SMTpp: preprocessors and analyzers for SMT-LIB

Richard Bonichon [1] David Déharbe [2] Pablo Dobal [3]
Cláudia Tavares [2]

CEA, LIST, Software Reliability Laboratory, Gif-sur-Yvette, France

DIMAp, Universidade Federal do Rio Grande do Norte, Natal, RN, Brazil

INRIA, Université de Lorraine & LORIA, Nancy, France

July 18th, 2015

Introduction

SMTpp = SMT pre-processor

What? source-to-source transformer and analyzer for the SMT-LIB

Why?

- SMT developer: factor simplifications found in solvers ≈ 20 % in veriT
- SMT developer: fuzzing-based tests, delta-debugging
- SMT user: factor SMT configurations from generators of verification conditions
- SMT-LIB: analyse and classify benchmarks
- SMT-COMP: scramble benchmarks

How? high-level programming language, symbolic programming (OCaml+menhir)

Introduction

SMTpp = SMT pre-processor

What? source-to-source transformer and analyzer for the SMT-LIB

Why?

- SMT developer: factor simplifications found in solvers $\approx$ 20 % in veriT
- SMT developer: fuzzing-based tests, delta-debugging
- SMT user: factor SMT configurations from generators of verification conditions
- SMT-LIB: analyse and classify benchmarks
- SMT-COMP: scramble benchmarks

How? high-level programming language, symbolic programming (OCaml+menhir)

Prototype release available from
`http://www.verit-solver.org`, under ISC licence.

Positioning SMTpp

- SatELite: a pre-processor for SAT
  - minimization of CNF
  - target a class of SAT encodings
- jSMTLIB: SMT-LIB for Java
  - Java-based
  - SMT-LIB parser and type-checker
  - interaction with SMT-LIB and SMT-solvers
    - Eclipse plug-in
    - API
- SMT solvers ($\supset$ SMTpp)
  - parsing
  - type inference
  - logic detection
  - simplification
- delta debugger, scrambler, fuzzer

AST

- one-to-one mapping from grammar to OCaml data type
- excerpt (commands)

```
type command_desc =
  | CmdAssert of term
  | CmdCheckSat
  | CmdDeclareConst of symbol * sort
  (* ... *)

type command = {
    command_desc : command_desc;
    command_loc  : Locations.t;
  }
```

