Proofs in cvc5: New Directions with AletheLF

Andrew Reynolds

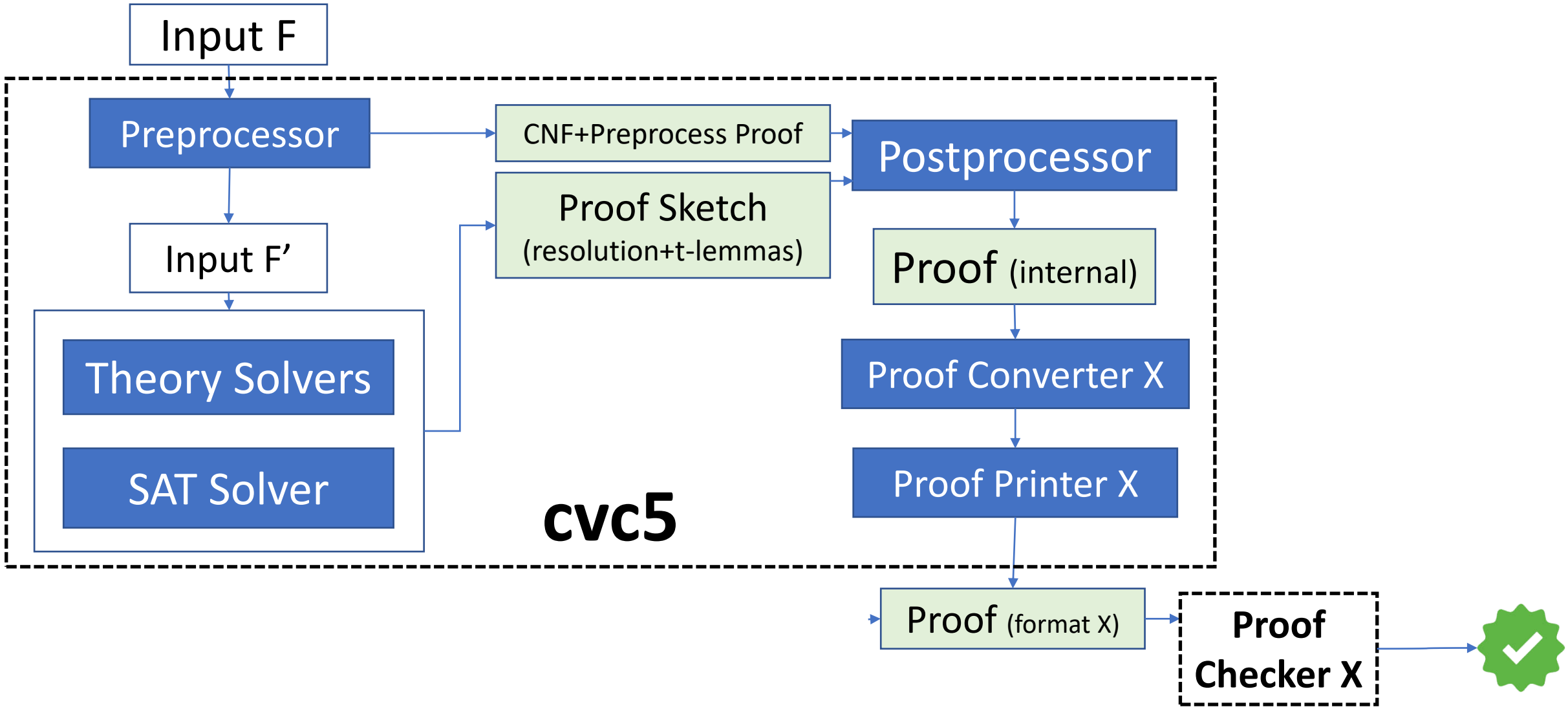
October 20, 2023

Outline

- Previous work on cvc5's proof architecture
- Introducing: AletheLF (ALF) proof format
  - Standard format for proofs from cvc5
- alfc: a proof checker for ALF
- Preliminary results and future work

cvc5’s internal proof architecture

Architecture of cvc5 + Proofs

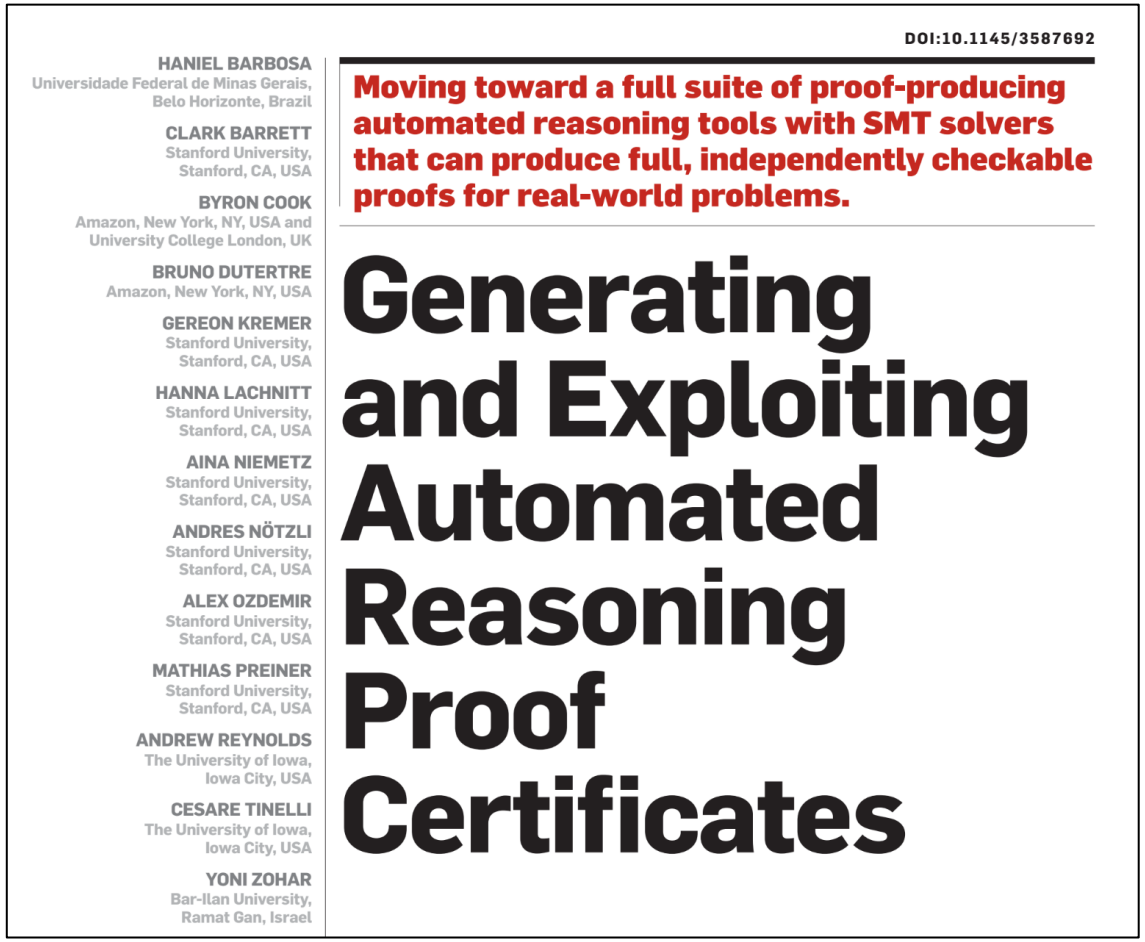


cvc5 + Proofs

Flexible Proof Production in an Industrial-Strength SMT Solver*

Haniel Barbosa[1], Andrew Reynolds[2], Gereon Kremer[3], Hanna Lachnitt[3], Aina Niemetz[3], Andres Nötzli[3], Alex Ozdemir[3], Mathias Preiner[3], Arjun Viswanathan[2], Scott Viteri[3], Yoni Zohar[4], Cesare Tinelli[2], Clark Barrett[3]

[1] Universidade Federal de Minas Gerais, Belo Horizonte, Brasil
[2] The University of Iowa, Iowa City, USA
[3] Stanford University, Stanford, USA
[4] Bar-Ilan University, Ramat Gan, Israel



