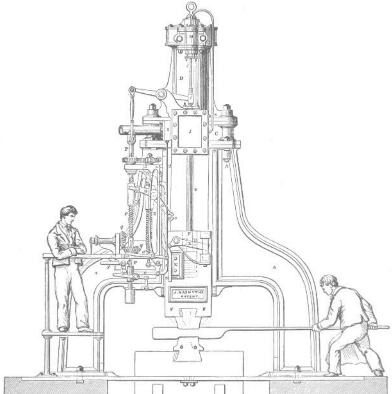
Stronger SMT Solvers for Proof Assistants

Proofs, Quantifier Simplification, Strategy Schedules

Hans-Jörg Schurr

PhD Defense

7 October 2022

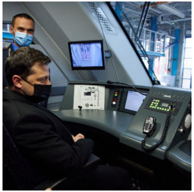


Starting Point

Many human pursuits demand precise and correct reasoning.

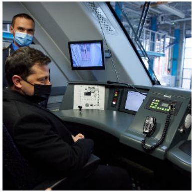
Starting Point

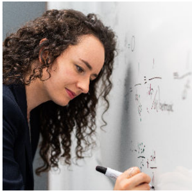
Many human pursuits demand precise and correct reasoning.



Starting Point

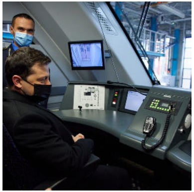
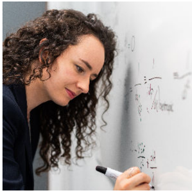
Many human pursuits demand precise and correct reasoning.





Starting Point

Many human pursuits demand precise and correct reasoning.

- Our tool: formal logic.
- It's unfeasible to write formal proofs by hand:

Reliability mistakes happen easily
Effort horribly time consuming

Software Supported Proof Construction

Proof Assistants

Reliability trusted kernel

Effort proof construction routines

Examples:

- **Isabelle/HOL**
- Coq
- Lean

Software Supported Proof Construction

Proof Assistants

Reliability trusted kernel

Effort proof construction routines

Examples:

- **Isabelle/HOL**
- Coq
- Lean

Automation

Must build uppon the kernel.

- Simplifier: replaces equal by equal.
- Integration of automated theorem provers.

Software Supported Proof Construction

Proof Assistants

Reliability trusted kernel

Effort proof construction routines

Examples:

- **Isabelle/HOL**
- Coq
- Lean

Automation

Must build uppon the kernel.

- Simplifier: replaces equal by equal.
- Integration of automated theorem provers.

Automated Theorem Provers

"Push Button"

Usually refute a problem and produce proofs.

Software Supported Proof Construction

Proof Assistants

Reliability trusted kernel

Effort proof construction routines

Examples:

- **Isabelle/HOL**
- Coq
- Lean

Automation

Must build uppon the kernel.

- Simplifier: replaces equal by equal.
- Integration of automated theorem provers.

Automated Theorem Provers

"Push Button"

Usually refute a problem and produce proofs.

Satisfiability Modulo Theories

Propositional reasoning + theories.

- Functions
- Linear Arithmetic
- Quantifiers

Examples:

- **veriT**
- cvc5
- Z3

Software Supported Proof Construction

Proof Assistants

Reliability trusted kernel

Effort proof construction routines

Examples:

- **Isabelle/HOL**
- Coq
- Lean

Automation

Must build uppon the kernel.

- Simplifier: replaces equal by equal.
- Integration of automated theorem provers.

Automated Theorem Provers

"Push Button"

Usually refute a problem and produce proofs.

Satisfiability Modulo Theories

Propositional reasoning + theories.

- Functions
- Linear Arithmetic
- Quantifiers

Examples:

- **veriT**
- cvc5
- Z3

Stronger SMT Solvers for Proof Assistants

The Sledgehammer Pipeline

lemma $f(x+5) = f((1 \times 5) + x)$

1. $f(x+5) = f(5+x)$ by ×_unit
2. $x+5 = 5+x$ by cong
3. $x+5 = x+5$ by +_com
4. $\top$ by refl

The Sledgehammer Pipeline

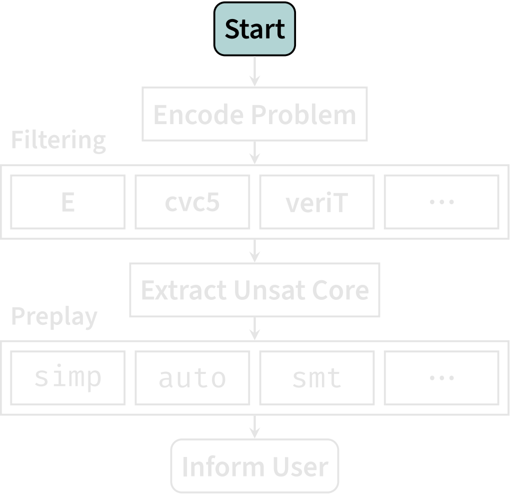
lemma $f(x+5) = f((1 \times 5) + x)$

🔨

The Sledgehammer Pipeline

lemma $f(x+5) = f((1 \times 5) + x)$

Goal $L := f(x+5) = f((1 \times 5) + x)$

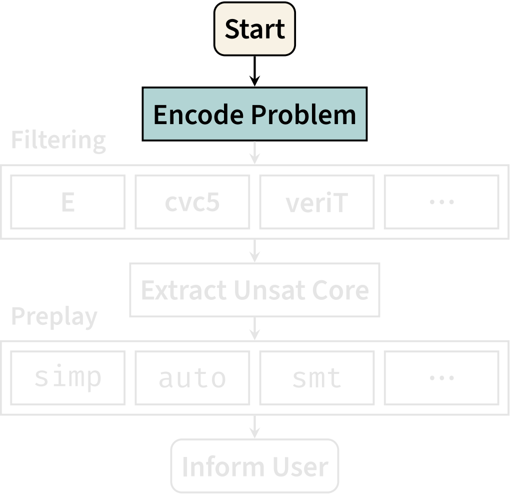


The Sledgehammer Pipeline

lemma $f(x+5) = f((1 \times 5) + x)$

Goal $L := f(x+5) = f((1 \times 5) + x)$

Add B := {×_unit, ×_com, ×_assoc, +_unit, +_com, +_assoc, cong, refl, ...}

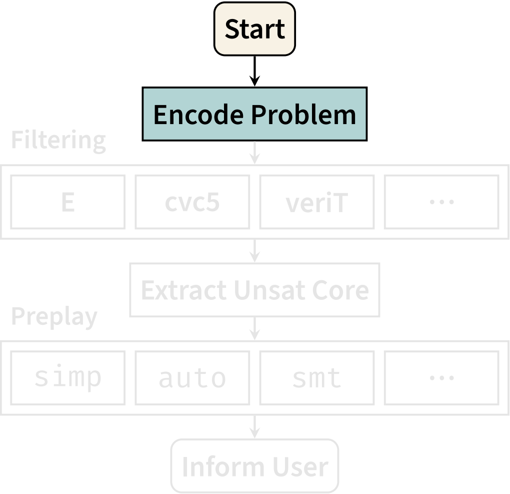


The Sledgehammer Pipeline

lemma $f(x+5) = f((1 \times 5) + x)$

Goal $L := f(x+5) = f((1 \times 5) + x)$

Add B := {**×_unit**, ×_com, ×_assoc, +_unit, **+_com**, +_assoc, **cong**, **refl**, ...}