- Full code available at:
  `https://github.com/ossanha/smtpp`

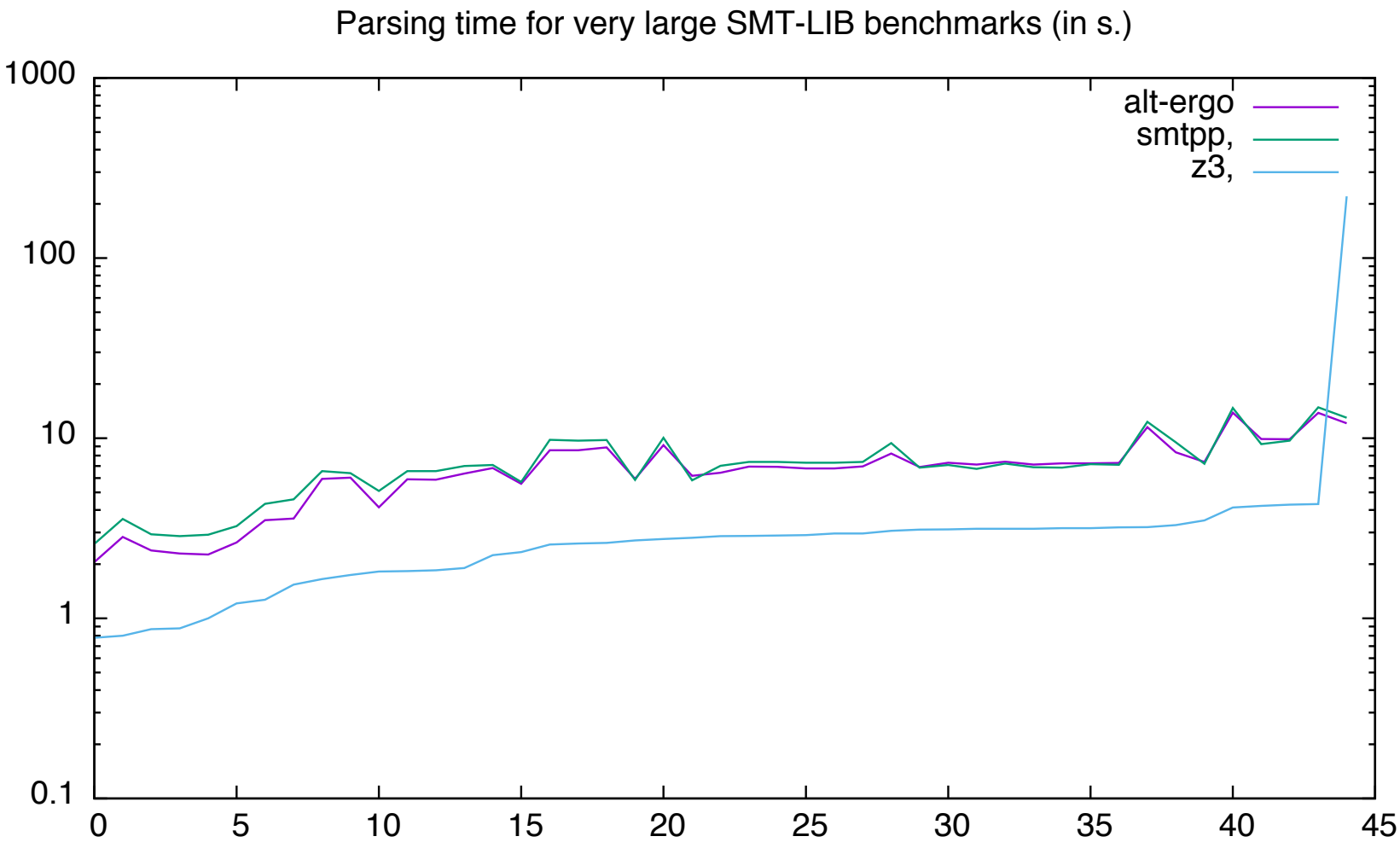
The parser module

- ocamlyacc or Menhir
  - LR(1) parser generator
  - keyword references to semantic values
  - improved debugging support

```
%inline command:
  | ASSERT t=term;
  { let loc = mk_loc $startpos $endpos in
    mk_command (CmdAssert t) loc }
  | CHECKSAT
  { let loc = mk_loc $startpos $endpos in
    mk_command CmdCheckSat loc }
  | DECLARECONST sym=symbol; so=sort;
  { let l = mk_loc $startpos $endpos in
    mk_command (CmdDeclareConst(sym, so)) l }
```

```
let mk_loc loc_start loc_end =
    { loc_start; loc_end; } ;;
```

Parser: experiments



Obfuscator module

- replace declared symbols with tool-generated names
- example input (excerpt):

```
(declare-fun x () Real)
(declare-fun f (Real) Real)
(assert (=> (> x 0) (< (f x) 0)))
```

- corresponding output:

```
(declare-fun S2 () Real)
(declare-fun S3 (Real) Real)
(assert (=> (> S2 0) (< (S3 S2) 0)))
```

Multi-script generation module

- Slice incremental benchmarks into non-incremental ones.

```
(set-logic QF_LIA)
(declare-fun w () Int)
(declare-fun x () Int)
(declare-fun y () Int)
(declare-fun z () Int)
(assert (> x y))
(assert (> y z))
(push 1)
(assert (> z x))
(check-sat)
(pop 1)
(push 1)
(check-sat)
(exit)
```

Multi-script generation module

- Slice incremental benchmarks into non-incremental ones.

```
(set-logic QF_LIA)
(declare-fun w () Int)
(declare-fun x () Int)
(declare-fun y () Int)
(declare-fun z () Int)
(assert (> x y))
(assert (> y z))
(assert (> z x))
(check-sat)
(exit)
```

```
(set-logic QF_LIA)
(declare-fun w () Int)
(declare-fun x () Int)
(declare-fun y () Int)
(declare-fun z () Int)
(assert (> x y))
(assert (> y z))
(check-sat)
(exit)
```