DOI:10.1145/3587692

HANIEL BARBOSA
Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

CLARK BARRETT
Stanford University, Stanford, CA, USA

BYRON COOK
Amazon, New York, NY, USA and University College London, UK

BRUNO DUTERTRE
Amazon, New York, NY, USA

GEREON KREMER
Stanford University, Stanford, CA, USA

HANNA LACHNITT
Stanford University, Stanford, CA, USA

AINA NIEMETZ
Stanford University, Stanford, CA, USA

ANDRES NÖTZLI
Stanford University, Stanford, CA, USA

ALEX OZDEMIR
Stanford University, Stanford, CA, USA

MATHIAS PREINER
Stanford University, Stanford, CA, USA

ANDREW REYNOLDS
The University of Iowa, Iowa City, USA

CESARE TINELLI
The University of Iowa, Iowa City, USA

YONI ZOHAR
Bar-Ilan University, Ramat Gan, Israel

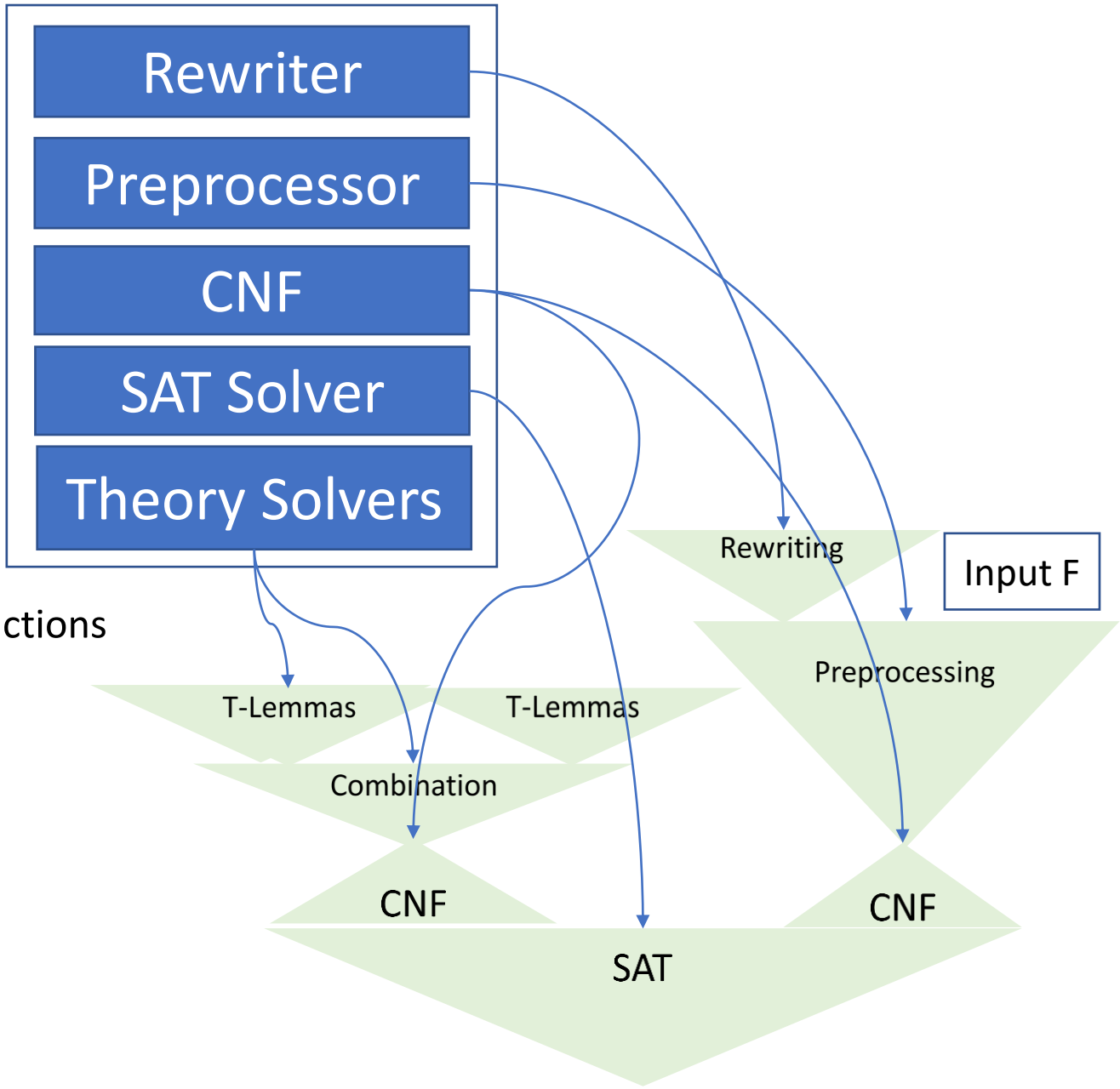
Moving toward a full suite of proof-producing automated reasoning tools with SMT solvers that can produce full, independently checkable proofs for real-world problems.

Generating and Exploiting Automated Reasoning Proof Certificates

- Internal proof calculus
  - Roughly 142 rules (132 core + 10 macro)
  - Native proof checker in cvc5's core
  - Original focus was on theory of strings
- Evaluated on many SMT-LIB theories **Barbosa et al IJCAR22**
- For more details, see **Barbosa et al CACM23**

cvc5 + Proofs

- Instrumented *many parts of system*:
  - Preprocessing
    - Boolean circuit propagation
  - Rewriting
  - SAT solver
  - CNF conversion
  - Quantifier instantiation
    - Used for reasoning about extended string functions
  - UF Theory Solver / congruence closure
  - Linear Arithmetic Solver
  - Theory Combination
  - Strings Theory Solver
    - Regular expression unfolding
    - Extended function reductions
    - Core calculus (Liang et al CAV 2014)



Rules (51) used in 4.5M Zelkova Problems

AND_ELIM	83866713	CONG	174513997	NOT_OR_ELIM	265133
AND_INTRO	32521977	CONTRA	198795	RE_INTER	62
ARITH_MULT_NEG	88	EQ_RESOLVE	50749239	RE_UNFOLD_POS	886262
ARITH_MULT_POS	88	EQUIV_ELIM1	1866156	REFL	116156677
ARITH_POLY_NORM	517	EQUIV_ELIM2	199786	REMOVE_TERM_FORMULA	44
ARITH_SUM_UB	88	EVALUATE	28201002	REORDERING	2811609
ASSUME	83889221	FACTORING	245432	RESOLUTION	2830376
BV_BITBLAST_STEP	660	FALSE_ELIM	1041313	SCOPE	3031886
CHAIN_RESOLUTION	3973499	FALSE_INTRO	208289	SKOLEM_INTRO	265
CNF_AND_NEG	113770	IMPLIES_ELIM	1052199	STRING_EAGER_REDUCTION	6
CNF_AND_POS	3105408	ITE_ELIM1	44	SYMM	35719138
CNF_EQUIV_POS1	45034	ITE_ELIM2	44	TRANS	71642174
CNF_EQUIV_POS2	32532	MODUS_PONENS	1041495	TRUE_ELIM	1312434
CNF_IMPLIES_NEG1	6316	NOT_AND	861574	TRUE_INTRO	37
CNF_IMPLIES_NEG2	6507	NOT_IMPLIES_ELIM1	47603	**TRUST_PREPROCESS**	545
CNF_OR_NEG	121563	NOT_IMPLIES_ELIM2	55869	**TRUST_THEORY_INFERENCE**	655
CNF_OR_POS	135584	NOT_NOT_ELIM	107846	**TRUST_THEORY_REWRITE**	86215359

Rules (51) used in 4.5M Zelkova Problems

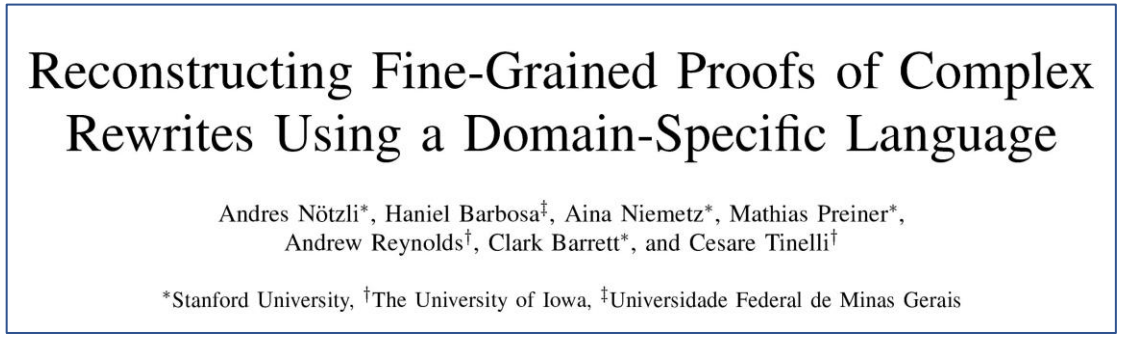
AND_ELIM	83866713	CONG	174513997	NOT_OR_ELIM	265133
AND_INTRO	32521977	CONTRA	198795	RE_INTER	62
ARITH_MULT_NEG	88	EQ_RESOLVE	50749239	RE_UNFOLD_POS	886262
ARITH_MULT_POS	88	EQUIV_ELIM1	1866156	REFL	116156677
ARITH_POLY_NORM	517	EQUIV_ELIM2	199786	REMOVE_TERM_FORMULA	44
ARITH_SUM_UB	88	EVALUATE	28201002	REORDERING	2811609
ASSUME	83889221	FACTORING	245432	RESOLUTION	2830376
BV_BITBLAST_STEP	660	FALSE_ELIM	1041313	SCOPE	3031886
CHAIN_RESOLUTION	3973499	FALSE_INTRO	208289	SKOLEM_INTRO	265
CNF_AND_NEG	113770	IMPLIES_ELIM	1052199	STRING_EAGER_REDUCTION	6
CNF_AND_POS	3105408	ITE_ELIM1	44	SYMM	35719138
CNF_EQUIV_POS1	45034	ITE_ELIM2	44	TRANS	71642174
CNF_EQUIV_POS2	32532	MODUS_PONENS	1041495	TRUE_ELIM	1312434
CNF_IMPLIES_NEG1	6316	NOT_AND	861574	TRUE_INTRO	37
CNF_IMPLIES_NEG2	6507	NOT_IMPLIES_ELIM1	47603	**TRUST_PREPROCESS**	545
CNF_OR_NEG	121563	NOT_IMPLIES_ELIM2	55869	**TRUST_THEORY_INFERENCE**	655
CNF_OR_POS	135584	NOT_NOT_ELIM	107846	**TRUST_THEORY_REWRITE**	86215359