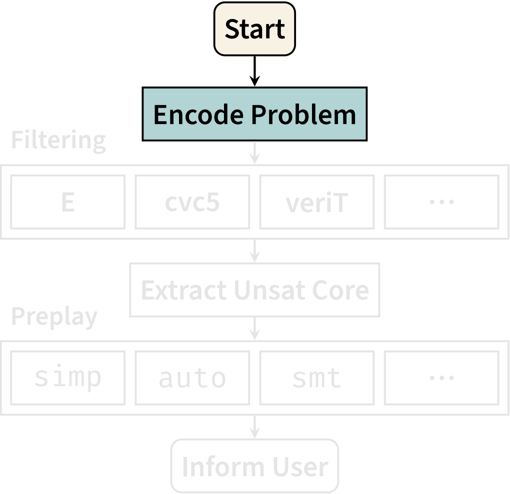


The Sledgehammer Pipeline

lemma $f(x+5) = f((1 \times 5) + x)$

Goal $L := f(x+5) = f((1 \times 5) + x)$

Add B := {×_unit, ×_com, ×_assoc, +_unit, +_com, +_assoc, cong, refl, ...}

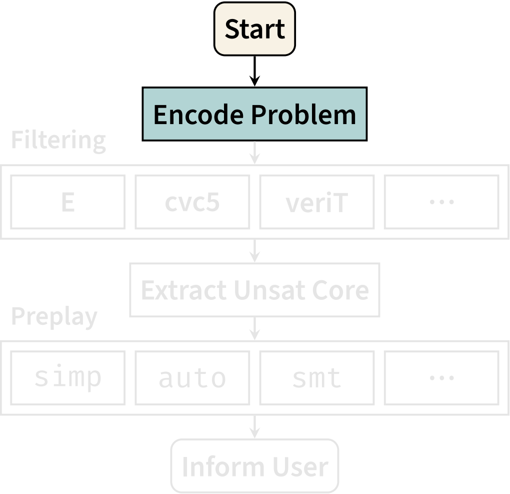


The Sledgehammer Pipeline

lemma $f(x+5) = f((1 \times 5) + x)$

Goal $L := f(x+5) = f((1 \times 5) + x)$

Add $B :=$ {×_unit, ×_com, ×_assoc, ...}



The Sledgehammer Pipeline

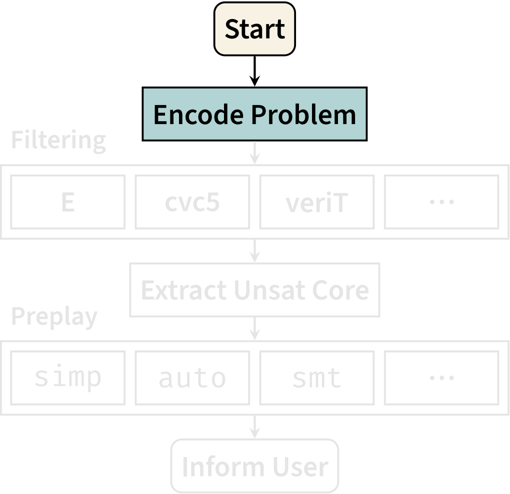
lemma $f(x+5) = f((1 \times 5) + x)$

🔨

Goal $L := f(x+5) = f((1 \times 5) + x)$

Add $B :=$ {×_unit, ×_com, ×_assoc, ...}

Encode $B \wedge \neg L$



The Sledgehammer Pipeline

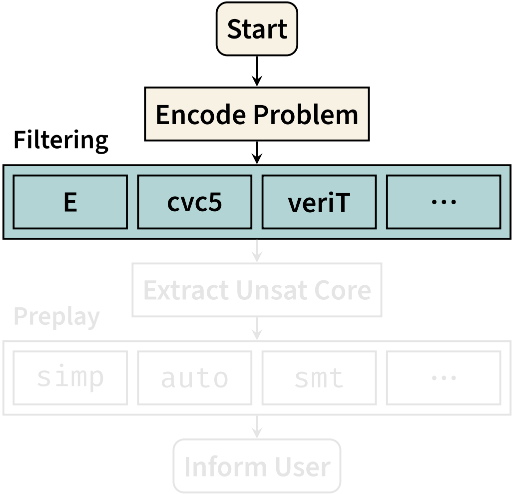
lemma $f(x+5) = f((1 \times 5) + x)$

Goal $L := f(x+5) = f((1 \times 5) + x)$

Add $B :=$ {x_unit, x_com, x_assoc, ...}

Encode $B \wedge \neg L$

Try E, cvc5, veriT, ...



The Sledgehammer Pipeline

lemma $f(x+5) = f((1 \times 5) + x)$

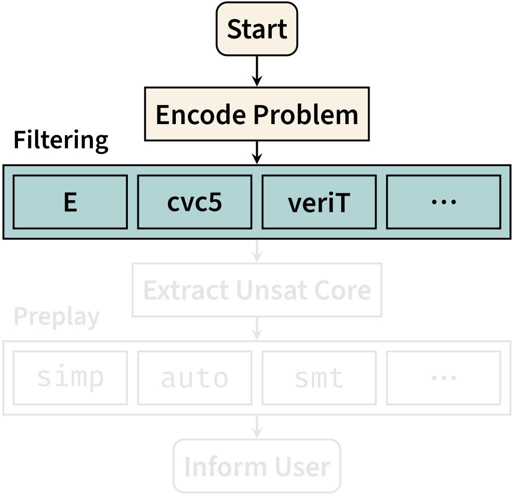
🔨

Goal $L := f(x+5) = f((1 \times 5) + x)$

Add $B :=$ {×_unit, ×_com, ×_assoc, ...}

Encode $B \wedge \neg L$

Try E, cvc5, veriT, ... veriT shows unsat!



The Sledgehammer Pipeline

lemma $f(x+5) = f((1 \times 5) + x)$

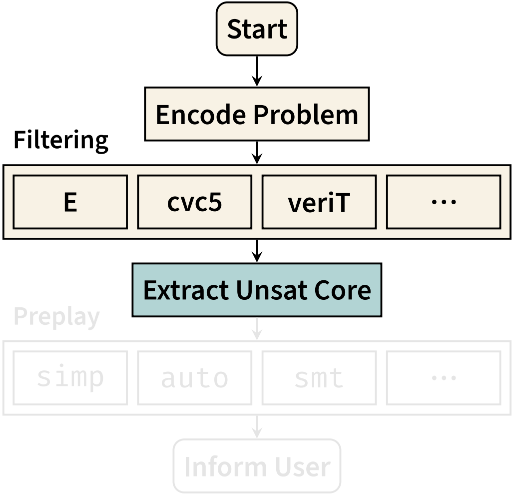
Goal $L := f(x+5) = f((1 \times 5) + x)$

Add $B :=$ {×_unit, ×_com, ×_assoc, ...}

Encode $B \wedge \neg L$

Try E, cvc5, veriT, ... veriT shows unsat!

Core $C :=$ {×_unit, +_com, cong, refl}



The Sledgehammer Pipeline

lemma $f(x+5) = f((1 \times 5) + x)$

Goal $L := f(x+5) = f((1 \times 5) + x)$

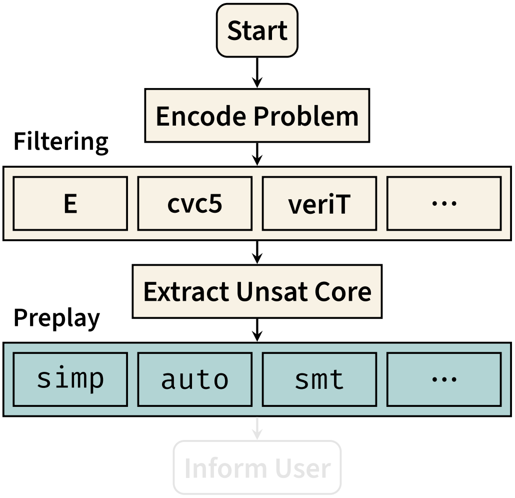
Add $B :=$ {×_unit, ×_com, ×_assoc, ...}

Encode $B \wedge \neg L$

Try E, cvc5, veriT, ... veriT shows unsat!

Core $C :=$ {×_unit, +_com, cong, refl}

Preplay `simp`, `auto`, `smt` on $C \wedge \neg L$, ...