Use-def analysis module

- detects:
  1. use of undefined symbols: error
  2. definition of never-used symbols: noise
     - commonplace in SMT-LIB
- results verified with grep
- application
  - simplify benchmarks
  - possible errors

Use-def analysis: experiments

- BWare project: prelude with symbols not used in all files
- a number of scripts have local bindings (`let`) for variables that are not used
- all benchmarks in LRA/keymaera have a homonymous global constant symbol and existentially quantified variable.
- QF_ABV has scripts with significant amounts (30-50% file size) of unused declarations.

Logic detection module

- Errors occur when logic is not precise enough
- Logic is too restrictive:
  - reduce choice of solvers
  - might reduce performance
  - might reduce precision of results (e.g. `unknown` vs `sat`)
- Benchmark generators often choose a default logic that is sometimes too expressive
- Approaches in existing solvers
  - have a default catch-all logic
  - inspection of AST to infer a logic

Logic detection module

- Errors occur when logic is not precise enough
- Logic is too restrictive:
  - reduce choice of solvers
  - might reduce performance
  - might reduce precision of results (e.g. `unknown` vs `sat`)
- Benchmark generators often choose a default logic that is sometimes too expressive
- Approaches in existing solvers
  - have a default catch-all logic
  - inspection of AST to infer a logic $*$
- $\Longrightarrow$ logic detector module in SMTpp.
  - Still conservative: e.g. LA benchmarks classified as NLA
  - DL: problems with non-canonical atoms, e.g. $x \leq y + 1$

Logic detection: experiments

- SMT-LIB (2014 version): 5% reported by SMTpp

  AUFLIRA 1368 scripts without `Array`.

  UF/grasshopper propositional

  QF_UFLRA 20% do not use arithmetics

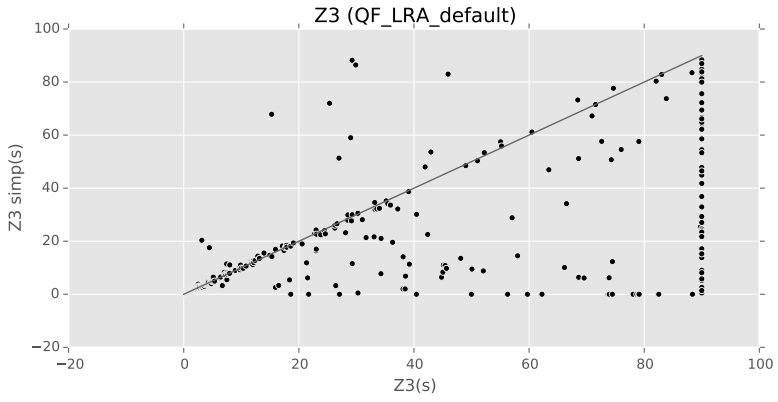
  QF_AUFLIA mostly do not use arithmetic functions, 1/4 have no free sorts and uninterpreted functions.