…can elaborate via rewrite DSL

Proofs for Rewrites in cvc5

RARE: a DSL for Proofs of Rewrites

```
(define-rule* str-concat-unify
  ((s1 String) (s2 String) (s3 String :list) (t2 String) (t3 String :list))
  (= (str.++ s1 s2 s3) (str.++ s1 t2 t3))
  (= (str.++ s2 s3) (str.++ t2 t3)))
```



Reconstructing Fine-Grained Proofs of Complex Rewrites Using a Domain-Specific Language

Andres Nötzli*, Haniel Barbosa‡, Aina Niemetz*, Mathias Preiner*,
Andrew Reynolds†, Clark Barrett*, and Cesare Tinelli†

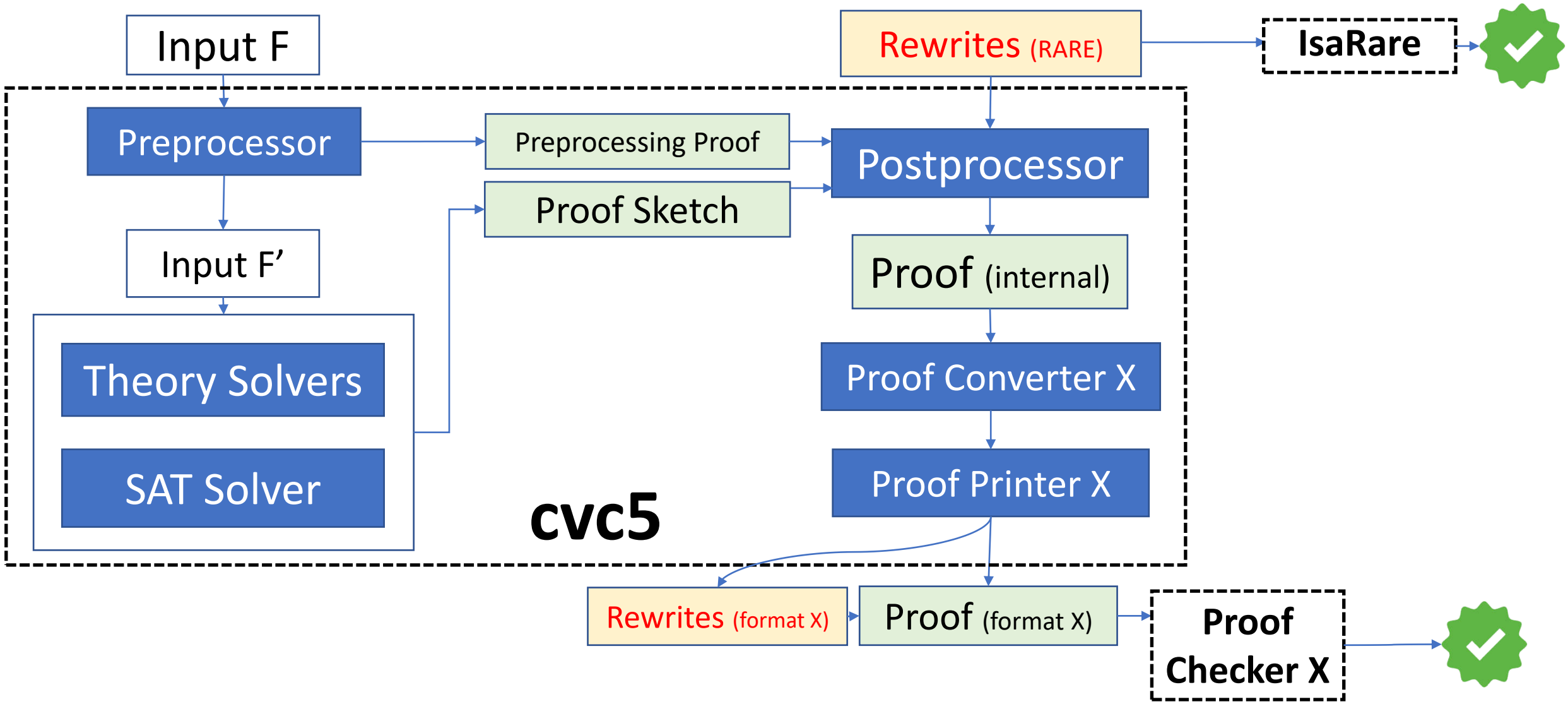
*Stanford University, †The University of Iowa, ‡Universidade Federal de Minas Gerais

- Introduced DSL (RARE) for smt2 rewrites
- Automatic proof elaboration from user specification
- Developed rewrites, focusing on strings

  Noetzli et al FMCAD22
- Under submission to TACAS 24:

  Lachnitt et al, “Automatic Verification of SMT Rewrites in Isabelle/HOL”

cvc5 + Proofs *+ RARE*



Rules (51) used in 4.5M Zelkova Problems

AND_ELIM	83866713	CONG	174513997	NOT_OR_ELIM	265133
AND_INTRO	32521977	CONTRA	198795	RE_INTER	62
ARITH_MULT_NEG	88	EQ_RESOLVE	50749239	RE_UNFOLD_POS	886262
ARITH_MULT_POS	88	EQUIV_ELIM1	1866156	REFL	116156677
ARITH_POLY_NORM	517	EQUIV_ELIM2	199786	REMOVE_TERM_FORMULA	44
ARITH_SUM_UB	88	EVALUATE	28201002	REORDERING	2811609
ASSUME	83889221	FACTORING	245432	RESOLUTION	2830376
BV_BITBLAST_STEP	660	FALSE_ELIM	1041313	SCOPE	3031886
CHAIN_RESOLUTION	3973499	FALSE_INTRO	208289	SKOLEM_INTRO	265
CNF_AND_NEG	113770	IMPLIES_ELIM	1052199	STRING_EAGER_REDUCTION	6
CNF_AND_POS	3105408	ITE_ELIM1	44	SYMM	35719138
CNF_EQUIV_POS1	45034	ITE_ELIM2	44	TRANS	71642174
CNF_EQUIV_POS2	32532	MODUS_PONENS	1041495	TRUE_ELIM	1312434
CNF_IMPLIES_NEG1	6316	NOT_AND	861574	TRUE_INTRO	37
CNF_IMPLIES_NEG2	6507	NOT_IMPLIES_ELIM1	47603	**TRUST_PREPROCESS**	545
CNF_OR_NEG	121563	NOT_IMPLIES_ELIM2	55869	**TRUST_THEORY_INFERENCE**	655
CNF_OR_POS	135584	NOT_NOT_ELIM	107846	**TRUST_THEORY_REWRITE**	5,085,132