The Sledgehammer Pipeline

lemma $f(x+5) = f((1 \times 5) + x)$

🔨

Goal $L := f(x+5) = f((1 \times 5) + x)$

Add $B :=$ {×_unit, ×_com, ×_assoc, ...}

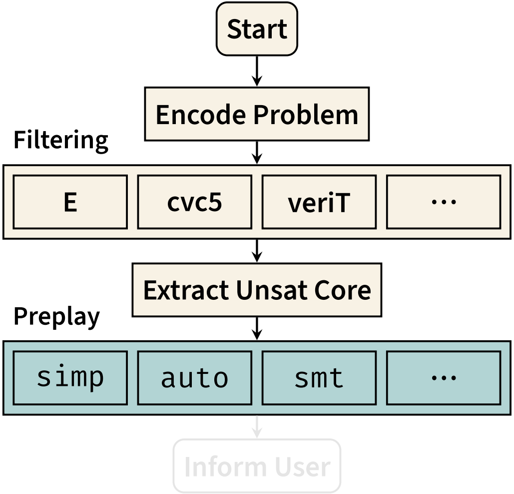
Encode $B \wedge \neg L$

Try E, cvc5, veriT, ... veriT shows unsat!

Core $C :=$ {×_unit, +_com, cong, refl}

Preplay `simp`, `auto`, `smt` on $C \wedge \neg L$, ...

`smt` shows unsat!



The Sledgehammer Pipeline

lemma $f(x+5) = f((1 \times 5) + x)$

Goal $L := f(x+5) = f((1 \times 5) + x)$

Add $B :=$ {×_unit, ×_com, ×_assoc, ...}

Encode $B \wedge \neg L$

Try E, cvc5, veriT, ... veriT shows unsat!

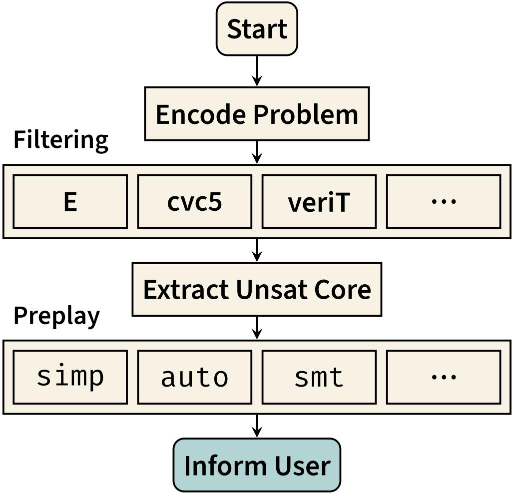
Core $C :=$ {×_unit, +_com, cong, refl}

Preplay `simp`, `auto`, `smt` on $C \wedge \neg L$, ...

`smt` shows unsat!

Done!

Try `smt`: ×_unit, cong, +_com, refl



The Sledgehammer Pipeline

lemma $f(x+5) = f((1 \times 5) + x)$

1. $\top$ by `smt`: ×_unit, cong, +_com, refl

Goal $L := f(x+5) = f((1 \times 5) + x)$

Add $B :=$ {×_unit, ×_com, ×_assoc, ...}

Encode $B \wedge \neg L$

Try E, cvc5, veriT, ... veriT shows unsat!

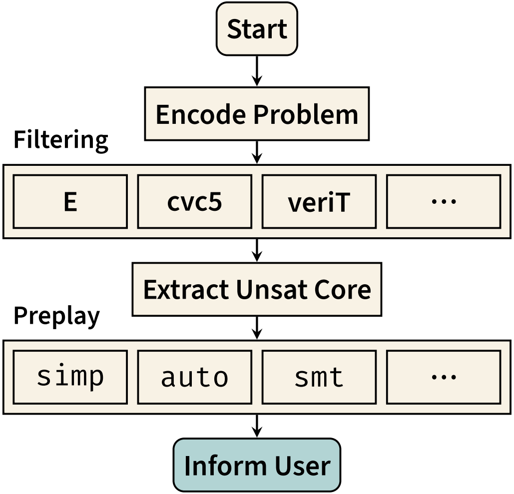
Core $C :=$ {×_unit, +_com, cong, refl}

Preplay `simp`, `auto`, `smt` on $C \wedge \neg L$, ...

`smt` shows unsat!

Done!

Try `smt`: ×_unit, cong, +_com, refl



The Sledgehammer Pipeline

lemma $f(x+5) = f((1 \times 5) + x)$

1. ⊤ by `smt`: ×_unit, cong, +_com, refl

Goal $L := f(x+5) = f((1 \times 5) + x)$

Add $B := \{$×_unit, ×_com, ×_assoc, ...$\}$

Encode $B \wedge \neg L$

Try E, cvc5, veriT, ... veriT shows unsat!

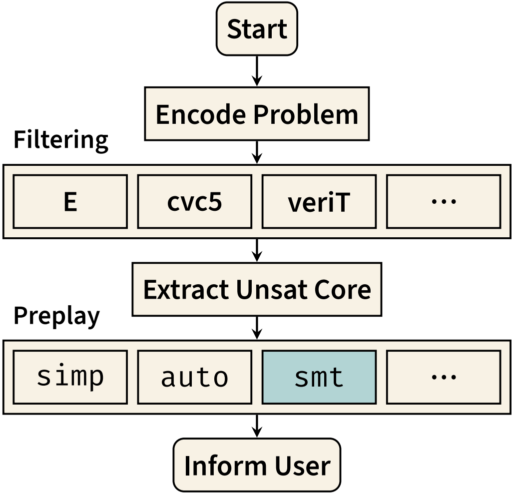
Core $C := \{$×_unit, +_com, cong, refl$\}$

Preplay `simp`, `auto`, `smt` on $C \wedge \neg L$, ...

`smt` shows unsat!

Done!

Try `smt`: ×_unit, cong, +_com, refl



Stronger SMT Solvers

Part 1: Improving Proofs

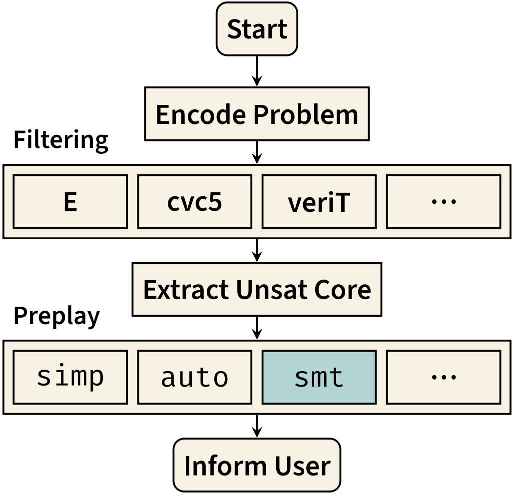
with Mathias Fleury & Martin Desharnais
published at CADE 2021

Part 2: Improving Quantifier Simplification

with Pascal Fontaine
published at FroCoS 2021 🏆

Part 3: A Toolbox for Strategy Schedules

Answers question: if we have limited time, how long should each prover run?
published at PAAR 2022



Part 1: Improving Proofs

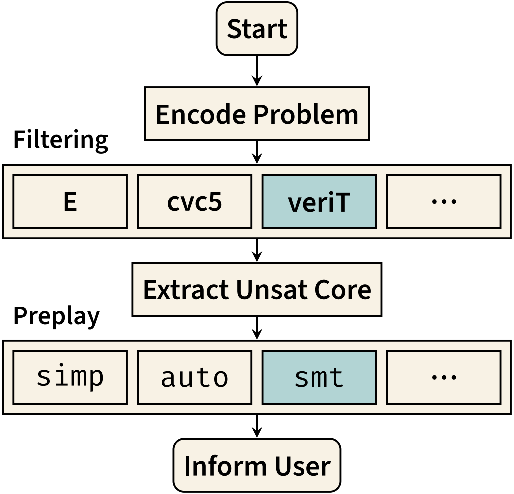
with Mathias Fleury & Martin Desharnais
published at CADE 2021

Part 2: Improving Quantifier Simplification

with Pascal Fontaine
published at FroCoS 2021 🏆

Part 3: A Toolbox for Strategy Schedules

Answers question: if we have limited time, how long should each prover run?
published at PAAR 2022



