Scalable fine-grained proofs for formula processing

Haniel Barbosa[1]
Jasmin Blanchette[1,2]
Pascal Fontaine[1]

[1]University of Lorraine, CNRS, Inria, LORIA, Nancy, France
[2]Vrije Universiteit Amsterdam, The Netherlands

matryoshka

CADE-26

2017–08–09, Gothenburg, Sweden

Why proofs?

- ▷ to check the result for unsatisfiable/valid formulas
- ▷ for solver/prover cooperation

- ▷ as a debugging facility
- ▷ for evaluation purposes (how good is the algorithm?)

- ▷ as a part of the reasoning framework (e.g. conflict clauses)
- ▷ to extract cores
- ▷ to compute interpolants

Challenges for proofs in FOL

- Collecting and storing proof information efficiently
- Producing proofs for sophisticated processing techniques
- Producing proofs for modules that use external tools
- Standardizing a proof format

Challenges for proofs in FOL

- ▷ Collecting and storing proof information efficiently
  no convergence, but quite active
  [KBT+16; HBR+15; KV13; Sch13; BODF09; WDF+09; Mos08; MB08; SZS04]

- ▷ Producing proofs for sophisticated processing techniques
  proofs with holes or too coarse

- ▷ Producing proofs for modules that use external tools
  depends on tool
  arbitrarily complex to reconstruct information

- ▷ Standardizing a proof format
  open

Challenges for proofs in FOL

- ▷ Collecting and storing proof information efficiently
  no convergence, but quite active
  [KBT+16; HBR+15; KV13; Sch13; BODF09; WDF+09; Mos08; MB08; SZS04]

- ▷ Producing proofs for sophisticated processing techniques
  ~~proofs with holes or too coarse~~ **scalable fine-grained proofs**

- ▷ Producing proofs for modules that use external tools
  depends on tool
  arbitrarily complex to reconstruct information

- ▷ Standardizing a proof format
  open

Proofs in veriT

Resolution chains, input formulas, tautologies for theory and quantifier reasoning

Proofs in veriT

Resolution chains, input formulas, tautologies for theory and quantifier reasoning

- ▷ SAT solver: resolution

$$\frac{A \vee \ell \qquad B \vee \overline{\ell}}{A \vee B}$$

Antecedents: $A \vee \ell$, $B \vee \overline{\ell}$
Pivot: ℓ or $\overline{\ell}$
Resolvent: $A \vee B = (A \vee \ell) \diamond (B \vee \overline{\ell})$

Proofs in veriT

Resolution chains, input formulas, tautologies for theory and quantifier reasoning

- SAT solver: resolution

$$\frac{A \vee \ell \qquad B \vee \overline{\ell}}{A \vee B}$$

Antecedents: $A \vee \ell$, $B \vee \overline{\ell}$
Pivot: ℓ or $\overline{\ell}$
Resolvent: $A \vee B = (A \vee \ell) \diamond (B \vee \overline{\ell})$

- theory solvers: theory lemmas

$$\neg(a \simeq c) \vee \neg(c \simeq b) \vee a \simeq b \qquad \neg(a \simeq b) \vee f(a) \simeq f(b)$$

$$\neg(y > 1) \vee \neg(x < 1) \vee y > x$$

Proofs in veriT

Resolution chains, input formulas, tautologies for theory and quantifier reasoning

- SAT solver: resolution

$$\frac{A \vee \ell \qquad B \vee \overline{\ell}}{A \vee B}$$

Antecedents: $A \vee \ell$, $B \vee \overline{\ell}$
Pivot: ℓ or $\overline{\ell}$
Resolvent: $A \vee B = (A \vee \ell) \diamond (B \vee \overline{\ell})$

- theory solvers: theory lemmas

$$\neg(a \simeq c) \vee \neg(c \simeq b) \vee a \simeq b \qquad \neg(a \simeq b) \vee f(a) \simeq f(b)$$

$$\neg(y > 1) \vee \neg(x < 1) \vee y > x$$

- instantiation module: instantiation lemmas

$$\neg(\forall x.\, \psi[x]) \vee \psi[t]$$

What is hard about proofs for formula processing?

Resolution does not capture all transformations

What is hard about proofs for formula processing?

Resolution does not capture all transformations

Some transformations do not preserve logical equivalence

What is hard about proofs for formula processing?