~~86,215,359~~

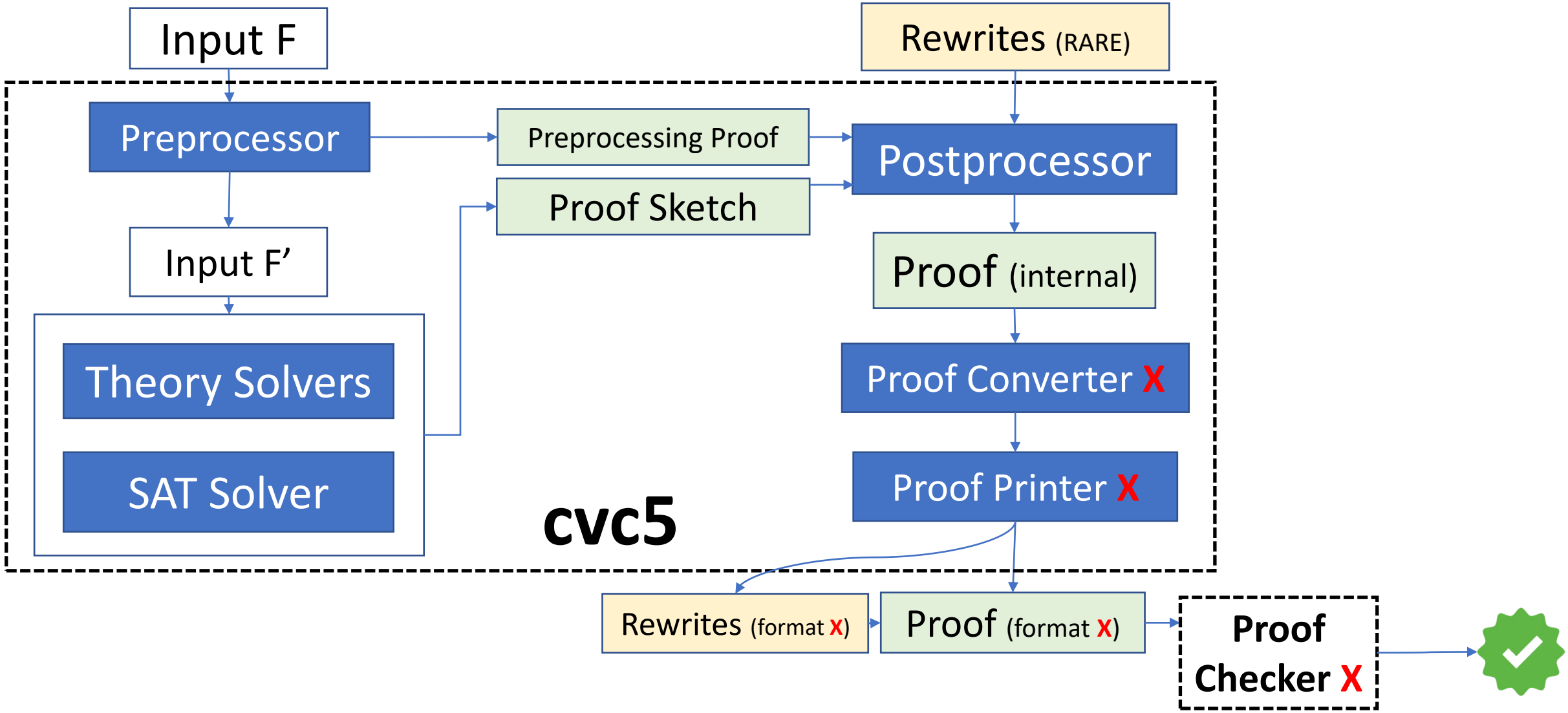
94.1% rewrites elaborated

arith-elim-leq	201	bv-and-one	120485
arith-elim-lt	646	bv-and-zero	103020
arith-geq-norm	3857	bv-commutative-and	205096
arith-geq-tighten	297	eq-refl	14406
bool-and-dup	2170	eq-symm	3567666
bool-and-false	6582139	re-concat-flatten	252
bool-and-flatten	19165966	re-concat-star-swap	16
bool-and-true	10246072	re-in-comp	48
bool-double-not-elim	1698700	re-in-cstring	1803006
bool-eq-false	6462474	re-in-empty	48
bool-eq-true	326548	re-in-sigma-star	4
bool-impl-false1	209064	str-concat-clash	3617
bool-impl-true2	279683	str-concat-flatten	854
bool-or-dup	36435	str-concat-flatten-eq	4727
bool-or-false	1562820	str-concat-unify	54
bool-or-flatten	463099	str-concat-unify-rev	7420
bool-or-true	57124	str-eq-ctn-false	34

…expands to 34 DSL rewrite rules
(as of March 2023)

Next Step: Exporting cvc5 Proofs

Key question: …which proof format X to Export?

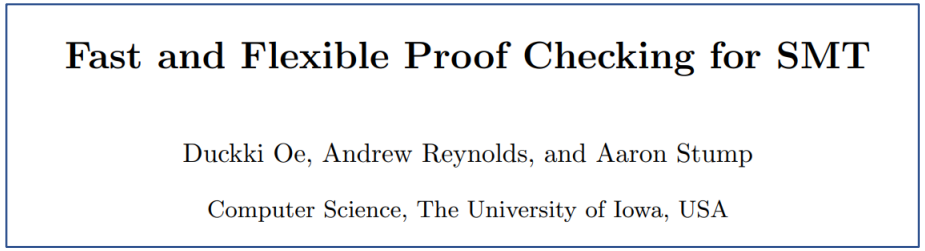


Previous Work: *LFSC* for SMT proofs

- Based on Edinburgh Logical Framework (LF)
  - Extended with side conditions
- User defined proof systems

Oe et al SMT09, Stump et al FMSD13, Hadarean et al LPAR15, Katz et al FMCAD16

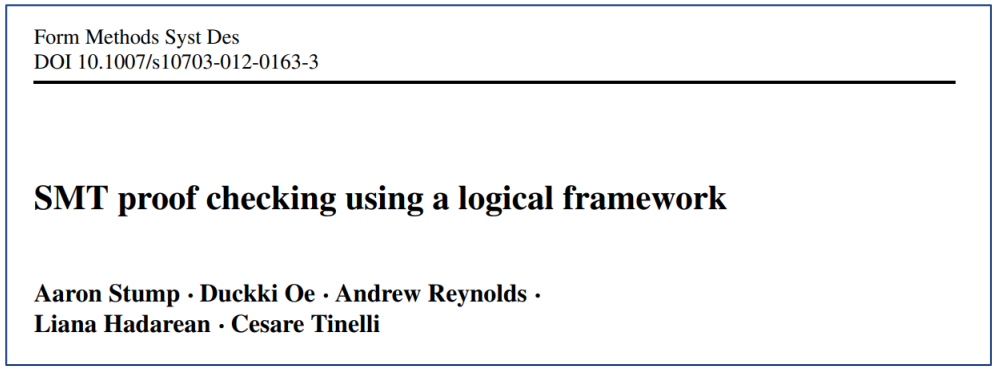
- Why this was a suboptimal solution:
  - ⛔ Performance
  - ⛔ Proof rules must encoded at a low level
  - ⛔ Syntax for terms does not match SMTLIB
    - No native for SMTLIB values
  - ⛔ Limited tooling support



Fast and Flexible Proof Checking for SMT

Duckki Oe, Andrew Reynolds, and Aaron Stump

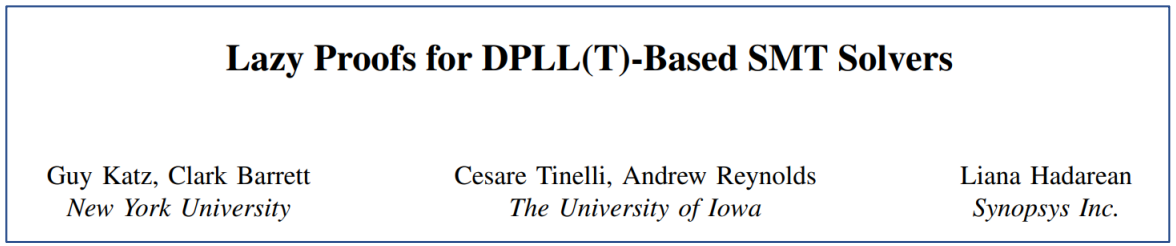
Computer Science, The University of Iowa, USA



Form Methods Syst Des
DOI 10.1007/s10703-012-0163-3

SMT proof checking using a logical framework

Aaron Stump · Duckki Oe · Andrew Reynolds · Liana Hadarean · Cesare Tinelli



Lazy Proofs for DPLL(T)-Based SMT Solvers

Guy Katz, Clark Barrett
New York University

Cesare Tinelli, Andrew Reynolds
The University of Iowa

Liana Hadarean
Synopsys Inc.

Previous Work : ***Alethe*** for SMT proofs

- Proof format based on SMTLIB syntax
  - Proofs are lists of extended smt2 commands
- Also produced by SMT solver veriT
- Efficient checker (Carcara)

Schurr et al PxTP 2021, Andreotti et al TACAS 2023

Alethe: Towards a Generic SMT Proof Format (extended abstract)

Hans-Jörg Schurr
University of Lorraine, CNRS, Inria, and LORIA, Nancy, France
hans-jorg.schurr@inria.fr

Mathias Fleury
Johannes Kepler University Linz, Austria
mathias.fleury@jku.at

Haniel Barbosa
Universidade Federal de Minas Gerais, Belo Horizonte, Brazil
hbarbosa@dcc.ufmg.br

Pascal Fontaine
University of Liège, Belgium
pascal.fontaine@uliege.be

- Why this was a suboptimal solution:
  - 🚫 Proof rules are hardcoded in pen-and-paper standard
    - Available only for limited theories
  - 🚫 Cannot capture cvc5's type system, proof calculus
  - 🚫 Time consuming and error prone to extend to new rules

Carcara: An efficient proof checker and elaborator for SMT proofs in the Alethe format*

Bruno Andreotti[1], Hanna Lachnitt[2], Haniel Barbosa[1]

[1] Universidade Federal de Minas Gerais, Belo Horizonte, Brazil
[2] Stanford University, Stanford, USA

Other Related Proof Efforts

- DRAT/LRAT formats used in SAT community
  - ⛔ Handles only the SAT portion of the proof
- Extensions to SMT theories, e.g. eDRAT
  - ⛔ Does not handle proofs of preprocessing or rewriting
- Verified proof checking in Lean
  - ⛔ Requires substantially more effort, expected to be slower

Introducing:
The AletheLF Proof Format

Introducing: *AletheLF*

- **Flexibility** of LFSC
- **Syntax** of Alethe and SMTLIB v3.0
- **Granularity** of RARE for rewrite rules
- **Speed** of checkers for other formats e.g. DRAT via *oracles*

Goals:

- Clear definition of cvc5's proof calculus
- Baseline proof checker (alfc)
- Intermediate step towards an efficient fully **verified** checking for SMT

What is AletheLF (ALF)?

- Proof format based on **SMTLIB version 3.0**
  - Commands for defining theory signatures
  - Commands for defining proof rules
  - Commands for writing proofs

```
(declare-const and (-> Bool Bool Bool)
   :right-assoc-nil true)
```

```
(declare-rule and_elim ((Fs Bool) (i Int))
    :premises (Fs)
    :args (i)
    :conclusion (alf.extract and Fs i))
```

```
(assume p0 (and A B C))
(step p1 B :rule  and_elim :premises (p0) :args (1))
```

- Features: side conditions, builtin evaluation operators, oracles

DEMO…

How have we used ALF so far?

- Translation of cvc5's internal calculus to ALF format:
  - ~2400 LOC (*.smt3)
  - ~90 proof rules
    - Most common rules of cvc5's internal calculus have been translated

- Includes:
  - Complete definition of cvc5's theory symbols and its type system
  - Formalization of most internal symbols (skolems) introduced by cvc5
  - Proof rules covering:
    - CNF, resolution, equality, strings, arithmetic, arrays, quantifiers
    - Alternative: DRAT for SAT proofs

alfc: A Proof Checker for AletheLF

DEMO…

alfc: A Proof Checker for AletheLF

- Efficient proof checker for ALF
  - ~9300 LOC (C++)
- Incorporates:
  - A parser and lexer for smt3
  - Utilities for evaluation over SMTLIB values (integers, rationals, bitvectors, strings)
  - Implementation of core logic for type checking and evaluation
- Support for several key features:
  - Side conditions
  - Reference validation `(reference "original.smt2")`
  - Oracle functions `(declare-oracle-fun f () Int ./binary)`
  - Signature compilation to C++
- Available at https://github.com/cvc5/alfc
  - User manual available

$\Rightarrow$ Also serves as a reference parser for SMTLIB version 3

Key Features of alfc

DEMO…

How do I know an ALF proof is for the right input?