Part 1: Improving Proofs

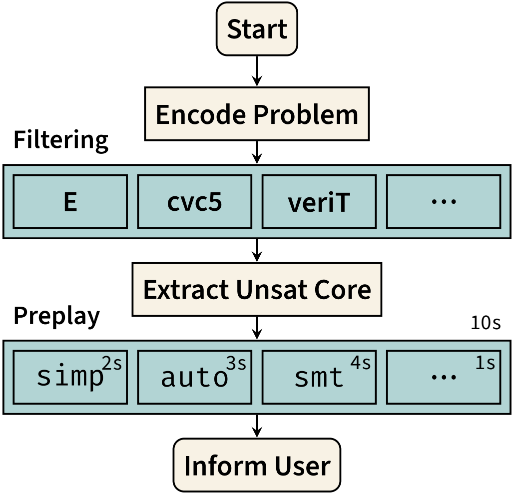
with Mathias Fleury & Martin Desharnais
published at CADE 2021

Part 2: Improving Quantifier Simplification

with Pascal Fontaine
published at FroCoS 2021 🏆

Part 3: A Toolbox for Strategy Schedules

Answers question: if we have limited time, how long should each prover run?
published at PAAR 2022



Part 1: Improving Proofs

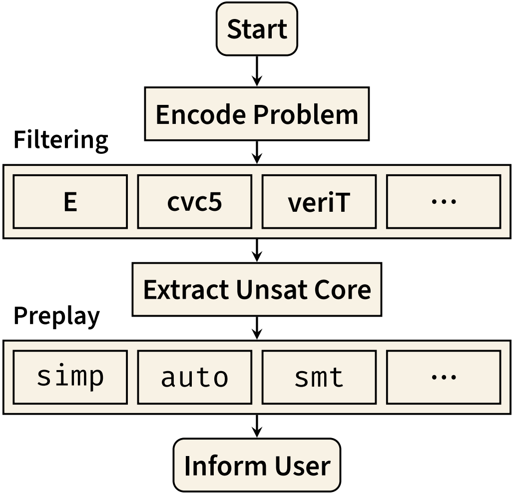
with Mathias Fleury & Martin Desharnais
published at CADE 2021

Part 2: Improving Quantifier Simplification

with Pascal Fontaine
published at FroCoS 2021 🏆

Part 3: A Toolbox for Strategy Schedules

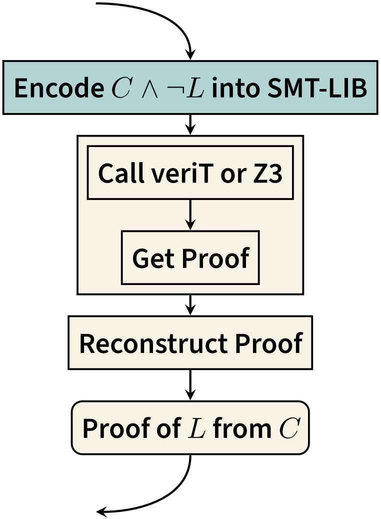
Answers question: if we have limited time, how long should each prover run?
published at PAAR 2022

