PlayGame: A Platform for Diagnostic Games

Li Tan*

Department of Computer and Information Science, University of Pennsylvania
Philadelphia, PA 19104 USA
tanli@saul.cis.upenn.edu

Abstract. We introduce an integrated tool for implementing and playing various diagnostic games. The tool uses a *semantics hierarchy* introduced in [6] to improve code sharing among different diagnostic games and reduce the cost of introducing a new game. PlayGame synthesizes the winning strategy for a game using the evidence that is an abstract and uniform encoding of the proof computed by a checker, and hence instead of relying on a particular checker the tool works on a variety of checkers that can be extended to produce such an evidence. PlayGame implements a μ-calculus game and a full range of equivalence/preorder games on the Concurrency Workbench-New Century (CWB-NC).

1 Introduction

Games have been used in the verification community to model verification problems, to seek better solutions, and to understand verification results. The early work by Stirling [5] on bisimulation games and μ-calculus games unveils the potential of such games as diagnostic routines. In a diagnostic game the user competes with the computer to show that the verification result is incorrect. By losing each and every play to the computer the user is then convinced of the correctness of the verification result. A diagnostic game can provide valuable diagnostic information in an interactive way that a traditional diagnostic routine such as counterexample mechanism cannot. Individual efforts have been made to implement certain types of diagnostic games. The recent release of Edinburgh Concurrency Workbench [1] includes the support for a μ-calculus game and a strong bisimulation game. The verification tool Truth [3] also implements a μ-calculus game. These tools are designed to use some specific checker, mostly a game-based checker, to build the winning strategy for the computer. It is not known how diagnostic games can be built on top of other existing checkers. Another problem in diagnostic games is that with so many verification semantics, each of which requires different rules for game, defining and implementing them separately is a daunting task. We introduce PlayGame, a tool that provides a consistent interface for implementing new diagnostic games and incorporating new checkers. Figure 1 shows the architecture of the tool.

* This work was done when the author was a graduate student in State University of New York at Stony Brook

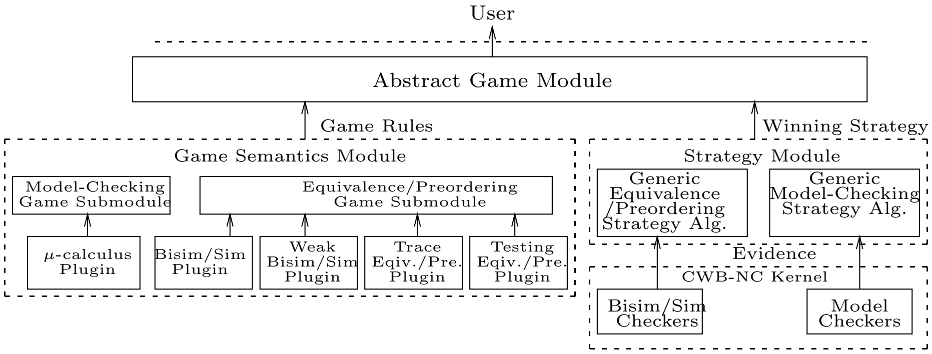

Fig. 1. The architecture of PlayGame

The tool is designed to use the evidence that is an abstract and uniform encoding of the proof constructed by a checker during the verification. Its precise definition is discussed in our previous work on evidence-based verification [8, 7], where we also discuss how various existing checkers can be extended to produce such evidence. Instead of relying on a particular checker, the tool works on any checker that can produce such evidence. To support different verification semantics, we introduce a *semantics hierarchy* [6] that abstracts game rules to different layers. To introduce a new game, one only needs to provide the semantic layers unique to the game. PlayGame implements a μ-calculus model-checking game and eight different eqivalence/preordering games including strong bisimulation/simulation games, weak bisimulation/simulation games, trace equivalence/preordering games, and testing equivalence/preordering games that cover all the equivalence/preordering semantics supported by Concurrency Workbench - New Century (CWB-NC)[2]. PlayGame also provides a consistent user interface for all the games that reduces the time required to learn a new game.

2 PlayGame

2.1 Designing PlayGame

The design of PlayGame reflects the game semantics hierarchy we defined in [6]. The abstract game module implements an abstract version of games and the features common to all games. The game semantics module defines the rules for each individual game. The strategy module synthesizes winning strategies from the evidences submitted by checkers.

Abstract Game Module A typical verification game has two players: player I, who insists a negative verification result, and player II, who believes otherwise. Each game has its own rules that must be *determined* in the following sense: if the correct answer to the verification problem is negative, then player I has a *strategy* to win each and every play no matter how player II moves; otherwise, player II shall have a winning strategy. When a game is used as a diagnostic routine, it involves two *sides*: the computer vs. the user. The computer assumes the role of a player in favor of the verification result. Thus, by losing each and every play to the computer, the user is convinced of the correctness of the verification result. The abstract game module implements the aforementioned abstract version of games. It introduces the role of a *referee* that enforces the rules supplied by the game semantics module. The module also implements the common functions including the bookkeeping and the user interface.