- alfc accepts a `reference` command
  - Checks that `assume` match `assert` in reference file

simple-lia.smt2 :

```
(set-logic QF_LIA)
(declare-fun x () Int)
(declare-fun y () Int)
(assert (= x y))
(assert (not (= x y)))
(check-sat)
```

```
(declare-fun x () Int)
(declare-fun y () Int)
(define t1 () (= x y))
(assume p1 t1)
(assume p2 (not t1))
(step p3 false :rule contra :premises (p1 p2))
```

```
(reference "simple-lia.smt2")
(define t1 () (= x y))
(assume p1 t1)
(assume p2 (not t1))
(step p3 false :rule contra :premises (p1 p2))
```

* cvc5 prints reference command if option `--alf-print-reference` is enabled

How can we leverage DRAT?

IPASIR-UP: User Propagators for CDCL

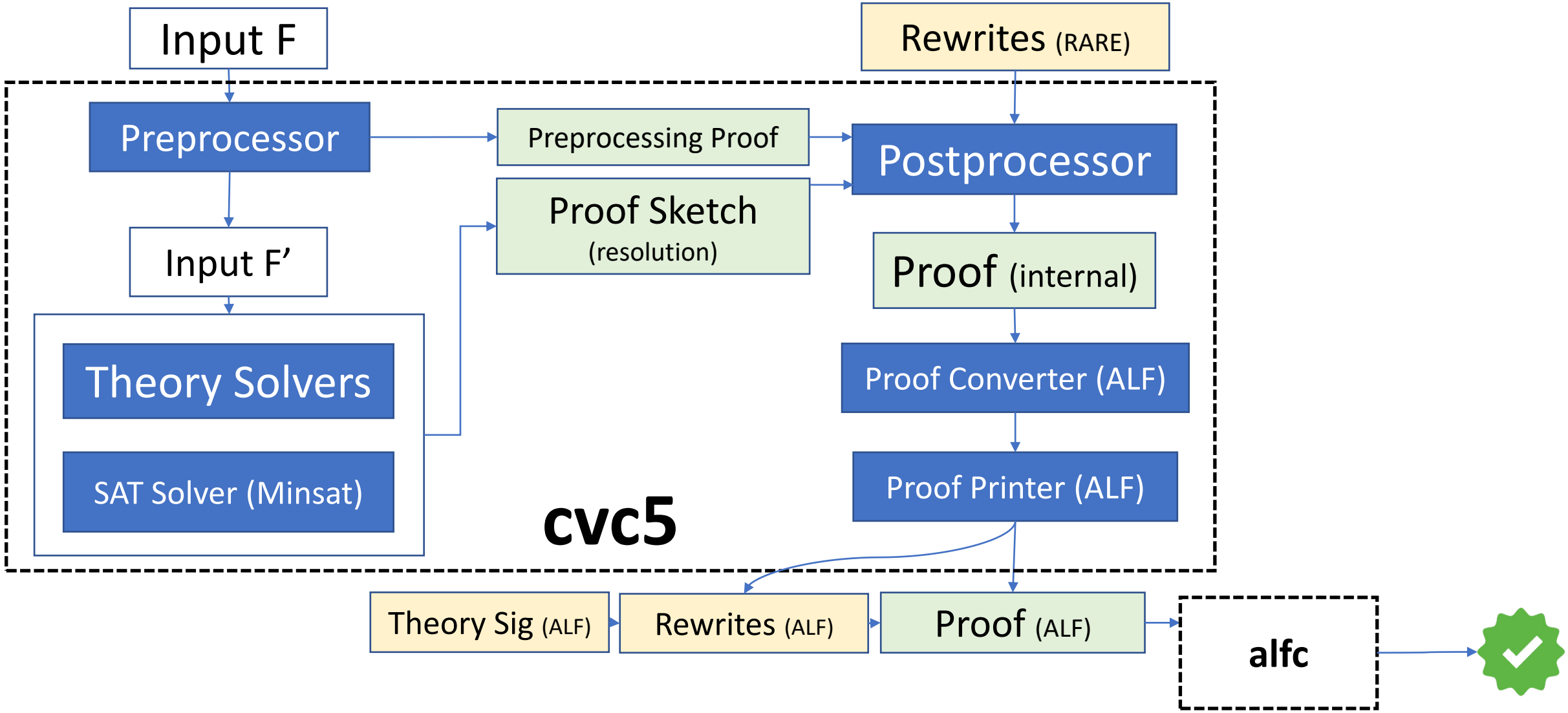
Katalin Fazekas
TU Wien, Vienna, Austria
Aina Niemetz
Stanford University, Stanford, USA
Mathias Preiner
Stanford University, Stanford, USA
Markus Kirchweger
TU Wien, Vienna, Austria
Stefan Szeider
TU Wien, Vienna, Austria
Armin Biere
University of Freiburg, Freiburg, Germany

- cvc5 now uses configurable SAT solver
  - Must conform to standard interface (IPASIR-UP)

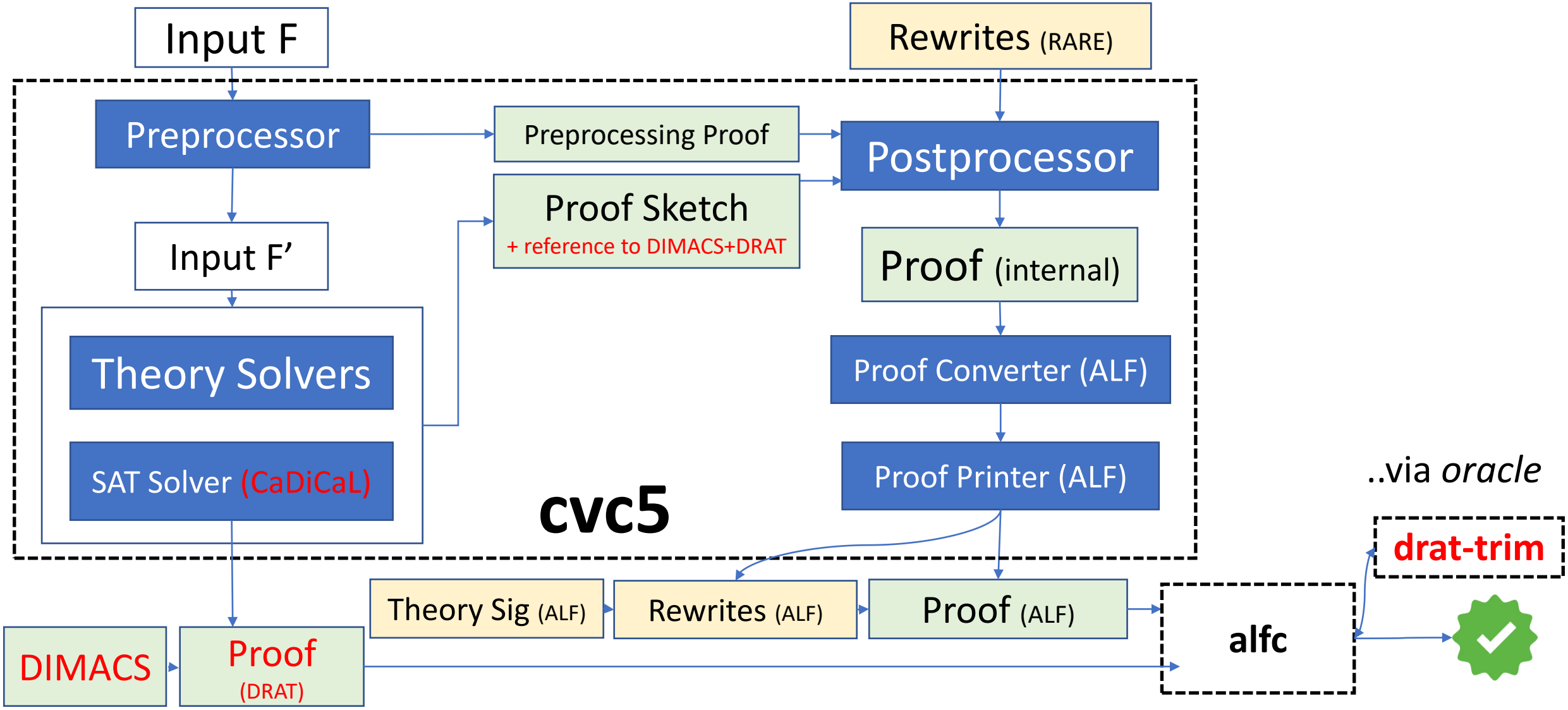
    Fazekas et al SAT 2023
- Notably, the main branch of cvc5 supports CaDiCaL
  - Via option `--sat-solver=cadical`
- Choice of SAT solver can dramatically improve solving times

$\Rightarrow$ *ALF proofs can incorporate black-box (DRAT) proof output from a SAT solver*