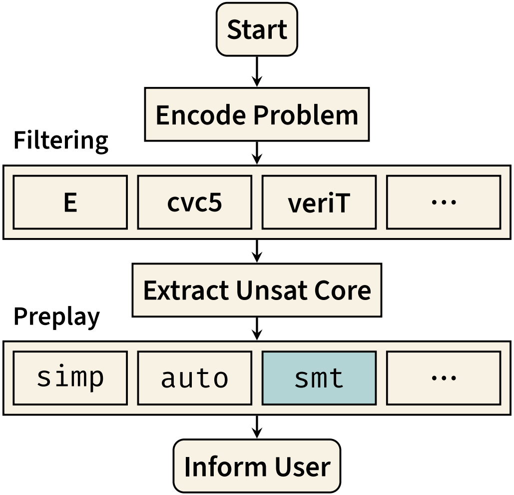


Part I

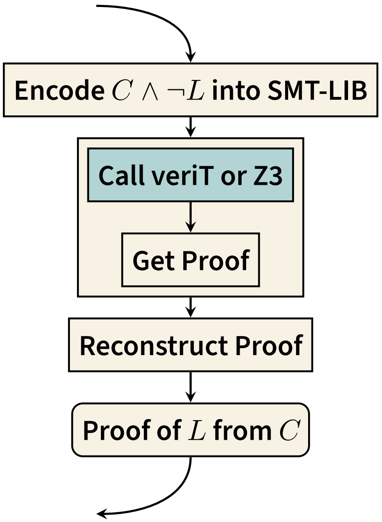
Improving Proofs

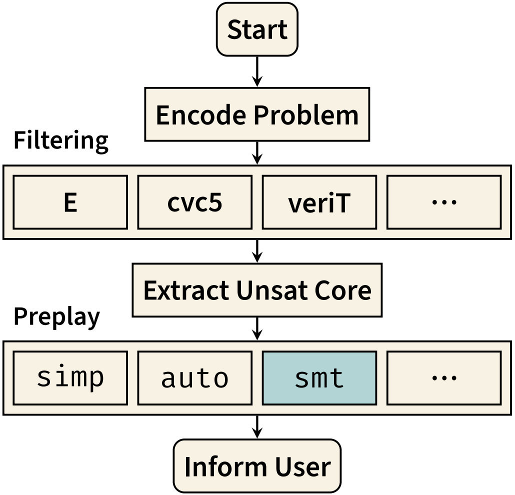
Using Proofs: `smt`



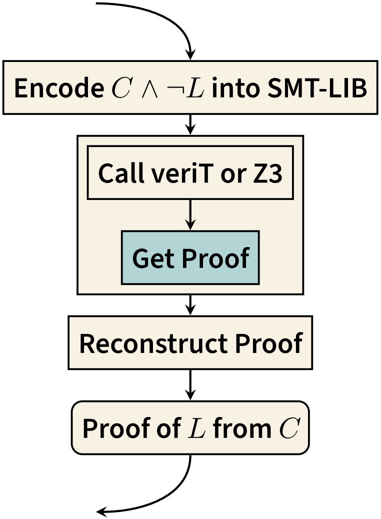


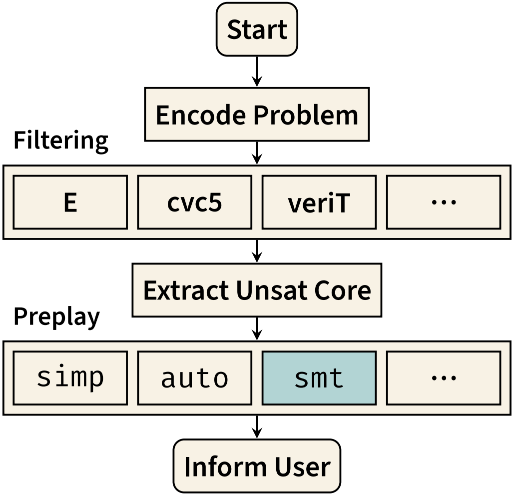
Using Proofs: `smt`



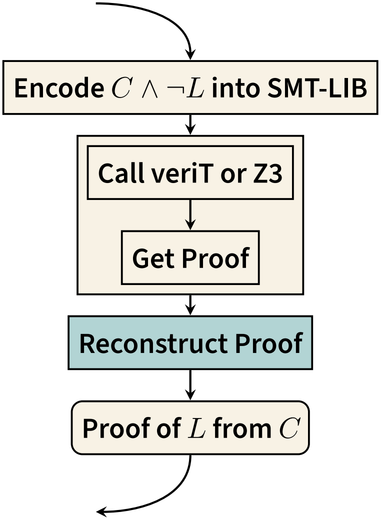


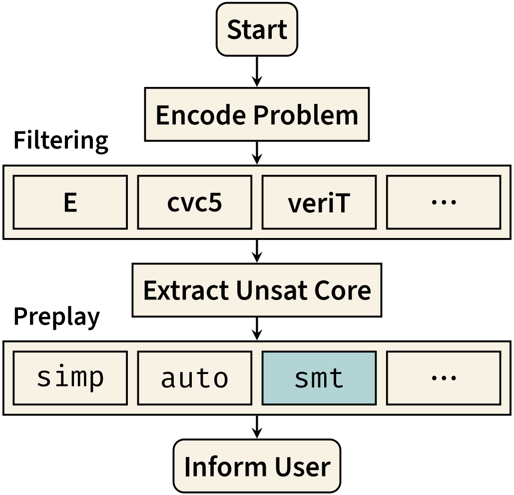
Using Proofs: `smt`



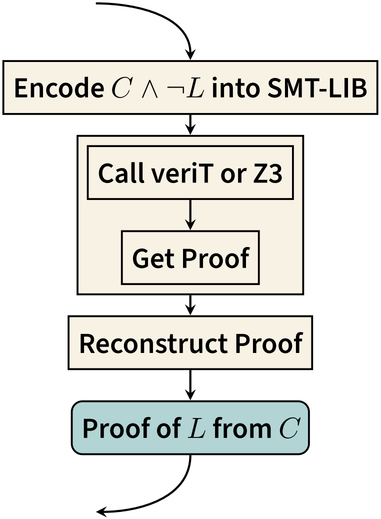


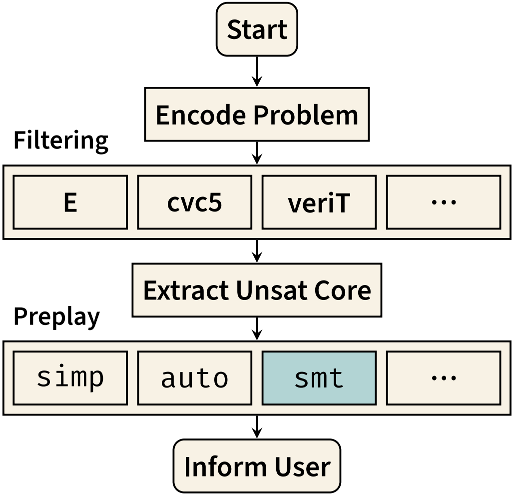
Using Proofs: `smt`





Using Proofs: `smt`





SMT Proof Reconstruction Circa 2018

veriT Proofs

- New in 2017: reasoning about binders. [Barbosa, et al. 2017]
- Reconstruction prototype by Fleury for validation. [Barbosa, et al. 2020]
- Philosophy: fine-grained proofs, natural deduction style.

SMT Proof Reconstruction Circa 2018

veriT Proofs

- New in 2017: reasoning about binders. [Barbosa, et al. 2017]
- Reconstruction prototype by Fleury for validation. [Barbosa, et al. 2020]
- Philosophy: fine-grained proofs, natural deduction style.

The `smt` tactic: Z3 only

- From 2009, by Böhme, et al.
- Stable, but bound to a specific Z3 version.
- Z3 proofs have a different philosophy (macro rules).

SMT Proof Reconstruction Circa 2018

veriT Proofs

- New in 2017: reasoning about binders. [Barbosa, et al. 2017]
- Reconstruction prototype by Fleury for validation. [Barbosa, et al. 2020]
- Philosophy: fine-grained proofs, natural deduction style.

The `smt` tactic: Z3 only

- From 2009, by Böhme, et al.
- Stable, but bound to a specific Z3 version.
- Z3 proofs have a different philosophy (macro rules).

Questions

- Can we make the proofs more rigorous?
- What can we learn from doing reconstruction?
- Is veriT's fine-grained proof & quantifier support useful?

SMT Proof Reconstruction Circa 2018

veriT Proofs

- New in 2017: reasoning about binders. [Barbosa, et al. 2017]
- Reconstruction prototype by Fleury for validation. [Barbosa, et al. 2020]
- Philosophy: fine-grained proofs, natural deduction style.

The `smt` tactic: Z3 only

- From 2009, by Böhme, et al.
- Stable, but bound to a specific Z3 version.
- Z3 proofs have a different philosophy (macro rules).

Questions

- Can we make the proofs more rigorous? **Yes: Alethe!**
- What can we learn from doing reconstruction?
- Is veriT's fine-grained proof & quantifier support useful?

SMT Proof Reconstruction Circa 2018

veriT Proofs

- New in 2017: reasoning about binders. [Barbosa, et al. 2017]
- Reconstruction prototype by Fleury for validation. [Barbosa, et al. 2020]
- Philosophy: fine-grained proofs, natural deduction style.

The `smt` tactic: Z3 only

- From 2009, by Böhme, et al.
- Stable, but bound to a specific Z3 version.
- Z3 proofs have a different philosophy (macro rules).

Questions

- Can we make the proofs more rigorous? **Yes: Alethe!**
- What can we learn from doing reconstruction? **Lessons for the future.**
- Is veriT's fine-grained proof & quantifier support useful?

SMT Proof Reconstruction Circa 2018

veriT Proofs

- New in 2017: reasoning about binders. [Barbosa, et al. 2017]
- Reconstruction prototype by Fleury for validation. [Barbosa, et al. 2020]
- Philosophy: fine-grained proofs, natural deduction style.

The `smt` tactic: Z3 only

- From 2009, by Böhme, et al.
- Stable, but bound to a specific Z3 version.
- Z3 proofs have a different philosophy (macro rules).

Questions

- Can we make the proofs more rigorous? **Yes: Alethe!**
- What can we learn from doing reconstruction? **Lessons for the future.**
- Is veriT's fine-grained proof & quantifier support useful? **Yes: veriT `smt`!**

Alethe Proofs: Basic Structure

$$\cfrac{t_1 \qquad \cfrac{\cfrac{t_2}{t_3}}{\vdots} \;\; \neg t_1}{\bot}\ \text{resolution}$$

$$t_1, t_2 \vdash \bot$$

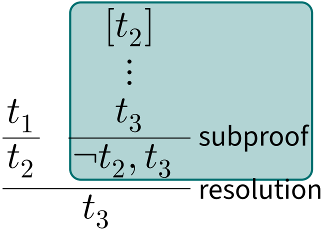
```
(assume a0 t1)
(assume a1 t2)
(step  s1 (cl t3)
        :premises (a1)      :rule rule1)
...
(step s20 (cl (not t1))
        :premises (s19)     :rule rule2)
(step s21 (cl )
        :premises (a0 s20) :rule resolution)
```

Alethe Proofs: Subproofs With Assumptions

$$\dfrac{\dfrac{t_1}{t_2} \qquad \dfrac{\begin{matrix}[t_2]\\ \vdots \\ t_3\end{matrix}}{\neg t_2, t_3}\ \text{subproof}}{t_3}\ \text{resolution}$$

$$t_1 \vdash t_3$$

```
(assume a0   t1)
(step s1 (cl t2)
        :premises (a0)    :rule rule1)
(anchor :step s2)
  (assume s2.a1      t2)
  ...
  (step   s2.s10 (cl t3)
        :premises (s2.s9) :rule rule2)
(step s2 (cl (not t2) t3) :rule subproof)
(step s3 (cl t3)
        :premises (s1 s2) :rule resolution)
```



$$t_1 \vdash t_3$$

```
(assume a0   t1)
(step s1 (cl t2)
        :premises (a0)    :rule rule1)
(anchor :step s2)
  (assume s2.a1      t2)
  ...
  (step   s2.s10 (cl t3)
        :premises (s2.s9) :rule rule2)
(step s2 (cl (not t2) t3) :rule subproof)
(step s3 (cl t3)
        :premises (s1 s2) :rule resolution)
```

Alethe Proofs: Reasoning With Binders

$$\dfrac{\dfrac{\dfrac{}{x \mapsto y \rhd \quad x = y}\text{refl}}{x \mapsto y \rhd f(x) = f(y)}\text{cong}}{\forall x.\, f(x) = \forall y.\, f(y)}\text{bind}$$

$$\vdash \forall x.\, f(x) = \forall y.\, f(y)$$

```
(anchor :step s2 :args ((:= (x S) y)))
  (step s2.s1 (cl (= x y))      :rule refl)
  (step s2.s2 (cl (= (f x) (f y)))
                                :rule cong)
(step s2 (cl (= (forall ((x S)) (f x))
                (forall ((y S)) (f y))))
                                :rule bind)
```

Improving Alethe for Reconstruction

Important Hurdles Solved

- Clear term simplifications
- No implicit clause normalizations
- Certificates for linear arithmetic

Improving Alethe for Reconstruction

Important Hurdles Solved

- Clear term simplifications
- No implicit clause normalizations
- Certificates for linear arithmetic

Other Improvments

- **Complete documentation of the format.**
- Rigorous handling of quantifiers
  - No implicit clausification.
  - $\forall$-instantiation certificate: explicit substitution.
- Proper printing of number constants depending on theory.
- A better algorithm for proof pruning.
- Clever term sharing.
- ...