Resolution does not capture all transformations

Some transformations do not preserve logical equivalence

Code is lengthy and deals with many cases

Proving formula processing

Extensible framework to represent proofs for processing techniques involving *locally replacing equals by equals* in the presence of *binders*

Proving formula processing

Extensible framework to represent proofs for processing techniques involving *locally replacing equals by equals* in the presence of *binders*

Some instances:

Skolemization: $(\neg\, \forall x.\, \mathsf{p}(x)) \simeq \neg\, \mathsf{p}(\varepsilon x.\, \neg\, \mathsf{p}(x))$

let elimination: $(\mathrm{let}\ x \simeq \mathsf{a}\ \mathrm{in}\ \mathsf{p}(x,\, x)) \simeq \mathsf{p}(\mathsf{a},\, \mathsf{a})$

theory simplifications: $(\mathsf{k} + 1 \times 0 < \mathsf{k}) \simeq (\mathsf{k} < \mathsf{k})$

Proving formula processing

Extensible framework to represent proofs for processing techniques involving *locally replacing equals by equals* in the presence of *binders*

Some instances:

Skolemization: $(\neg \forall x.\, \mathsf{p}(x)) \simeq \neg\, \mathsf{p}(\varepsilon x.\, \neg\, \mathsf{p}(x))$

let elimination: $(\text{let } x \simeq \mathsf{a} \text{ in } \mathsf{p}(x,\, x)) \simeq \mathsf{p}(\mathsf{a},\, \mathsf{a})$

theory simplifications: $(\mathsf{k} + 1 \times 0 < \mathsf{k}) \simeq (\mathsf{k} < \mathsf{k})$

▷ Challenge is to manipulate bound variables and substitutions soundly and efficiently

Inference system

A context Γ fixes a set of variables and specifies a substitution

substitution

$$\Gamma ::= \varnothing \mid \Gamma,\, x \mid \Gamma,\, \bar{x}_n \mapsto \bar{s}_n$$

bound variable

Inference system

A context Γ fixes a set of variables and specifies a substitution

substitution

$$\Gamma ::= \varnothing \mid \Gamma, x \mid \Gamma, \bar{x}_n \mapsto \bar{s}_n$$

bound variable

Rules have the form

derivations of premises

$$\frac{\mathcal{D}_1 \qquad \cdots \qquad \mathcal{D}_n}{\Gamma \triangleright t \simeq u}\ \mathrm{R}$$

assumptions

transformation

- ▷ Semantically, the judgment expresses the equality of the terms $\Gamma(t)$ and u for all variables fixed by Γ

Example of ‘let’ expansion

$$\dfrac{\dfrac{}{\rhd\ \mathsf{a} \simeq \mathsf{a}}\ \textsc{Cong} \qquad \dfrac{\dfrac{}{x \mapsto \mathsf{a}\ \rhd\ x \simeq \mathsf{a}}\ \textsc{Refl} \qquad \dfrac{}{x \mapsto \mathsf{a}\ \rhd\ x \simeq \mathsf{a}}\ \textsc{Refl}}{x \mapsto \mathsf{a}\ \rhd\ \mathsf{p}(x, x) \simeq \mathsf{p}(\mathsf{a}, \mathsf{a})}\ \textsc{Cong}}{\rhd\ (\text{let } x \simeq \mathsf{a} \text{ in } \mathsf{p}(x, x)) \simeq \mathsf{p}(\mathsf{a}, \mathsf{a})}\ \textsc{Let}$$

Example of theory simplification

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\rhd\ \mathsf{k} \simeq \mathsf{k}}\,\textsc{Cong} \qquad \dfrac{}{\rhd\ 1{\times}0 \simeq 0}\,\textsc{Taut}_{\times}}{\rhd\ \mathsf{k} + 1{\times}0 \simeq \mathsf{k} + 0}\,\textsc{Cong} \qquad \dfrac{}{\rhd\ \mathsf{k} + 0 \simeq \mathsf{k}}\,\textsc{Taut}_{+}}{\rhd\ \mathsf{k} + 1{\times}0 \simeq \mathsf{k}}\,\textsc{Trans} \qquad \dfrac{}{\rhd\ \mathsf{k} \simeq \mathsf{k}}\,\textsc{Cong}}{\rhd\ (\mathsf{k} + 1{\times}0 < \mathsf{k}) \simeq (\mathsf{k} < \mathsf{k})}\,\textsc{Cong}$$