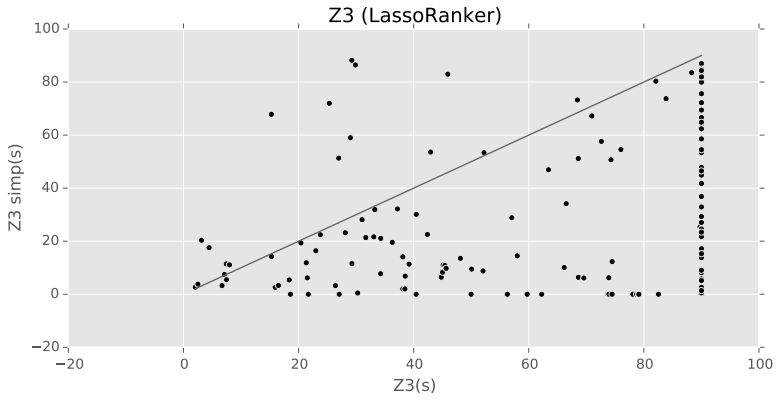
LA simplifications module

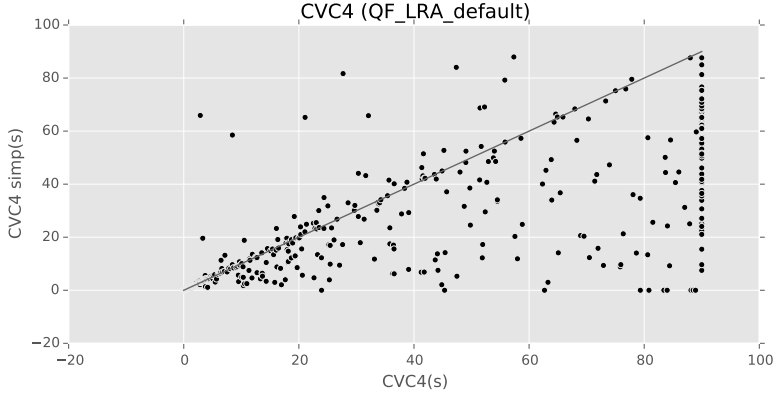
- motivated on LassoRanker subfamily
- straigthforward transformations
  - absorbing element
    - multiplication
    - logical and
    - logical or
  - neutral element
    - multiplication
    - addition
  - turn relational constraints into boolean constants (i.e. $1 > 0$)
- Local Gauss elimination
  - identify equational constraints
  - split additive expression
    - replacing monoms
    - monoms to be replaced
  - perform substitution on relational constraints