Incorporating ALF+DRAT



Incorporating ALF+DRAT (option 1)



alfc + Oracles

- alfc supports *oracle functions*
  - Syntax and semantics introduced in:

    Polgreen et al VMCAI 2022
- Command `declare-oracle-fun`

Satisfiability and Synthesis Modulo Oracles

Elizabeth Polgreen[1,2] and Andrew Reynolds[3] and Sanjit A. Seshia[1]

[1] University of California, Berkeley
[2] University of Edinburgh
[3] University of Iowa

- Oracle funs are like declare-fun but have semantics implemented by external binary
  - Communication via smt2 (or smt3) text

DEMO…

Oracles for DRAT

```
; ./drat-check.sh
; - A DIMCAS input file, whose file name is given as a string
; - A DRAT proof file, whose file name is given as a String.
; It returns "true" if the DRAT proof file is a valid refutation proof.

(declare-oracle-fun run_drat_check (String String) Bool ./drat-check.sh)
```

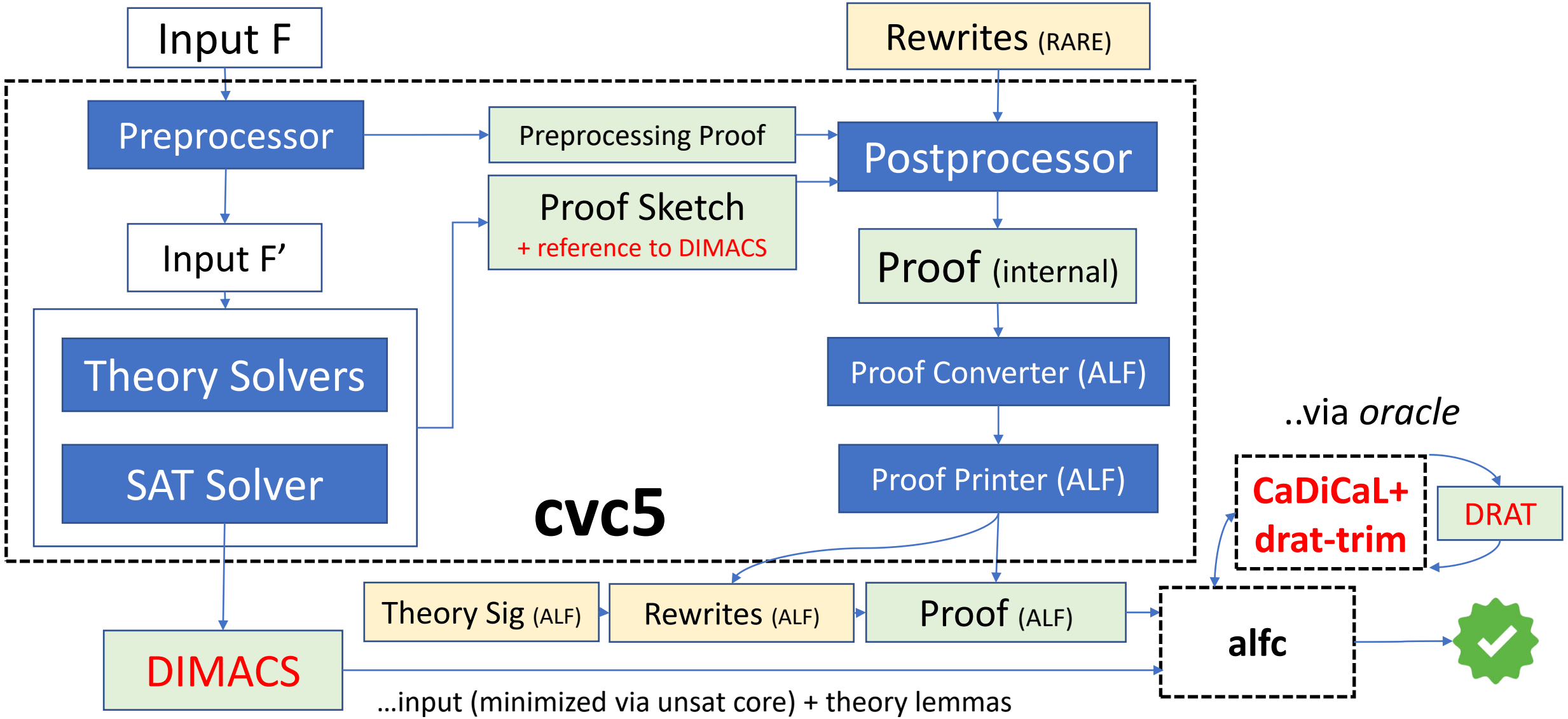
```
; ./dratt-verify.sh takes:
; - A string corresponding to a DIMACS declaration of the input as computed by the ALF checker
; - A DIMACS input file, whose file name is given as a String.
; It returns "true" if the preamble of the DRAT proof file matches (modulo renaming identifiers).

(declare-oracle-fun run_drat_verify (String String) Bool ./drat-verify)
```

⇒ cvc5's ALF signature for DRAT calls two oracles:

1. A proof checker for DRAT (drat-trim)

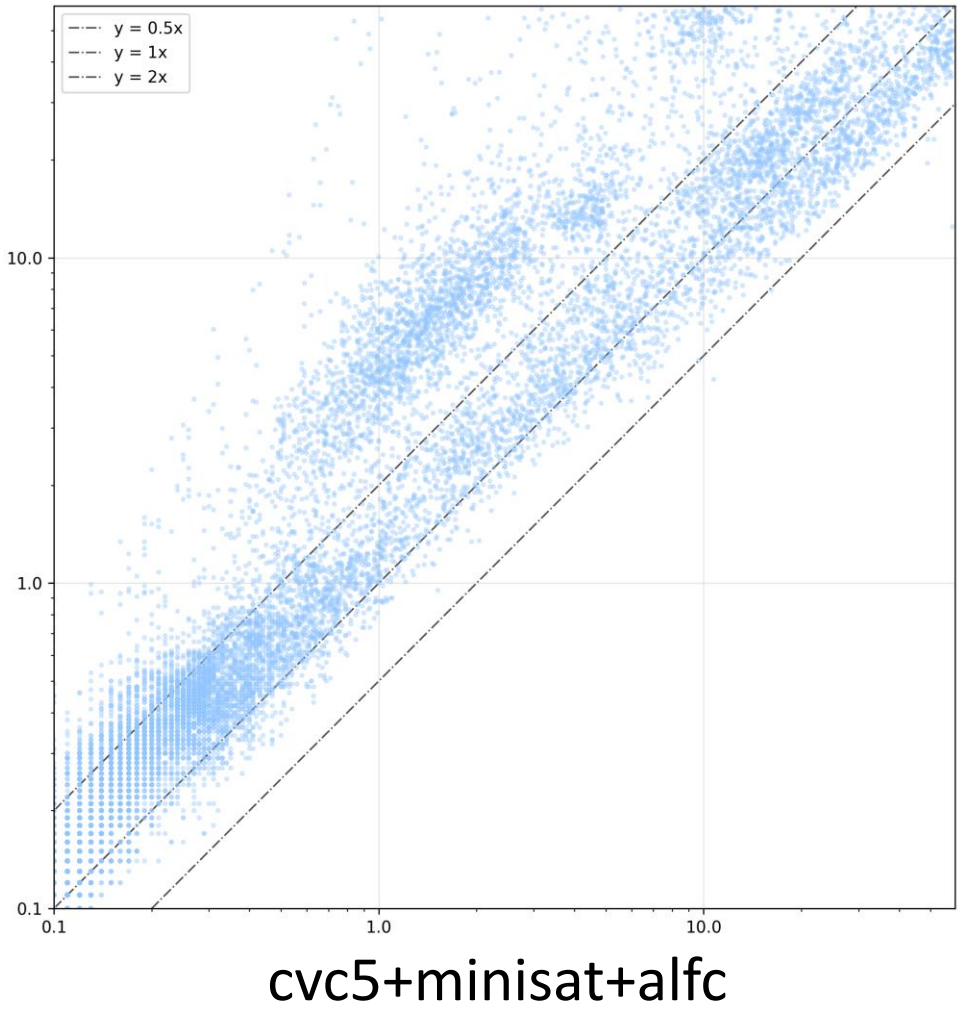
2. A input verifier DIMACS⇔smt2 (drat-verify, ~150 LOC C++)

Incorporating ALF+DRAT (option 2)



Preliminary Evaluation

Results: alfc vs lfsc (on resolution proofs)