Output skolemization

The skolemization proof of the formula $\neg \forall x.\, \mathsf{p}(x)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x \mapsto \varepsilon x.\, \neg \mathsf{p}(x) \;\rhd\; x \simeq \varepsilon x.\, \neg \mathsf{p}(x)}\;\textsc{Refl}}{x \mapsto \varepsilon x.\, \neg \mathsf{p}(x) \;\rhd\; \mathsf{p}(x) \simeq \mathsf{p}(\varepsilon x.\, \neg \mathsf{p}(x))}\;\textsc{Cong}}{\rhd\; (\forall x.\, \mathsf{p}(x)) \simeq \mathsf{p}(\varepsilon x.\, \neg \mathsf{p}(x))}\;\textsc{Sko}_{\forall}}{\rhd\; (\neg \forall x.\, \mathsf{p}(x)) \simeq \neg \mathsf{p}(\varepsilon x.\, \neg \mathsf{p}(x))}\;\textsc{Cong}$$

Output skolemization

The skolemization proof of the formula $\neg\,\forall x.\ \mathsf{p}(x)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x \mapsto \varepsilon x.\,\neg\,\mathsf{p}(x)\ \rhd\ x \simeq \varepsilon x.\,\neg\,\mathsf{p}(x)}\ \textsc{Refl}}{x \mapsto \varepsilon x.\,\neg\,\mathsf{p}(x)\ \rhd\ \mathsf{p}(x) \simeq \mathsf{p}(\varepsilon x.\,\neg\,\mathsf{p}(x))}\ \textsc{Cong}}{\rhd\ (\forall x.\ \mathsf{p}(x)) \simeq \mathsf{p}(\varepsilon x.\,\neg\,\mathsf{p}(x))}\ \textsc{Sko}_{\forall}}{\rhd\ (\neg\,\forall x.\ \mathsf{p}(x)) \simeq \neg\,\mathsf{p}(\varepsilon x.\,\neg\,\mathsf{p}(x))}\ \textsc{Cong}$$

veriT syntax:

$$\begin{array}{l}
(.\mathrm{c0}\ (\mathsf{Sko\_All}\ \ \mathrm{:conclusion}\ ((\forall x.\ \mathsf{p}(x)) \simeq \mathsf{p}(\varepsilon x.\,\neg\,\mathsf{p}(x)))\\
\qquad\quad \mathrm{:args}\ (x \mapsto (\varepsilon x.\,\neg\,\mathsf{p}(x)))\\
\qquad\quad \mathrm{:subproof}\ ((.\mathrm{c1}\ (\mathsf{Refl}\ \ \mathrm{:conclusion}\ (x \simeq (\varepsilon x.\,\neg\,\mathsf{p}(x)))))\\
\qquad\qquad\qquad\quad\ (.\mathrm{c2}\ (\mathsf{Cong}\ \ \mathrm{:clauses}\ (.\mathrm{c1})\\
\qquad\qquad\qquad\qquad\qquad\quad \mathrm{:conclusion}\ (\mathsf{p}(x) \simeq \mathsf{p}(\varepsilon x.\,\neg\,\mathsf{p}(x))))))))\\
(.\mathrm{c3}\ (\mathsf{Cong}\ \ \mathrm{:clauses}\ (.\mathrm{c0})\ \ \mathrm{:conclusion}\ ((\neg\,\forall x.\ \mathsf{p}(x)) \simeq \neg\,\mathsf{p}(\varepsilon x.\,\neg\,\mathsf{p}(x)))))
\end{array}$$

Proof-producing contextual recursion

```
function $\mathit{process}(\Delta, t)$
  match $t$
    case $x$:
      return $\mathit{build\_var}(\Delta, x)$
    case $\mathsf{f}(\bar{t}_n)$:
      $\bar{\Delta}'_n \leftarrow (\mathit{ctx\_app}(\Delta, \mathsf{f}, \bar{t}_n, i))_{i=1}^{n}$
      return $\mathit{build\_app}\big(\Delta, \bar{\Delta}'_n, \mathsf{f}, \bar{t}_n, (\mathit{process}(\Delta'_i, t_i))_{i=1}^{n}\big)$
    case $Qx.\, \varphi$:
      $\Delta' \leftarrow \mathit{ctx\_quant}(\Delta, Q, x, \varphi)$
      return $\mathit{build\_quant}(\Delta, \Delta', Q, x, \varphi, \mathit{process}(\Delta', \varphi))$
    case $\mathrm{let}\ \bar{x}_n \simeq \bar{r}_n\ \mathrm{in}\ t'$:
      $\Delta' \leftarrow \mathit{ctx\_let}(\Delta, \bar{x}_n, \bar{r}_n, t')$
      return $\mathit{build\_let}(\Delta, \Delta', \bar{x}_n, \bar{r}_n, t', \mathit{process}(\Delta', t'))$
```