Clear Term Simplifications

Can we improve proofs of preprocessing?

Clear Term Simplifications

Can we improve proofs of preprocessing?

Proofs

Before a single rule combining all simplifications, **undocumented**

$$\vDash_T \Gamma \rhd t = u$$

Now 17 rules arranged by operators. **Documented** as rewrite rules.
e.g. $x + 0 \rightarrow x$ in sum_simplify.

Clear Term Simplifications

Can we improve proofs of preprocessing?

Proofs

Before a single rule combining all simplifications, **undocumented**

$$\vDash_T \Gamma \rhd t = u$$

Now 17 rules arranged by operators. **Documented** as rewrite rules.
e.g. $x + 0 \rightarrow x$ in sum_simplify.

Reconstruction

Before automatic proof tactics are necessary, with tweaked timeouts.

Now directed use of the simplifier parameterized with the rewrite rules.

No Implicit Clause Normalizations

Clauses in conclusions are sometimes simplified, why?

No Implicit Clause Normalizations

Clauses in conclusions are sometimes simplified, why?

Proofs

Before $\neg\neg\varphi$ implicitly simplified to φ **in the proof module**

Before clauses with complementary literals simplified to $\top$

Before repeated literals implicitly eliminated

Now patch every **proof step**, e.g, add step $\neg\neg\neg\varphi \vee \varphi$ and a resolution step

No Implicit Clause Normalizations

Clauses in conclusions are sometimes simplified, why?

Proofs

Before $\neg\neg\varphi$ implicitly simplified to φ **in the proof module**

Before clauses with complementary literals simplified to $\top$

Before repeated literals implicitly eliminated

Now patch every **proof step**, e.g, add step $\neg\neg\neg\varphi \vee \varphi$ and a resolution step

Reconstruction

Before special case possible at every step!

rule $(\textbf{if}\ \varphi\ \textbf{then}\ \psi_1\ \textbf{else}\ \psi_2) \Rightarrow \neg\varphi \vee \psi_1$

step $(\textbf{if}\ \varphi\ \textbf{then}\ \neg\varphi\ \textbf{else}\ \psi_2) \Rightarrow \neg\varphi$

Now no pollution in rule reconstruction.

step $(\textbf{if}\ \varphi\ \textbf{then}\ \neg\varphi\ \textbf{else}\ \psi_2) \Rightarrow \neg\varphi \vee \neg\varphi$

Certificates for Linear Arithmetic

Reconstruction fails on this LA tautology: $(2x < 3) = (x \leq 1)$ over $\mathbb{Z}$

Why? Strengthening!

Certificates for Linear Arithmetic

Reconstruction fails on this LA tautology: $(2x < 3) = (x \leq 1)$ over $\mathbb{Z}$
Why? Strengthening!

Proofs

Before just a clause of inequalities, no certificate.
Now strengthening documented.

$$(2x < 3) = (x \leq 1)$$

Strengthened: $(2x \leq 2) = (x \leq 1)$

Now certificate: coefficient. Here: $\frac{1}{2}$ and 1.

Reconstruction fails on this LA tautology: $(2x < 3) = (x \leq 1)$ over $\mathbb{Z}$
Why? Strengthening!

Proofs

Before just a clause of inequalities, no certificate.
Now strengthening documented.

$$(2x < 3) = (x \leq 1)$$

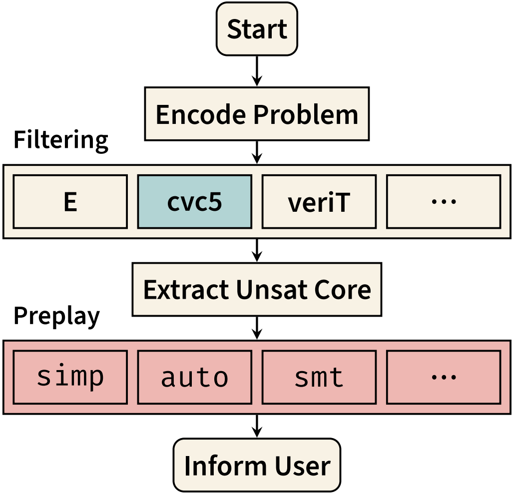
$$\text{Strengthened: } (2x \leq 2) = (x \leq 1)$$

Now certificate: coefficient. Here: $\frac{1}{2}$ and 1.

Reconstruction

Before certificate derived again.
Now reconstruction amounts to calculations.
Now can abstract nested terms: $2 \times (\textbf{if} \top \textbf{ then } 1 \textbf{ else } 0)$ treated as $2 \times x$.

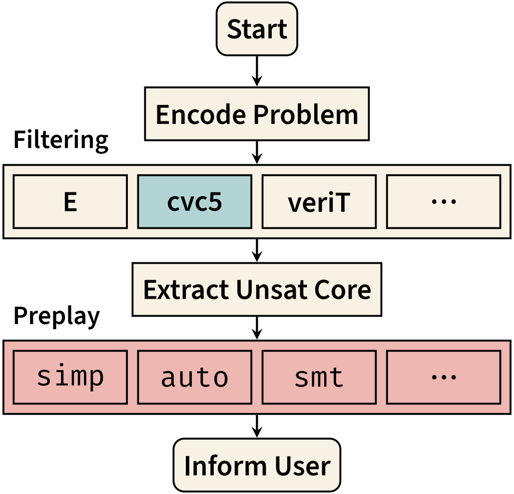
1. Pick an existing theory.
2. Try Sledgehammer on each obligation.