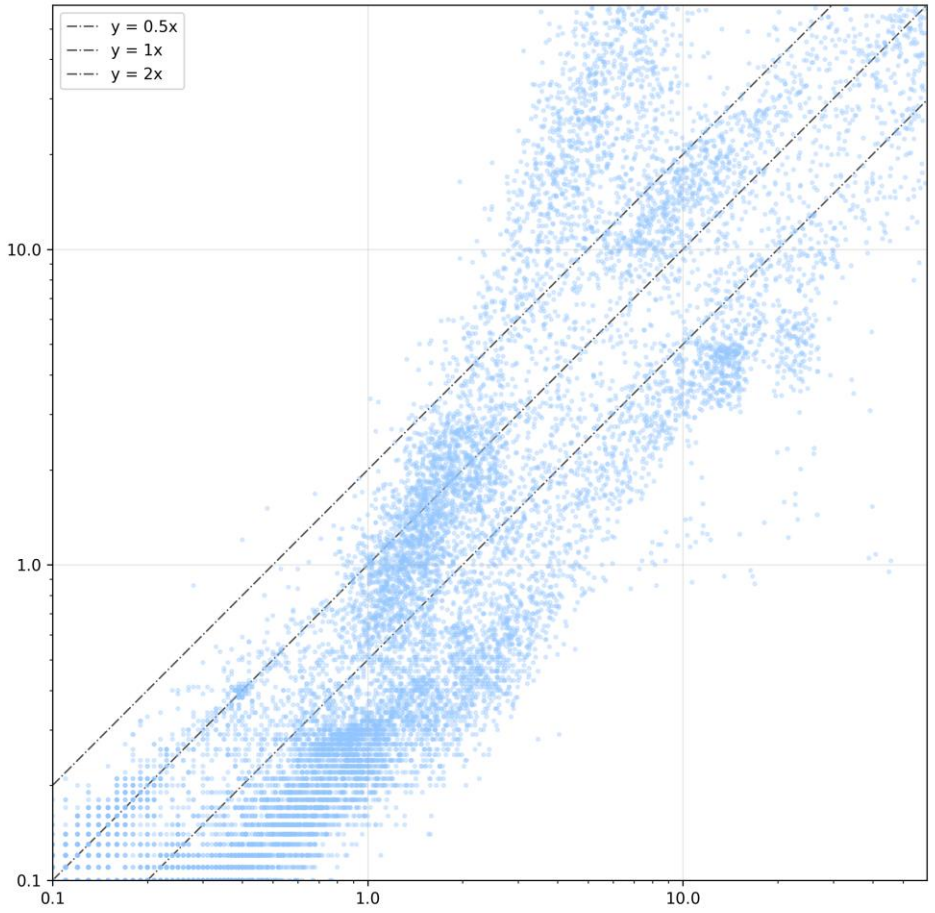


97348 benchmarks
60 second timeout
All quantifier-free SMTLIB logics with
- strings (S)
- linear arithmetic (LIRA)
- uninterpreted functions (UF)

⇒alfc 1.56x faster proof checking than lfsc

- Due to flexibility in alfc (e.g. uses chain resolution)

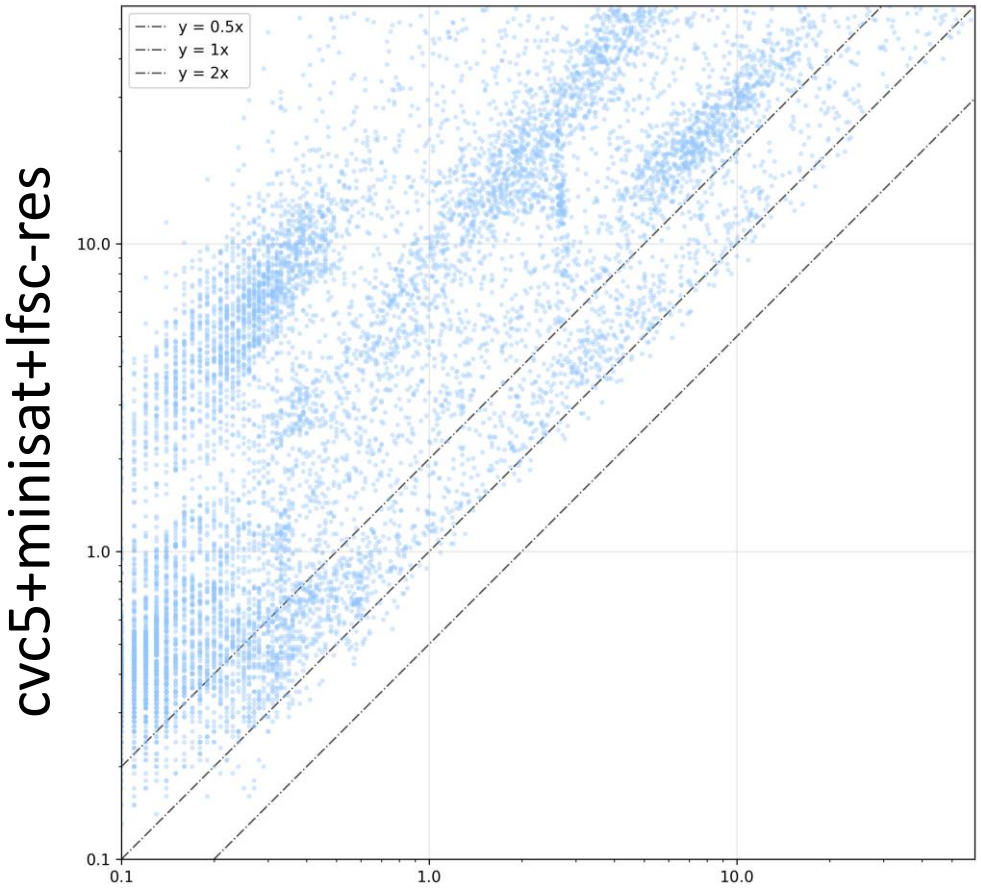
Results: alfc+DRAT vs alfc+resolution



⇒DRAT scales better than res on harder examples

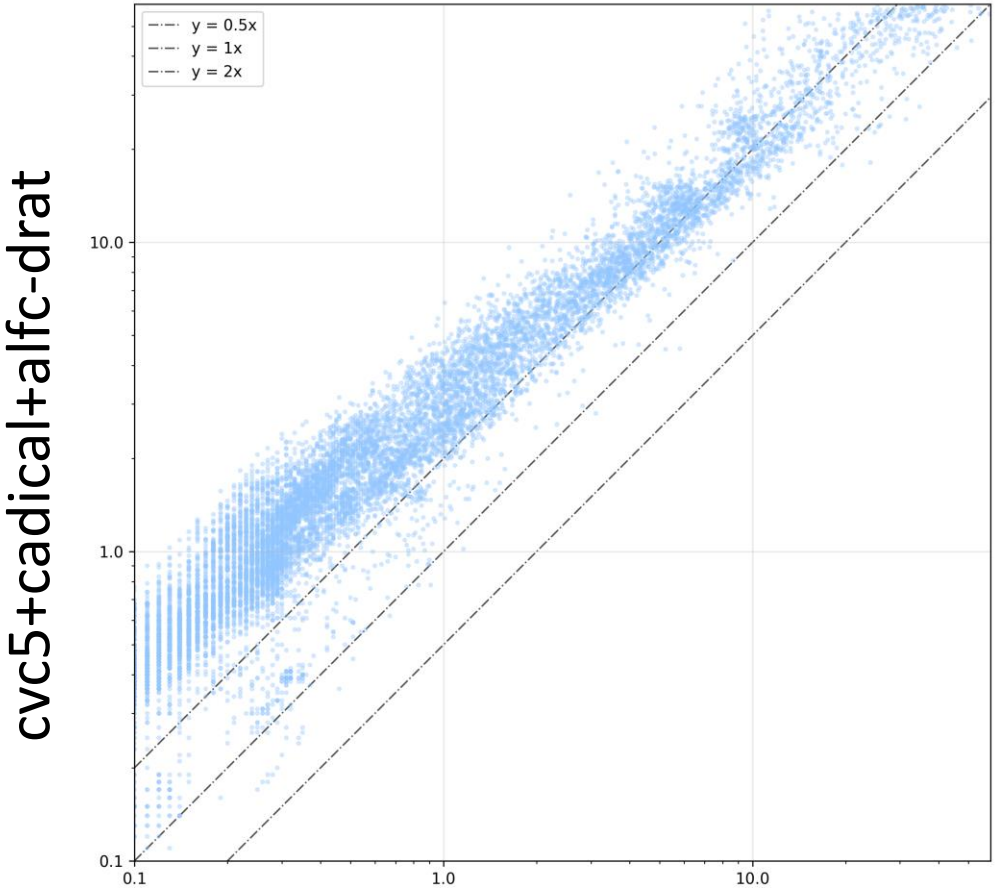
- 1.34x faster for benchmarks >5 seconds