▷ Parameterized by a notion of context and plugin functions

Theoretical properties

Soundness of inference rules proven through an encoding into simply typed λ-calculus

Correctness of proof-producing contextual recursion algorithm

Cost of proof production is linear and of proof checking is (almost) linear*

* assuming suitable data structures

Implementation

Proof output for veriT

Framework implemented with a proof-producing contextual recursion algorithm

- ⊕ fine-grained proofs for most processing transformations
- ⊕ No negative impact on performance
- ⊕ More transformations in proof producing mode
- ⊕ Dramatic simplification of the code base

Implementation

Proof output for veriT

Framework implemented with a proof-producing contextual recursion algorithm

- ⊕ fine-grained proofs for most processing transformations
- ⊕ No negative impact on performance
- ⊕ More transformations in proof producing mode
- ⊕ Dramatic simplification of the code base

Prototype checker in Isabelle/HOL

Maps proofs into Isabelle theorems

- ⊕ Judgements encoded in λ-calculus

Conclusions

- Centralizes manipulation of bound variables and substitutions
- Accommodates many transformations (e.g. Skolemization)
- Proof checking is (almost) linear
- Implementation and integration within veriT

Conclusions

- ▷ Centralizes manipulation of bound variables and substitutions
- ▷ Accommodates many transformations (e.g. Skolemization)
- ▷ Proof checking is (almost) linear
- ▷ Implementation and integration within veriT

Future work

- ▷ Support global rewritings within the framework
- ▷ Support richer logics (e.g. HOL)
- ▷ Implement proof reconstruction in Isabelle/HOL

References

Guy Katz, Clark W. Barrett, Cesare Tinelli, Andrew Reynolds, and Liana Hadarean. "Lazy proofs for DPLL(T)-based SMT solvers". In: Formal Methods In Computer-Aided Design (FMCAD). Ed. by Ruzica Piskac and Muralidhar Talupur. IEEE, 2016, pp. 93–100.

Liana Hadarean, Clark W. Barrett, Andrew Reynolds, Cesare Tinelli, and Morgan Deters. "Fine Grained SMT Proofs for the Theory of Fixed-Width Bit-Vectors". In: Logic for Programming, Artificial Intelligence, and Reasoning (LPAR). Ed. by Martin Davis, Ansgar Fehnker, Annabelle McIver, and Andrei Voronkov. Vol. 9450. Lecture Notes in Computer Science. Springer, 2015, pp. 340–355.

Laura Kovács and Andrei Voronkov. "First-Order Theorem Proving and Vampire". English. In: Computer Aided Verification (CAV). Ed. by Natasha Sharygina and Helmut Veith. Vol. 8044. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 1–35.

Stephan Schulz. "System Description: E 1.8". English. In: Logic for Programming, Artificial Intelligence, and Reasoning (LPAR). Ed. by Ken McMillan, Aart Middeldorp, and Andrei Voronkov. Vol. 8312. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 735–743.

References

Thomas Bouton, Diego Caminha B. de Oliveira, David Déharbe, and Pascal Fontaine. "veriT: An Open, Trustable and Efficient SMT-Solver". In: Proc. Conference on Automated Deduction (CADE). Ed. by Renate A. Schmidt. Vol. 5663. Lecture Notes in Computer Science. Springer, 2009, pp. 151–156.

Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. "SPASS Version 3.5". English. In: Proc. Conference on Automated Deduction (CADE). Ed. by RenateA. Schmidt. Vol. 5663. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 140–145.

Michał Moskal. "Rocket-Fast Proof Checking for SMT Solvers". In: Tools and Algorithms for Construction and Analysis of Systems (TACAS). Ed. by C. R. Ramakrishnan and Jakob Rehof. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 486–500.

Leonardo Mendonça de Moura and Nikolaj Bjørner. "Proofs and Refutations, and Z3". In: Logic for Programming, Artificial Intelligence, and Reasoning (LPAR) Workshops. Ed. by Piotr Rudnicki, Geoff Sutcliffe, Boris Konev, Renate A. Schmidt, and Stephan Schulz. Vol. 418. CEUR Workshop Proceedings. CEUR-WS.org, 2008.

References

Geoff Sutcliffe, Jürgen Zimmer, and Stephan Schulz. "TSTP Data-Exchange Formats for Automated Theorem Proving Tools". In: Distributed Constraint Problem Solving and Reasoning in Multi-Agent Systems. Ed. by Weixiong Zhang and Volker Sorge. Vol. 112. Frontiers in Artificial Intelligence and Applications. IOS Press, 2004, pp. 201–215.