A Model Checking Tool Based On Game Semantics and CSP

October 9, 2005

1 Introduction

The aim of this tool is checking safety properties such as assertion violations or array-out-of-bounds errors of open programs. The tool implements a Data-Abstraction Refinement Procedure to check for reachability of a specified move (*abort*) in the game model of a program. The tool compiles an abstracted program into a CSP process whose set of all its finite traces is the set of all plays of the game strategy for the program. The result process is then verified using CSP model checker – FDR. If no counter-example is found by FDR, the tool reports that the program is safe. If a genuine (deterministic) counter-example is found, the tool outputs the error trace. Otherwise, it uses the nondeterminism of the trace to refine the abstract program. Figure 1 shows the tool architecture, based on the abstract-check-refine loop.

2 Download and Installation

The tool can be downloaded as a tar.gz file from the web page. It needs JDK 1.4.2 (or later) version.

Unzip and untar the downloaded file.

tar xvfz gamechecker.tar.gz

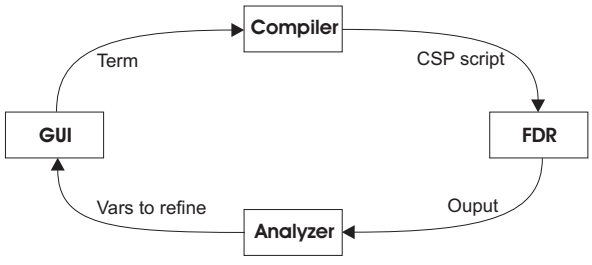

Figure 1: The tool architecture

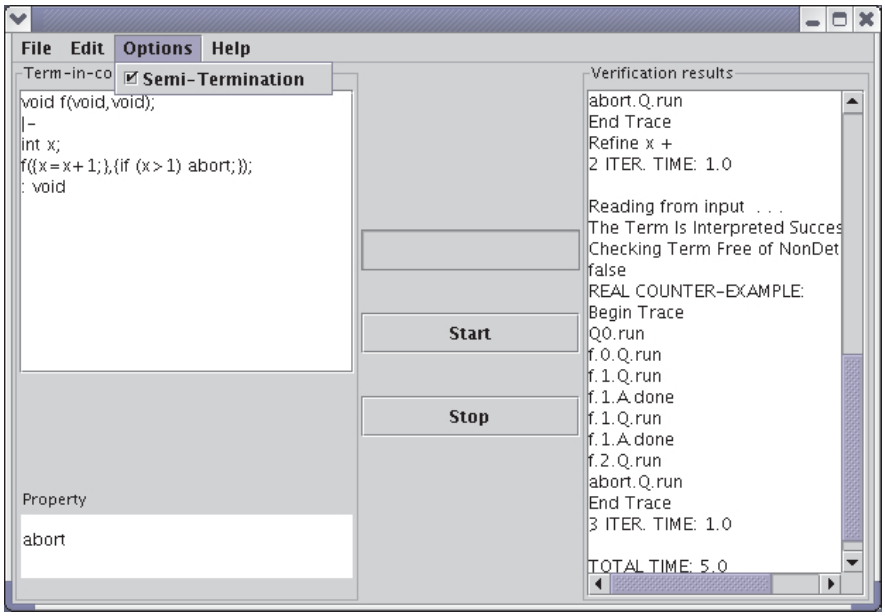

Figure 2: Screen-shot of the tool

Enter the *gamechecker* directory and run the tool.

$$java \ -jar \ gamechecker.jar$$

or

$$./gamechecker$$

The tool also requires the FDR model checker to operate correctly. Note that in order for our tool to use FDR, the executable for FDR must be in your current path.

FDR is a product by Formal Systems (Europe) Ltd. It can be downloaded without charge for academic use from their web page `http://www.fsel.com`.It is available for several architectures and operating systems.

3 A Tutorial Introduction

3.1 Graphical User Interface

Figure 2 presents the front-end of the tool developed in Java.

The inputs are an open program, and a property given by an error move whose reachability in the program model will be checked. The default error move is *abort*. The result pane shows all iteration steps in the procedure, including all reported nondeterministic unsafe plays and applied refinements. In the end,

if the procedure terminates, it is reported whether the term is safe or unsafe. In the latter case, a genuine unsafe play is returned. The tool allows the user to choose which variant of Abstraction Refinement Procedure (ARP) will be applied, the one which is guaranteed to terminate if the term is unsafe or a simpler one which analyses only the shortest nondeterministic unsafe play and which might diverge for unsafe terms. Both procedures might not terminate for safe terms. By default, the tool uses the simpler ARP. The semi-terminating ARP can be activated by checking the Semi-Termination check-box in Options menu.

4 Examples

A few examples come with the installation file. They illustrate the distinctive features of our method and help the user to get used to the language syntax.

4.1 A Simple Example

Consider the program

$$f(\mathsf{com},\mathsf{com}) : \mathsf{com} \ \vdash \ \mathsf{newint}\, x := 0 \,\mathsf{in}\, f(x := !x + 1, \mathsf{if}\, !x > 1 \,\mathsf{then}\, \mathsf{abort})$$

which uses local variable x, *non-local* function f, and a command abort which causes abnormal termination. We want to check whether this term is safe from terminating abnormally.

The procedure-call mechanism is by-name, so every call to the first argument increments x, and any call to the second uses the new value of x. So the program is not safe and this is how our tool will discover the bug.

