. The 2! is caused by the permutation of the 2 groups with 4 nodes. In fact, enumerating these isomorphisms by an agent corresponds to repeatedly trying filling the numbers in the cells by human.

For the phase of logical equivalence verifying, MiniSAT [1] is applied as a SAT solver. Since MiniSAT only accepts inputs in Conjunctive Normal Form (CNF), Tseitin transformation [10] is used to transfer the logical functions to CNF.

As a result, the equivalence of Tic-tac-toe and Number Scrabble is detected by the GEDA in 9.73 s on average over 10 experiments, running on a laptop with an Intel i5 CPU.

Since game equivalence happens rarely in nature, some manual examples are tested. Four ConnectFour series games are modified with some logical conversions. Each game is tested if it is equivalent with its modified version by the GEDA with improvements. Table 2 shows the results.

In practice, the number of enumerated bijections primarily determines the running time of a game equivalence detection approach. Table 3 displays a comparison of the enumerated self-mapping numbers of some games using different approaches, which simulate the bijection numbers between equivalent games. It reveals that the performance of the GEDA is close to the optimal for some games, while for some other games it is still unsatisfactory.

Taking into consideration that it usually takes negligible time to reject inequivalent games, the GEDA has potential to be applied in real applications.

6 Conclusion

This work makes progress toward detecting game equivalence automatically by an agent. First, it discusses the classification of game equivalence and defines the SCGE, which covers more complex game equivalences than the previous works. Second, it proposes the GEDA, which solves the problem of detecting the SCGE by using a grounded rule graph and transferring the problem to well-studied problems to achieve state-of-the-art efficiency. It works well for some small games, while there is still room for further improvement.

This work benefits knowledge transfer between equivalent games, and can be easily extended to similar games by relaxing some conditions. Based on this work, solutions which standardize state spaces of equivalent games can be proposed for space-inconsistent game equivalence detection in the future.

References

1. Een, N., Sörensson, N.: MiniSat: a SAT solver with conflict-clause minimization. SAT 5 (2005)
2. Genesereth, M., Love, N., Pell, B.: General game playing: overview of the AAAI competition. AI Mag. **26**(2), 62–72 (2005)
3. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W., Bohlinger, J.D. (eds.) Complexity of Computer Computations. Springer, New York (1972)
4. Kuhlmann, G., Stone, P.: Graph-based domain mapping for transfer learning in general games. In: Kok, J.N., Koronacki, J., Lopez de Mantaras, R., Matwin, S., Mladenič, D., Skowron, A. (eds.) ECML 2007. LNCS (LNAI), vol. 4701, pp. 188–200. Springer, Heidelberg (2007)
5. Love, N., Hinrichs, T., Haley, D., Schkufza, E., Genesereth, M.: General game playing: game description language specification (2008)
6. McKay, B.D., Piperno, A.: Practical graph isomorphism, II. J. Symb. Comput. **60**, 94–112 (2014)
7. Pell, B.: Strategy generation and evaluation for meta-game playing. Ph.D. thesis, University of Cambridge (1993)
8. Schiffel, S.: Symmetry detection in general game playing. In: AAAI (2010)
9. Schiffel, S., Thielscher, M.: A multiagent semantics for the game description language. In: Filipe, J., Fred, A., Sharp, B. (eds.) ICAART 2009. CCIS, vol. 67, pp. 44–55. Springer, Heidelberg (2010)
10. Tseitin, G.S.: On the complexity of proof in prepositional calculus. Zapiski Nauchnykh Seminarov POMI **8**, 234–259 (1968)