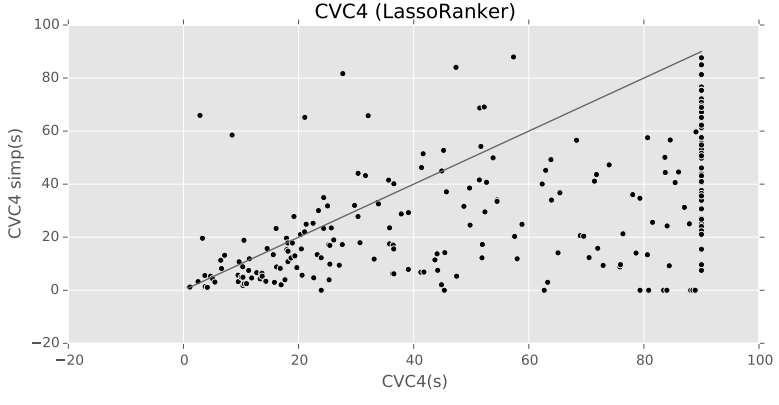
LA simplifications: experiments

Comparison of execution times



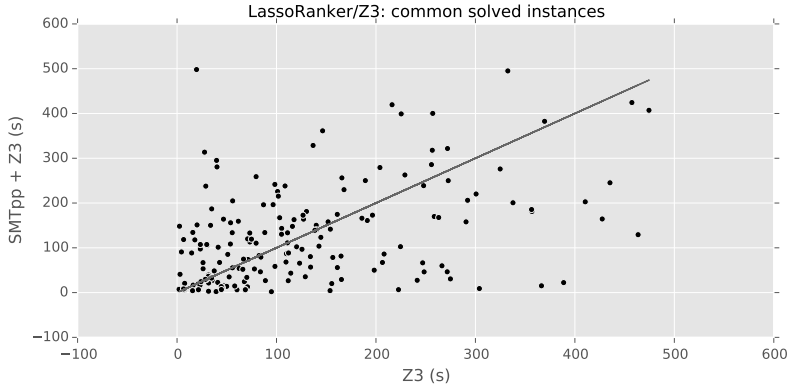


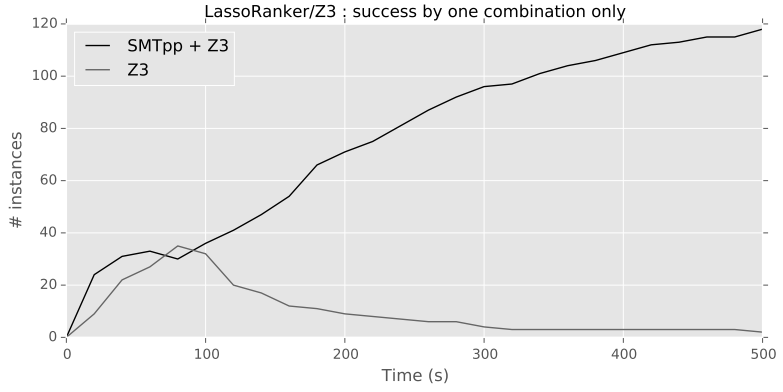


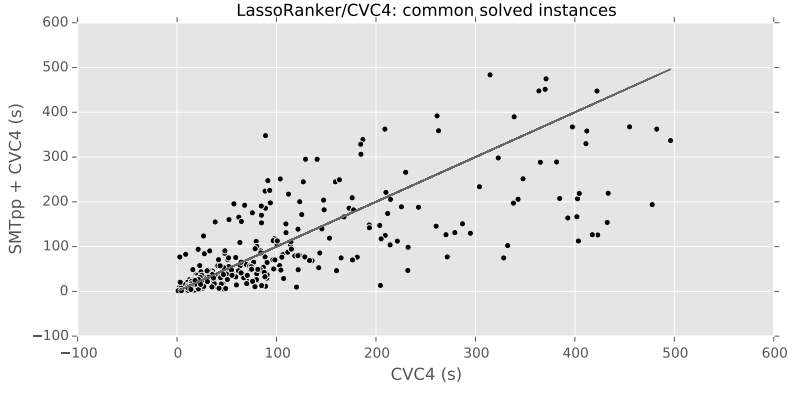


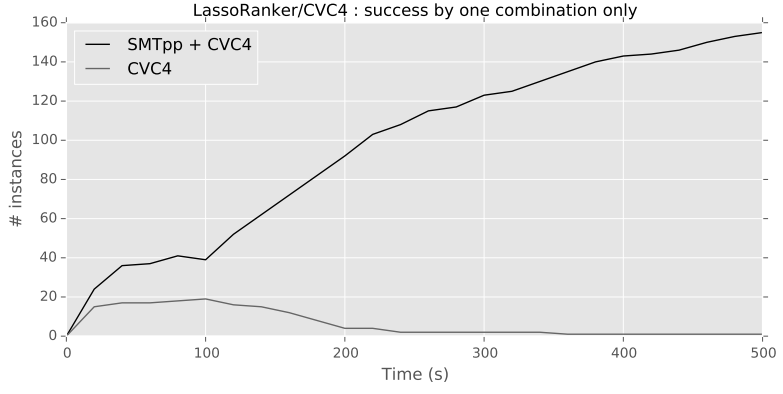
LA simplifications: experiments

Execution time on solved instances and count of uniquely solved instances



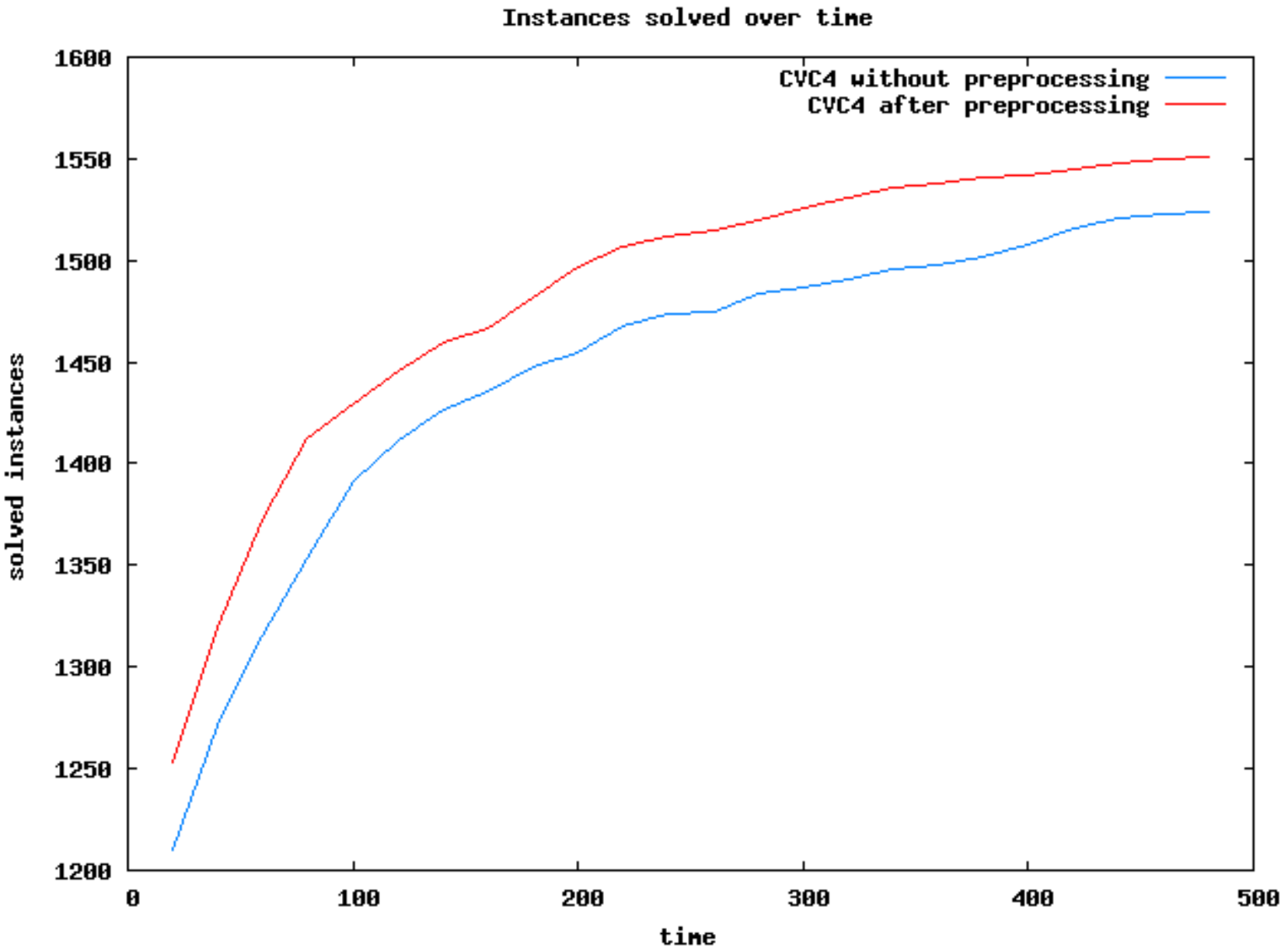






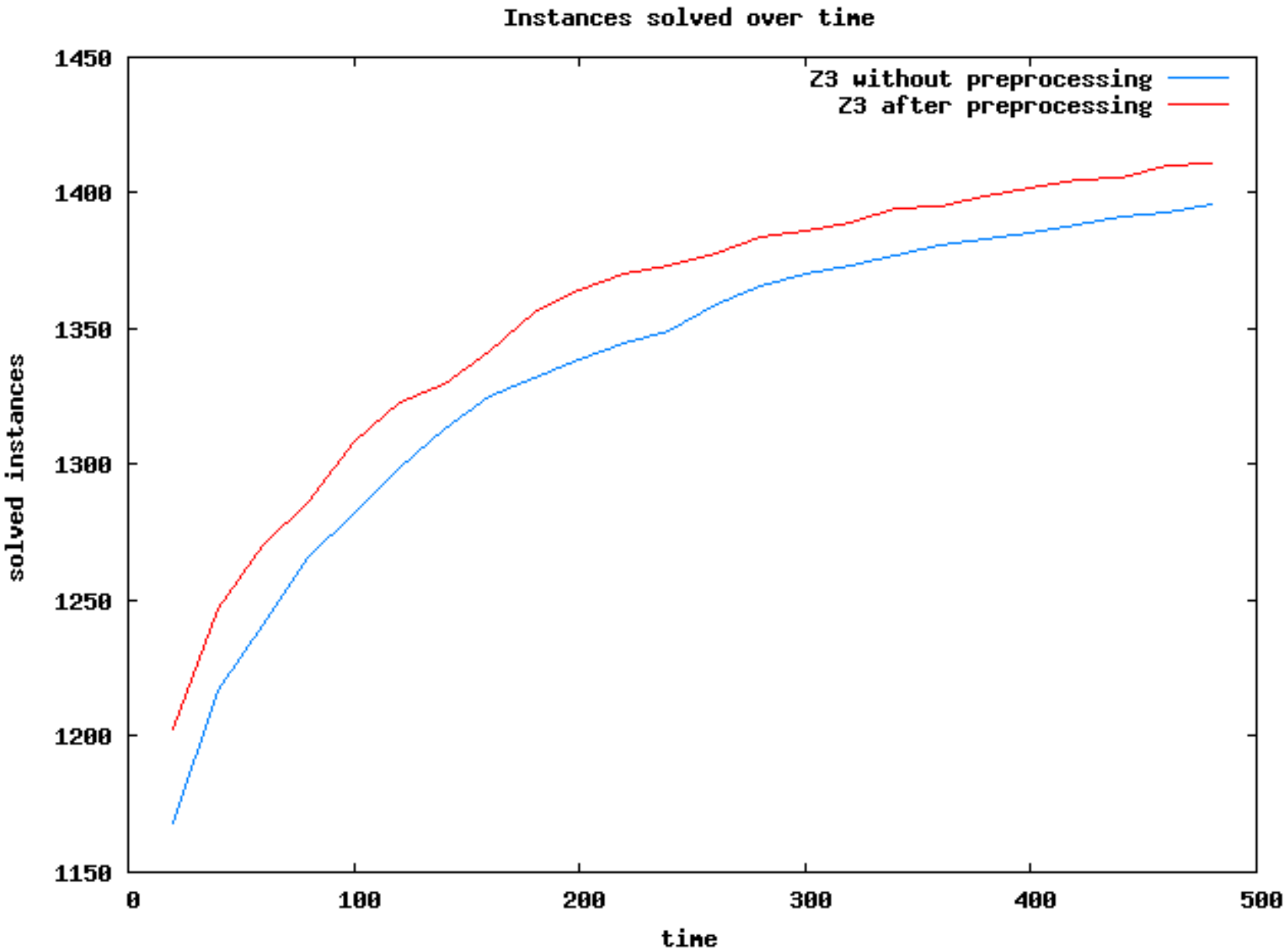
LA simplifications: experiments

Cumulated amount of solved instances: CVC4/smtpp+CVC4



LA simplifications: experiments

Cumulated amount of solved instances: Z3/smtpp+Z3



LA simplifications: user experience

- The implementer of this module was initially completely unfamiliar with OCaml.
  - 3-month experience with Scheme
- One day to get familiar with SMTpp code
  - core knowledge: AST API
- A prototype version of the module was implemented in two weeks
  - quick-and-dirty mode
  - more effort needed to make it stable

Conclusions

- SMTpp: already available
  - supports SMT-LIB 2.5
  - implemented several features useful to clients of SMT solvers
- preliminary results cross-checked with other tools
- usability:
  - permissive free software license
  - high-level programming language
- `https://github.com/ossanha/smtpp`

Future work

- type checking
- architectural challenges
  - processing of very-large benchmarks and memory usage (multiple-passes?)
  - integration with SMT-solvers (quantifier instantiation)
  - pluggable architecture: plug-ins *à la* Frama-C
  - tactic language: user-programmable combination of simplifications?
- handle requirements such as: traceability, proof production
- additional simplifications, rewriting:
  - symmetry detection and symmetry breaking
  - if-then-else simplifications
- extend to support delta-debugging, scrambling

Feedback

Questions?

`https://github.com/ossanha/smtpp`

License

Essentially equivalent to BSD license.

License

Essentially equivalent to BSD license.

Copyright (c) Year(s), Company or Person's Name ¡E-mail address¿ Permission to use, copy, modify, and/or distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies. THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.