The initial abstracted program is

$$f(\mathsf{com},\mathsf{com}) : \mathsf{com} \ \vdash \ \mathsf{newint}[]\, x := 0 \,\mathsf{in}\, f(x := !x + 1, \mathsf{if}\, !x > 1 \,\mathsf{then}\, \mathsf{abort})$$

The initial abstraction has only one value (nondeterministic choice over all integers), $[] = \{\mathbb{Z}\}$ where $\mathbb{Z} = \{n' \mid n' \in \mathbb{Z}\}$. An spurious (nondeterministic) unsafe play is identified by FDR, corresponding to the function that evaluates its second argument. Note that the nondeterminism in a trace is marked by a special move nd.

$$run\ run_f\ run_{f,2}\ nd\ abort$$

Next, we reconstruct the corresponding interaction ("uncovered") unsafe play:

$$\begin{array}{c} run\ run_f\ run_{f,2}\ run_{<1>}\ q_{<1,1>}\ q_{<1,1,1>}\ read_{<1,1,1,1>}\ read_x\ \mathbb{Z}_x \\ \mathbb{Z}_{<1,1,1,1>}\ \mathbb{Z}_{<1,1,1>}\ q_{<1,1,2>}\ 1_{<1,1,2>}\ true_{<1,1>}\ nd \end{array}$$

The refinement analyzer starts by examining the sub-term with coordinates $\langle 1,1\rangle$ (the guard of if) whose answer move precedes nd. The nd move marks that this term has been nondeterministically evaluated. Since this sub-term

represents a logic operation, the analyzer is recursively called to examine its operands, i.e. terms $\langle 1,1,1\rangle$ and $\langle 1,1,2\rangle$. The answer move of $\langle 1,1,1\rangle$ term is the abstract value $\mathbb{Z}$, so the examination proceeds for its sub-term $\langle 1,1,1,1\rangle$. Here, it will be detected that $\langle 1,1,1,1\rangle$ term corresponds to de-referencing of x and that the abstract value $\mathbb{Z}$ is read from x. Thus, it will be discovered that the variable x is involved in causing the *nd* move, and the analyzer will indicate that the abstraction of x needs to be improved.

The second iteration uses the refined term:

$$f(\mathsf{com},\mathsf{com}):\mathsf{com} \;\vdash\; \mathsf{newint}[0,0]\, x := 0 \,\mathsf{in}\, f(x := !x+1, \mathsf{if}\, !x > 1 \,\mathsf{then}\, \mathsf{abort})$$

Here, $[0,0] = \{<0, 0, >0\}$ where $<0 = \{n' \mid n' < 0\}$, $0 = \{0\}$, $>0 = \{n' \mid n' > 0\}$. Another spurious unsafe play is found, which represents the function evaluating its first and then its second argument.

$$run\ run_f\ run_{f,1}\ done_{f,1}\ run_{f,2}\ nd\ abort$$

The corresponding interaction play is:

$$\begin{array}{c} run\ run_f\ run_{f,1}\ run_{<1>}\ q_{<1,2>}\ q_{<1,2,1>}\ read_{<1,2,1,1>}\ read_x, 0_x\ 0_{<1,2,1,1>} \\ 0_{<1,2,1>}\ q_{<1,2,2>}\ 1_{<1,2,2>}\ 1_{<1,2>}\ write.1_{<1,1>}\ write(>0)_x\ ok_x\ ok_{<1,1>} \\ done_{<1>}\ done_{f,1}\ run_{f,2}\ run_{<2>}\ q_{<2,1>}\ q_{<2,1,1>}\ read_{<2,1,1,1>}\ read_x, (>0)_x \\ (>0)_{<2,1,1,1>}\ (>0)_{<2,1,1>}\ q_{<2,1,2>}\ 1_{<2,1,2>}\ true_{<2,1>}\ nd \end{array}$$

Similarly as in the previous iteration, the analyzer starts exploring the nondeterministic term $\langle 2,1\rangle$. Searching for abstract values recursively through its sub-terms, it will be detected that the abstract value (>0) read from x has caused the nondeterminism. So, further refinement of x will be recommended.

The third iteration program is:

$$f(\mathsf{com},\mathsf{com}):\mathsf{com} \;\vdash\; \mathsf{newint}[0,1]\, x := 0 \,\mathsf{in}\, f(x := !x+1, \mathsf{if}\, !x > 1 \,\mathsf{then}\, \mathsf{abort})$$

Now, a genuine unsafe play is detected: function f increments x two times then evaluates its second argument.

$$run\ run_f\ run_{f,1}\ done_{f,1}\ run_{f,1}\ done_{f,1}\ run_{f,2}\ abort$$

4.2 A Semi-Termination Example

The program

$$\begin{array}{l} e:\mathsf{int}, f(\mathsf{com},\mathsf{com}):\mathsf{com} \;\vdash \\ \mathsf{newint}\, x := e \,\mathsf{in}\;\; \mathsf{newint}\, y := 0 \,\mathsf{in} \\ \quad \mathsf{if}\ (!x == !x+1)\ \mathsf{then}\ \mathsf{abort};\;\; \mathsf{else}\ \mathsf{f}(y := !y+1, \mathsf{if}\ (!y == 0)\,\mathsf{then}\,\mathsf{abort}); \end{array}$$

is a simple example of unsafe program that will cause the simpler ARP which analyses only one spurious unsafe play to diverge. Namely, FDR will always report the shortest counter-example with the first occurrence of the nondeterminism in the guard of if statement. So, the abstraction of x will be refined

forever. But, when the ARP analyses the spurious unsafe play with minimal rank value, then after few iterations refining x, the abstraction of y will be refined and a genuine counter-example will be found. For this particular example, e and x are refined to $[-2, 1]$ and y to $[0, 1]$ before the real counterexample is detected.