DIVINE

Characteristics

- Keywords: fast, reliable, general purpose, easy-to-use
- Useful for verification of large systems (as opposed to sequentional model checkers - problems with space explosion) - uses effective space-reduction techniques (Partial Order Reduction, Path Compression)
- Supports implementations of a majority of POSIX thread APIs (pthread.h) - enables verification of multithreaded programs
- Model checking of LTL properties, also suitable for a model checking algorithms development or experimental comparisions/evaluations.

Characteristics II

- Verification of models in DVE language
- Verification via LLVM
- UPAAL timed automata
  - LTL model checking
  - Deadlock detection
  - Implements Upaal Time Automata Parser Library, DBM library and interpreter for timed automata
  - Accepts .xml
- MurPHI models
  - Implemented compiler that generates native code
  - Distributed and parallel state-space analysis, deadlock detection

Characteristics – state space compression

- Lossless
- Based on tree compression
  - Effective on large models
- Can achieve 90% compression ratio
- Time of comprimation is negligible
- Supported by all algorithms
- --compression or –compression=tree
- Also supported in parallel verification, but it's recommended to use shared memory

Characteristics – Windows version

- From 2.1 version provides GUI
- Windows version supports parallel systems with shared memory only
- Doesn't support verification of C/C++ code via LLVM

GUI

- Unix version – uses Qt
- One click verification
- Graphic simulator (counterexample generation)
- Graphic debugger (variables check, steps, random run)

Installation I - requirements

- HW: 2GB disk space, at least 4GB RAM
- For Windows: 32bit system, MinGW compiler, CMake
- For Unix:
  - GNU C++ (4.7.3) or clang (3.2)
  - Cmake
  - Other:
    - LLVM (3.2)
    - Clang (3.2)
    - Qt (4.5) – GUI
    - Boost
    - libxml2
    - Pandoc (+ pdflatex/bibtex)
    - MPI (OpenMPI)
    - flex, byacc

Installation II

- Extract tar to a folder
- ./configure
  - Check compatibility
- make
- make check
- make install

Commands

- divine - -version
  - Displays list of options available in installed version of the program (depends on which plugins it was compiled with)

- divine info <model>
  - Displays information about given model - list of properties that can be checked (you can use this information in other commands) - for example deadlock, assert or LTL

Commands II - combine

- divine combine [-f <formula file>] <model file>
- Some languages have inner support of LTL properties - in this case are verified properties available automatically (verify --property). If you verify DVE model, is necessary to specify LTL properties in separate .ltl. file
- divine combine [-f formula.ltl] [-p N] [-o] [-q] model.dve
- divine combine [-f ...] [...] model.mdve [P1=VAL] [P2=VAL] ..
- Combine command translates LTL on Büchi automata and includes it in DVE file. For every LTL property is created one separate .dve file

LTL - overview

- Model checking using linear temporal logic formulas
- We create a formal model M of a given system (system is a set of infinite runs), that we want to verify and the subject of verification we express using LTL formula
- We express φ using LTL and decide if M |= φ (e.g. if M is model of φ)
- 2 different runs are equal if their interpretation of atomic propositions matches
- φ is evaluated over one run and express validity of atomic propositions in states of a run

LTL – overview - operators

- Fφ - (future) – somewhere in the run φ is valid
- Gφ - (globally) – φ in valid during whole run
- φUΨ - (until) – somewhere in the run Ψ is valid, and until then φ is valid
- Xφ - (next) – in the next state φ is valid
- φWΨ - (weak until) – like „until“, but Ψ doesn't necessarily becomes valid
- φRΨ - (release) - Ψ is valid until (Ψ AND φ) is valid, after that none of those is valid (+also (Ψ AND φ) does not necessarily have to become valid)

LTL - syntax

```
#define at1 (Proces1.vStaveX)
#define at2 (Proces2.vStaveY)
#define at3 (premenna1 == 100)

#property F (at1 && at2)
#property G at1
#property !F at3
```