Game Semantics Module Game semantics module defines the rules for each game. For the games studied before such as μ-calculus model-checking games and strong bisimulation games [5], our definition is close to these previous results but also takes into account the human factor. Our revision intends to keep plays shorter and more informative. For example, the definition of a μ-calculus game by Stirling [5] requires two steps and the participant of a player to unroll a fixpoint expression ($\mu z.\Psi$ or $\nu z.\Psi$), while the choice of the player unrolling the expression is really irrelevant to the result. In our revised rules, it takes only one step and becomes part of the *referee*'s job. For those games that to the best of our knowledge have not been defined in literature such as testing equivalence/preordering games, we define the games based on the target verification semantics. To further improve code sharing we exploit the common aspects among each category of diagnostic games. For instance, in [6] we introduce a generalized equivalence game submodule and semantics plugins. To introduce an equivalence game, one only needs to supply a relatively small *plugin* that specifies only the part of rules unique to this particular game. In our experience abstracting game rules to different layers saves about 70-80% coding work when introducing a new game.

Strategy Module Synthesizing a winning strategy for the computer is the key to diagnostic games. Traditionally a winning strategy is constructed by a game-based algorithm [4]. In PlayGame the strategy is constructed from the uniformly-encoded evidence supplied by a checker. In [6, 8, 7] we propose uniformly-encoded evidences for different verification semantics: for equivalence games, it is a partition refinement tree; for preordering games the evidence is a Kernel-Auxilary partition refinement tree [6, 7], a variant of partition refinement tree in which each node contains a set of upper states (auxiliary set) in addition to a set of lower (and equivalent) states (kernel set); for model-checking games the evidence is a support set [8]. It turns out that constructing a winning strategy for the same category of games is fairly similar. For instance, in an equivalence game, the strategy is to keep the states of two processes in different leaves in the partition refinement tree, and hence the algorithm for computing a winning strategy for the computer is implemented across a category.

2.2 Using PlayGame

```
################## Round 2####################
Starting configuration:
  Agent1: 'out_easy.Strongjobber | 'out_easy.Strongjobber
  Agent2: (Start_easy | Start_easy | Hammer | Mallet)
          \ {geth,puth,getm,putm}
Referee: User goes first to choose an agent and make a transition.
 Which agent do you choose (1/2)?: 1
 Available user options:
  Option 0: --<'out_easy>-->'out_easy.Strongjobber | Strongjobber
  Option 1: --<'out_easy>-->Strongjobber | 'out_easy.Strongjobber
 Which transition do you choose?[Type (0-1), or (r)eview options]:1
Step 1: User chose agent 1 and made transition:
        --<'out_easy>-->Strongjobber | 'out_easy.Strongjobber
Step 2: Computer matchs user's choice by choosing
        the transition for agent 2:
        --<<'out_easy>>-->(Start_easy | Jobber
        | Hammer | Mallet)\ {geth,puth,getm,putm}
Continue game?[(c)ontinue, e(x)it or (b)ack]:c
```

Fig. 2. A sample round

PlayGame is implemented on the CWB-NC. To activate a game, the user simply issues a verification command with a special flag. The CWB-NC with PlayGame calls a checker and enters the interactive game mode after the verification ends. Referee declares the roles of the computer and the user based on the verification result, then the game proceeds by rounds. A sample round for a weak bisimulation game is given in Figure 2. The look and feel for other games are quite similar. The referee judges the winner with an explanation. If there is no winner yet, the referee decides how a round shall proceed. The user is prompted for his/her choice of the next move. The user may also choose to continue the play or to take back a few steps.

3 Conclusions and Future Work

We introduce PlayGame, an integrated platform for diagnostic games. Two novel features in its design are the use of semantics hierarchy, which enables code sharing among different games, and the building of winning strategy using checker-independent evidences, which makes it easier to incorporate new checkers. It implements a μ-calculus game and the full range of equivalence/preordering games. In future we want to study and implement the diagnostic game for CTL*.

References

1. The Edinburgh Concurrency Workbench. The University of Edinburgh, 1999.
2. R. Cleaveland and S. Sims. The NCSU concurrency workbench. In *Proceedings of CAV'96*, volume 1102 of *LNCS*, 1996.
3. M. Leucker and T. Noll. Truth/SLC — A parallel verification platform for concurrent systems. In *Proceedings of CAV'01*, volume 2102 of *LNCS*, 2001.
4. P. Stevens and C. Stirling. Practical model-checking using games. In *Proceedings of TACAS '98*, volume 1384 of *LNCS*, 1998.
5. C. Stirling. Bisimulation, model checking and other games. Notes for Mathfit Instructional Meeting on Games and Computation, University of Edinburgh, 1997.
6. L. Tan. An abstract schema for equivalence games. In *Proceedings of VMCAI'02*, volume 2294 of *LNCS*, 2002.
7. L. Tan. *Evidence-based Verification.* PhD thesis, State University of New York at Stony Brook, May 2002.
8. L. Tan and R. Cleaveland. Evidence-based model checking. In *CAV'02*, volume 2404 of *LNCS*, 2002.