

Construction of linear temporal property verification extension

Abstract

This project is a collaboration with "LTL verification in JPF" project, by Ewgenij Starostin, mentored by Franco Raimondi. Ewgenij implements double depth first search algorithm. In this project, we implement the other parts and the key is to allow combining symbolic execution with the search algorithm.

We will implement 1) LTL and atomic parsers; 2) Implement a listener to combine with DDFS Search object (by Ewgenij's project); 3) Extend ChoiceGenerator and override setCurrentPC() which check for satisfiability of the path condition and the guard constraint; 4) Use Choco solver for subsumption checking. In addition if time is available we will visualize Buchi automata using Jung and write a LTL editor right in eclipse, which has syntax highlight and intellisense.

Contact

students: Dinh-Phuc Nguyen <luckymanphuc@...> <PND> and Chung-Tuyen Luu <tuyenlc52@...> <TLC>

mentor: Hoang Truong <email> <HT>

co-mentor:

Program & Timeline

This project is funded by the [Google Summer of Code \(GSoC\)](#) program. It follows the [GSoC timeline](#) (start 05/24, finish 08/16)

- 24/5 – 15/6: T1, T4 (See below for T1-T6)
- 15/6 – 5/7: T2 and T3
- 5/7 – 12/7: Review code; Fix bugs; Make some tests
- 12/7 – 2/8: T5, T6
- 2/8 – 16/8: Make tests; Fix bugs; Finalize the product and documentation.

Repository

The sources for this project are available from a Mercurial repository on <http://bitbucket.org/>

Description

Our planned tasks include:

T1. Construct a LTL formula parser and an atomic proposition parser

This task includes writing a LTL grammar for automatically generating lexer and parser by ANTLR. After parsing a LTL formula, we obtain an instance of Formula<String> and extract atoms from that formula. An atom is expressed as a condition statement in Java language. Variables in atom may be instance fields or method local variables. Also, method calls can appear in the atom and can be understood as whether a method is invoked. However, a variable or method call must include package name (if any), class name.

T2. Implement a listener to combine with DDFS Search object

This listener gets the LTL formula from annotation when classes loaded then construct negated Buchi automata. It is also used to check the satisfiability of a run sequence with the product Buchi automata. This task requires extracting the symbolic value of fields and local variables. A symbolic value is an instance of Expression and is stored in the attribute of the field and local variable. Fields are stored in the heap and local variables are stored in the stack frames. We will check whether executing an instruction violates the transition guard. We treat every execution step as a branch point. Suppose that we have m successors for a given state and in this state there are n allowed choices by symbolic execution, then we have m*n next candidate states. We use ChoiceGenerator to handle the branching purpose.

T3. Extend ChoiceGenerator

We store the guard constraint in the ChoiceGenerator and override the setCurrentPC() method. Whenever setCurrentPC() is called, it takes the conjunction of path condition and the guard constraint and checks for the satisfiability.

T4. Use Choco solver to check for subsumption checking

Currently, we can check state subsumption using Omega native library. But this library only works in Linux, we will consider building a version for Windows. Or we will try to use newer version of Choco solver to check for state subsumption used in state matching.

T5. Visualize Buchi automata using Jung

T6. Write a LTL editor (Optional)

Make syntax highlight and intellisense for LTL editor right in Eclipse.

Project Blog

(most recent on top)

2010-05-26 (NR) - details added based on HT's email.

2010-05-07 (NR) - project page created