- #define – assigns symbolic name for atomic proposition
- #property – specifies an LTL formula

Commands III - metrics

- divine metrics <flags> <model>
- [--reduce=R]
- [--no-reduce]
- [--fair]
- [--report[=<report format>] | -r]
- [--property=N]
- [engine options]

- Determines state availability on the whole state space of a given model

- Prints out statistics - number of states, transitions, accepting or deadlocks

Commands IV - draw

- divine draw
- [--distance=N]
- [--trace=N,N,N...]
- [-l|--labels|--trace-labels]
- [--bfs-layout]
- [--reduce=R]
- [--no-reduce]
- [-f|--fair]
- [--render=<cmd>|-r <cmd>]
- [--compression]

Commands V - verify

- divine verify <flags> <model>
- [--reachability|--owcty|--map|--nested-dfs] – in case we want to use a specific algorithm (otherwise there is automatically chosen a suitable algorithm for model checking, according to type of a property)
- [--property=<name> | -p <name>] - specifies which property we want to check. We can use divine info to display a list of available properties
- [--fair] – accepts only weakly fair runs. For now available only for DVE models, suitable when using LTL

Commands V – verify cont.

- [--reduce=<reduction>] - forces usage of heuristics that ensmalls state space.
- [--report[=<report format>] | -r] – generates report, format: text, text:file, plain, etc.
- [--no-counterexample | -n] – forbids generating of counterexamples
-
- [--display-counterexample | -d] – forces generating of counterexamples
- [engine options] – *undocumented!*

Commands VI - gen-explicit

- divine gen-explicit
- [--fair]
- [--reduce=<reduction>]
- [--report[=<report format>]]
- [engine options]
- [--no-save-states]
- [-o <file> | --output=<file>]

- Generates states space of a model into a file that can later be used by DIVINE or other tool capable of working with DIVINE Explicit Space Format

LLVM

- Low Level Virtual Machine
- Infrastructure for compiler (libraries and interfaces)
- Written in C++
- Used by Clang, and many other compilers for various languages (Python, Haskell). Clang (but also GCC with plugins)can generate optimized and unoptimized bitcode
- Supports life long compilation model, including link-time, install-time, run-time

DIVINE a LLVM

- You can use any code written in C/C++
  - However, DIVINE has problems with I/O operations
- Compiles with divine compile –llvm prog.c
- That will create whole runtime environment – prog.bc
- You can use divine info (and take a look at available properties)
- Or divine metrics (number of states, transitions, accepting, deadlocks)
  - Keep in mind that deadlock in DIVINE is different than deadlock in C. Deadlock in DIVINE can be also a state without a successor.

DIVINE a LLVM

  - Turning off the reductions – increases number of states and transitions (reductions can be very demanding of resources)

- Verification – divine verify prog.bc -p <property>, for example assertion etc

- Also with -d

- With multithreaded programs, DIVINE checks all thread interactions systematically, on a bitcode instructions level. That enables to prove absence of deadlock/assertion violations, which is impossible with standard techniques

DVE I

- Formalism for asynchronous systems modelling - protocols.
- Partially derived from a modelling languages used for Uppaal, but is mre concentrated on a model expressibility that comfortable modelling
- Creates an abstrack model - automata, resp. a set of extended finite automatas
- Synchronous/asynchronous mode - synchronous not yet supported

DVE II - syntax

- Basic unit - process, which consists of variables, transitions, states,...
- Global/local variables - shared between processes
- Communication via channels (typed/untyped, buffered/unbuffered)
- Transitions - sync, guard, effect
- 2 data types - int, byte and also one dimensional arrays
- Commited states, assertions

DVE II – syntax example

```
int glob_var = 0;

process P {
int loc_var = 0;
state s1, s2, s3;
init s1;
trans
s1 -> s1 {guard glob_var<3; effect loc_var = loc_var +1;},
s1 -> s2 {effect glob_var = glob_var +1;},
s2 -> s3 {};
}

system async;
```