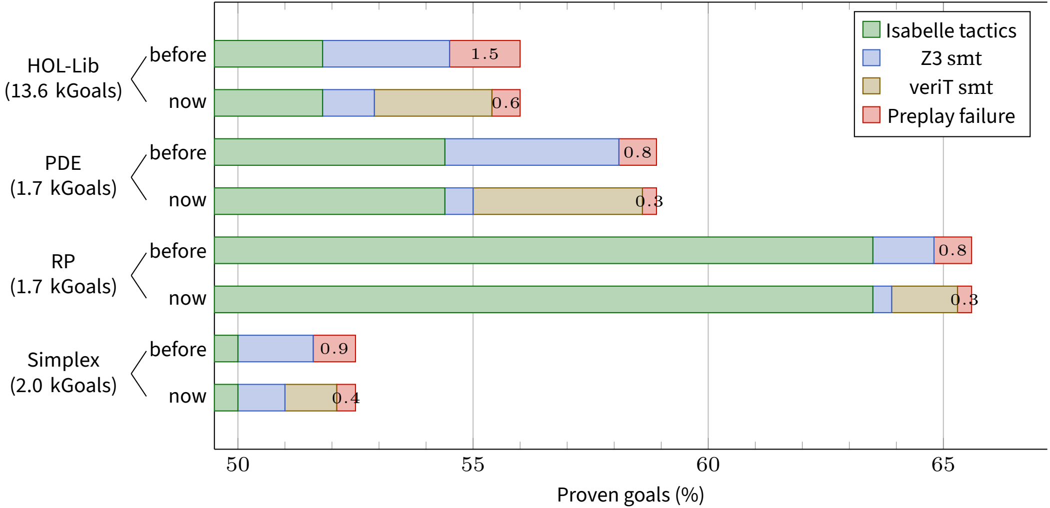
Evaluating smt

1. Pick an existing theory.
2. Try Sledgehammer on each obligation.

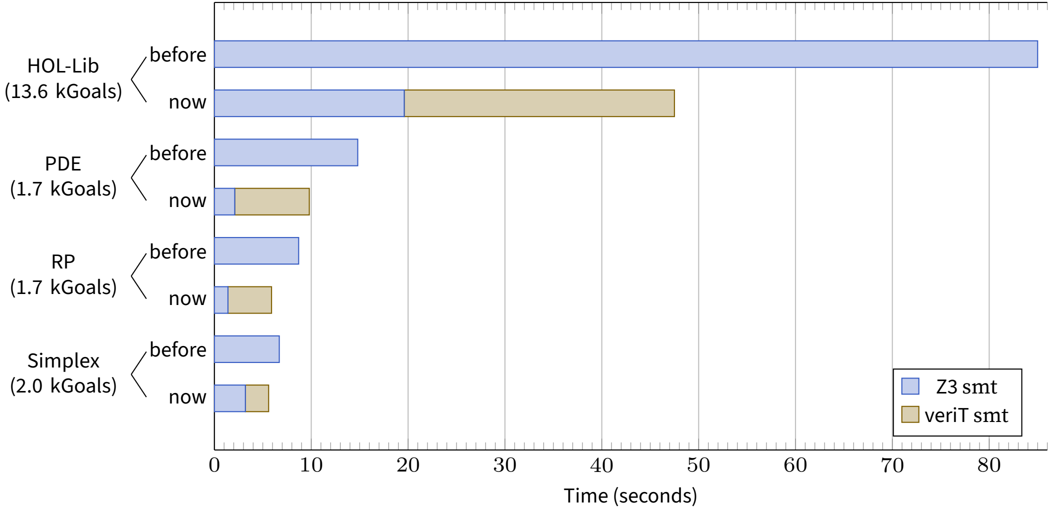
- Did Sledgehammer succeed?
- Which tactic did preplay suggest?
- Preplay failure: there is a proof, but it’s not usable!
- Also: how long does the tactic run?



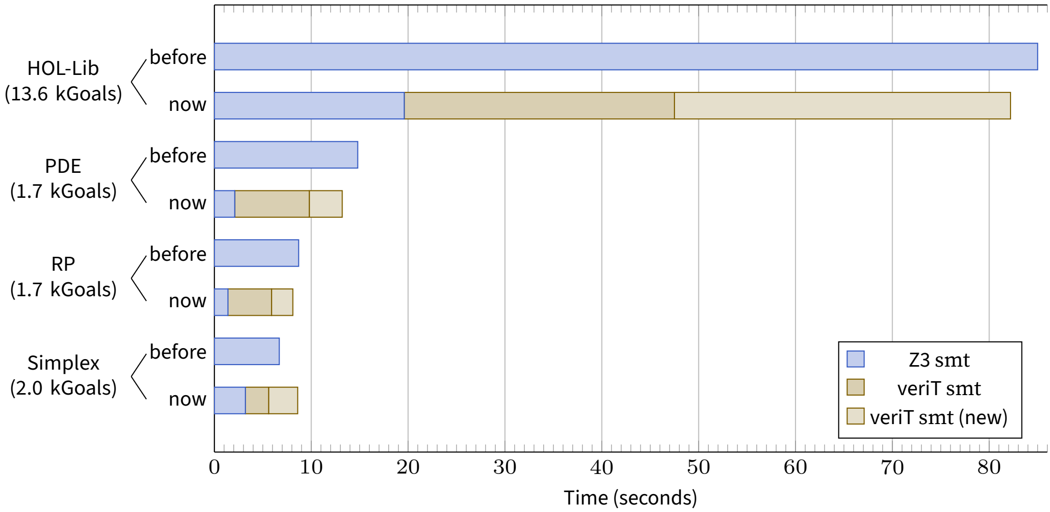
CVC4: Preplay Success Rate



CVC4: Preplay Time (smt only)



CVC4: Preplay Time (smt only)



Conclusion

Reconstruction

- 611 smt-veriT calls in AFP.
- Granular proofs matter.
- Proof size is critical.

SMT Proofs

- Danger of “Proof Rot.”
- Fine-grained proofs can prevent this.

Conclusion

Reconstruction

- 611 smt-veriT calls in AFP.
- Granular proofs matter.
- Proof size is critical.

SMT Proofs

- Danger of “Proof Rot.”
- Fine-grained proofs can prevent this.

Outlook

Alethe

- Support for more features (logics, ...).
- Improve some rules.
- Support in cvc5.

SMT Proofs

- How to support various solvers?
- How to support various consumers?
- Community collaboration.

Part II

Improving Quantifier Simplification

Part 1: Improving Proofs

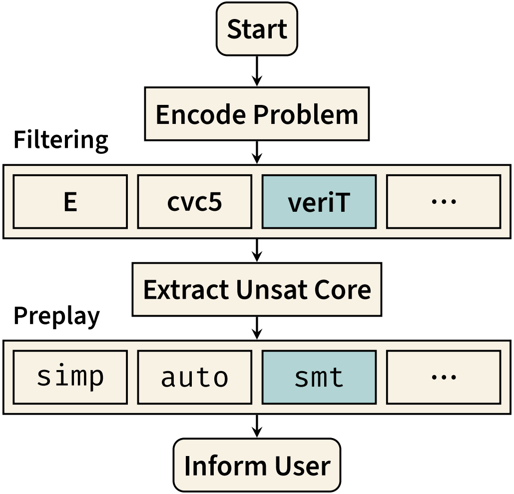
with Mathias Fleury & Martin Desharnais
published at CADE 2021

Part 2: Improving Quantifier Simplification

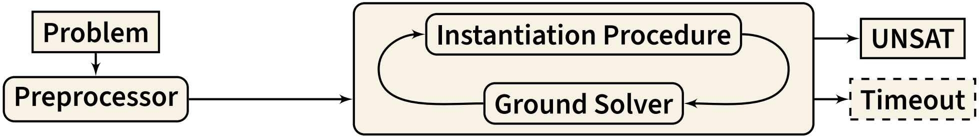
with Pascal Fontaine
published at FroCoS 2021 🏆

Part 3: A Toolbox for Strategy Schedules

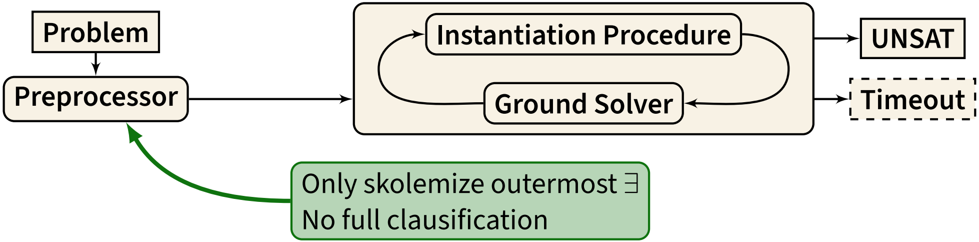
Answers question: if we have limited time, how long should each prover run?
published at PAAR 2022