Results: LFSC vs alfc+DRAT, checking overhead



cvc5+minisat+proofs

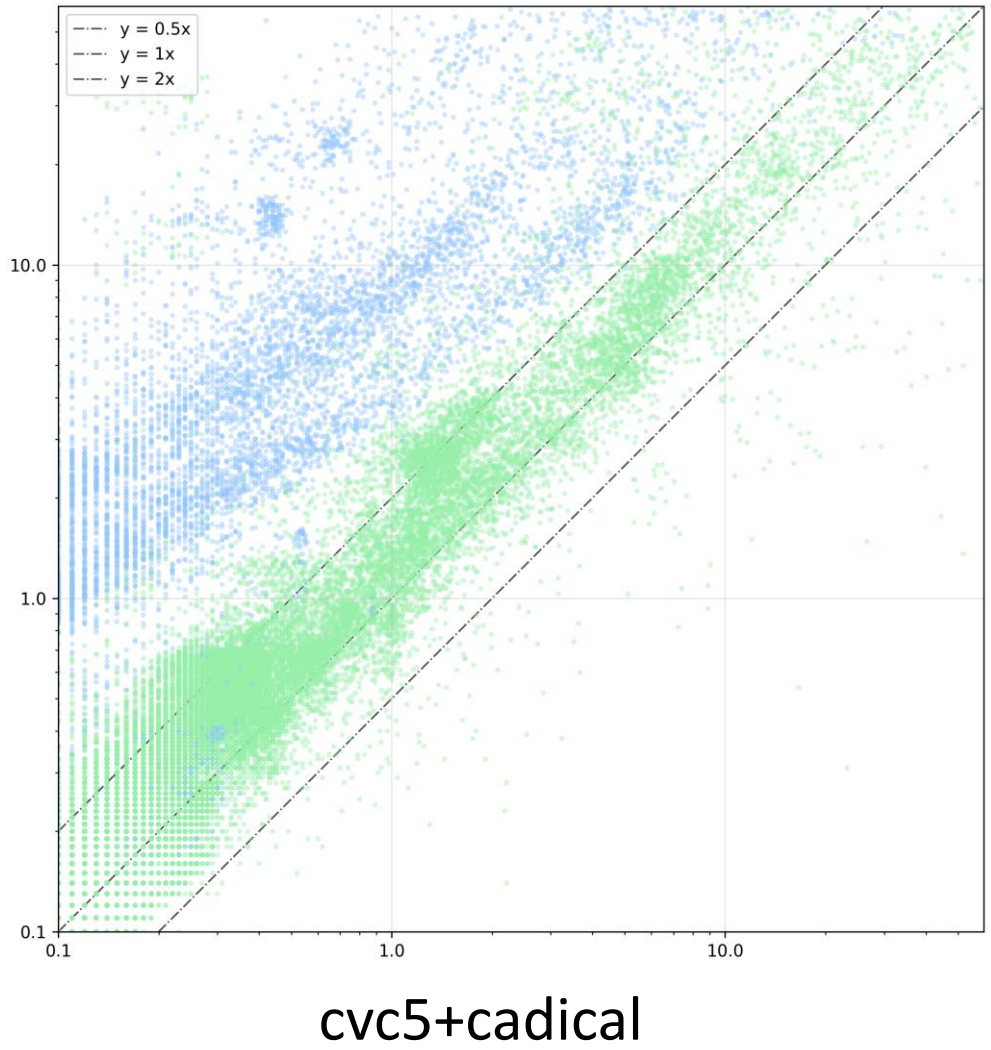
(6.0x overhead)



cvc5+cadical+proofs

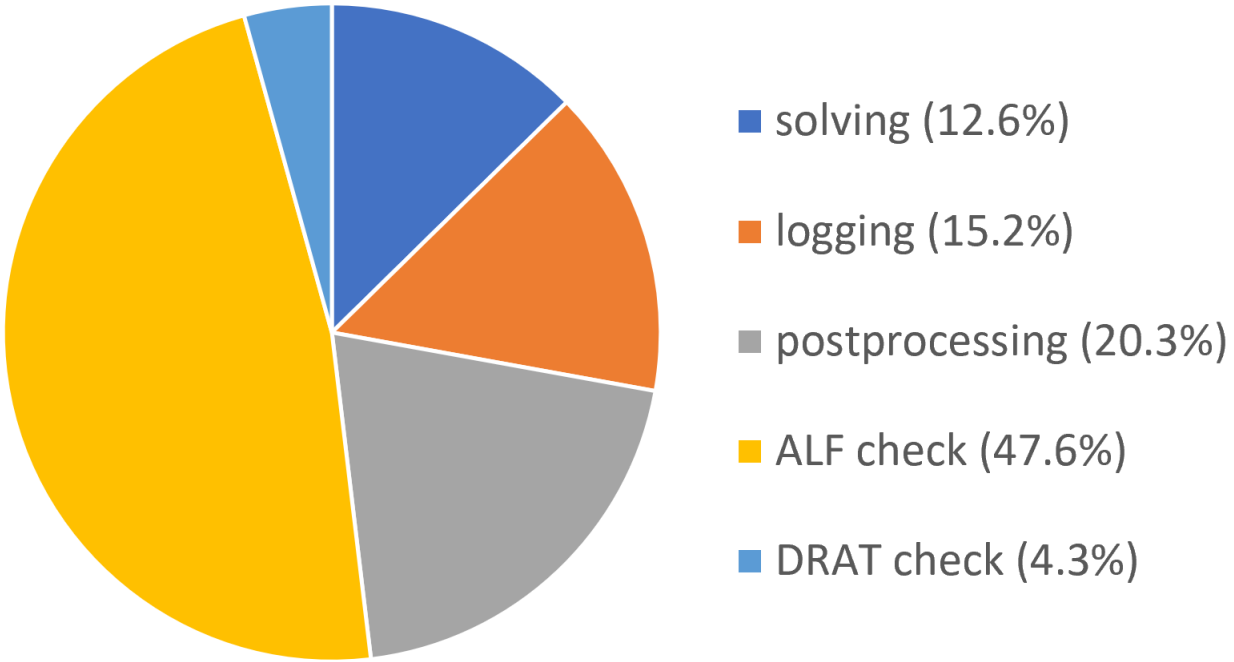
(2.3x overhead)

Results: alfc+DRAT vs Solving



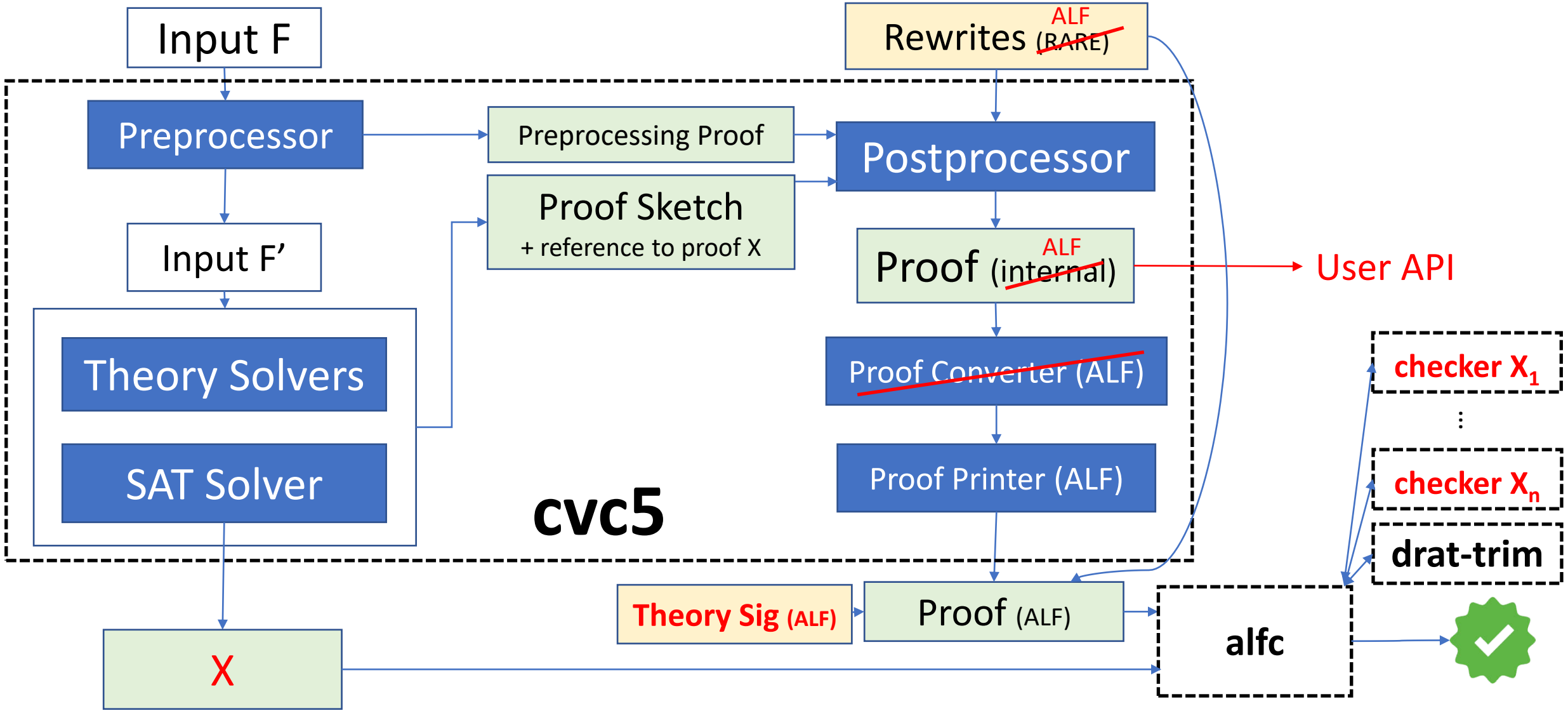
⇒ 7.9x overhead solving+proof checking (UNSAT)

- 1.4x for SAT problems



Future Work

Future Work


Input F
Preprocessor
Input F'
Theory Solvers
SAT Solver
X
Preprocessing Proof
Proof Sketch
+ reference to proof X
cvc5
Rewrites ALF (BARE)
Postprocessor
Proof (internal) ALF
Proof Converter (ALF)
Proof Printer (ALF)
User API
Theory Sig (ALF)
Proof (ALF)
alfc
checker X1
checker Xn
drat-trim

Future Work

- Support current uses of Alethe in SMTCoq, Isabelle
- Formalize core logic of ALF checker in Agda
- Improvements to cvc5 performance
  - In particular, preprocessing and minimizing theory lemmas in DRAT
- Support further integrations with formats (LRAT), other checkers

Conclusion

- `cvc5` at https://cvc5.github.io/
  - Proof output now available on main via `--dump-proofs --proof-format=alf`
  - DRAT and RARE integrations available on dev branch, will be in v1.1

- ALF checker `alfc` available at https://github.com/cvc5/alfc
  - Available via cvc5 repo `./contrib/get-alf-checker`