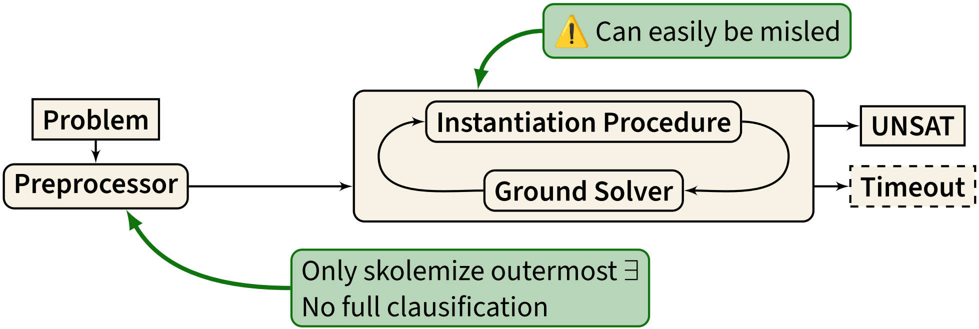
How SMT Solving Works: The Instantiation Loop



How SMT Solving Works: The Instantiation Loop



How SMT Solving Works: The Instantiation Loop



An Example

$$\forall x.\, P(x) \to P(f(x, c))$$

$$\forall y.\, (\forall z.\, P(z) \to P(f(z, y))) \to \neg P(y)$$

$$P(c)$$

An Example

Lemma

$$\forall x.\, P(x) \to P(f(x, c))$$

$$\forall y.\, (\forall z.\, P(z) \to P(f(z, y))) \to \neg P(y)$$

$$P(c)$$

An Example

Lemma

$$\forall x.\, P(x) \to P(f(x, c))$$
$$\forall y.\, (\forall z.\, P(z) \to P(f(z, y))) \to \neg P(y)$$
$$P(c)$$

Using the lemma

An Example

$$\forall x.\, P(x) \rightarrow P(f(x, c))$$

$$\forall y.\, (\forall z.\, P(z) \rightarrow P(f(z, y))) \rightarrow \neg P(y)$$

$$P(c)$$

Instantiate with c

$$\forall x.\, P(x) \rightarrow P(f(x, c))$$

$$\forall y.\, (\forall z.\, P(z) \rightarrow P(f(z, y))) \rightarrow \neg P(y)$$

$$P(c)$$

An Example

$$\forall x.\, P(x) \to P(f(x,c))$$

$$\big(\forall z.\, P(z) \to P(f(z,c))\big) \to \neg P(c)$$

$$P(c)$$

An Example

$$\forall x.\, P(x) \to P(f(x, c))$$

$$(\forall z.\, P(z) \to P(f(z, c))) \to \neg P(c)$$

$$P(c)$$

Skolemize z

An Example

$$\forall x.\, P(x) \to P(f(x, c))$$

$$\big(P(s_1) \to P(f(s_1, c))\big) \to \neg P(c)$$

$$P(c)$$

An Example

$$\forall x.\, P(x) \rightarrow P(f(x, c))$$

$$\big(P(s_1) \rightarrow P(f(s_1, c))\big) \rightarrow \neg P(c)$$

$$P(c)$$

An Example

Instantiate with s_1

$$\forall x.\, P(x) \rightarrow P(f(x, c))$$

$$\big(P(s_1) \rightarrow P(f(s_1, c))\big) \rightarrow \neg P(c)$$

$$P(c)$$

An Example

$$P(s_1) \to P(f(s_1, c))$$

$$\big(P(s_1) \to P(f(s_1, c))\big) \to \neg P(c)$$

$$P(c)$$

Let's use Unification

$$\forall x.\, P(x) \to P(f(x, c))$$

$$\forall y.\, (\forall z.\, P(z) \to P(f(z, y))) \to \neg P(y)$$

$$P(c)$$

Let's use Unification

$$\forall x.\, P(x) \to P(f(x, c))$$

$$\forall y.\, \big(P(s_1(y)) \to P(f(s_1(y), y))\big) \to \neg P(y)$$

$$P(c)$$

Let's use Unification

$$\forall x.\, P(x) \to P(f(x, c))$$
$$\forall y.\, \big(P(s_1(y)) \to P(f(s_1(y), y))\big) \to \neg P(y)$$
$$P(c)$$

Unifier: $y \mapsto c$, $x \mapsto s_1(c)$

Let's use Unification

$$P(s_1(c)) \to P(f(s_1(c), c))$$

$$\big(P(s_1(c)) \to P(f(s_1(c), c))\big) \to \neg P(c)$$

$$P(c)$$

Unifier: $y \mapsto c$, $x \mapsto s_1(c)$

Let's use Unification

$$P(s_1(c)) \to P(f(s_1(c), c))$$
$$(P(s_1(c)) \to P(f(s_1(c), c))) \to \neg P(c)$$
$$P(c)$$

Unifier: $y \mapsto c$, $x \mapsto s_1(c)$

Augment Problem: $\top \to \neg P(c)$

The General Rule

$$\frac{\forall x_1, \dots, x_n . \psi_1 \quad \forall x_{n+1}, \dots, x_m . \varphi[Qy_1, \dots, y_o . \psi_2]}{\forall x_{k_1}, \dots, x_{k_j} . \varphi[b]\sigma}$$

The General Rule

$$\frac{\forall x_1, \ldots, x_n .\, \psi_1 \qquad \forall x_{n+1}, \ldots, x_m .\, \varphi[Q y_1, \ldots, y_o .\, \psi_2]}{\forall x_{k_1}, \ldots, x_{k_j} .\, \varphi[b]\sigma}$$

$b \in \{\top, \bot\}$ dependent on polarity of ψ_1, ψ_2.

The General Rule

$Q \in \{\forall, \exists\}$
first nested quantifier

$$\frac{\forall x_1, \ldots, x_n .\, \psi_1 \qquad \forall x_{n+1}, \ldots, x_m .\, \varphi[Q y_1, \ldots, y_o .\, \psi_2]}{\forall x_{k_1}, \ldots, x_{k_j} .\, \varphi[b]\sigma}$$

$b \in \{\top, \bot\}$ dependent on polarity of ψ_1, ψ_2.

The General Rule

After Skolemization,
ψ_1 and ψ_2 are unified by σ.

$$\frac{\forall x_1, \dots, x_n.\, \psi_1 \qquad \forall x_{n+1}, \dots, x_m.\, \varphi[Q y_1, \dots, y_o.\, \psi_2]}{\forall x_{k_1}, \dots, x_{k_j}.\, \varphi[b]\sigma}$$

$b \in \{\top, \bot\}$ dependent on polarity of ψ_1, ψ_2.

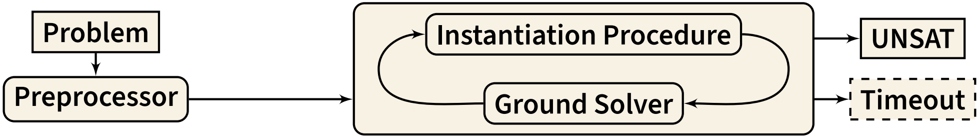
The General Rule

After Skolemization, ψ_1 and ψ_2 are unified by σ.

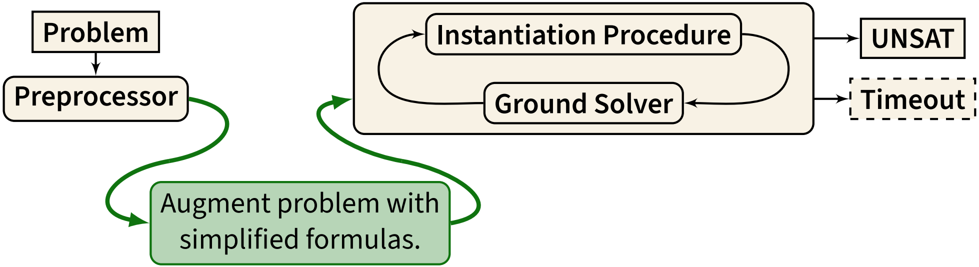
$$\frac{\forall x_1, \ldots, x_n .\, \psi_1 \qquad \forall x_{n+1}, \ldots, x_m .\, \varphi[Q y_1, \ldots, y_o .\, \psi_2]}{\forall x_{k_1}, \ldots, x_{k_j} .\, \varphi[b]\sigma}$$

$b \in \{\top, \bot\}$ dependent on polarity of ψ_1, ψ_2.

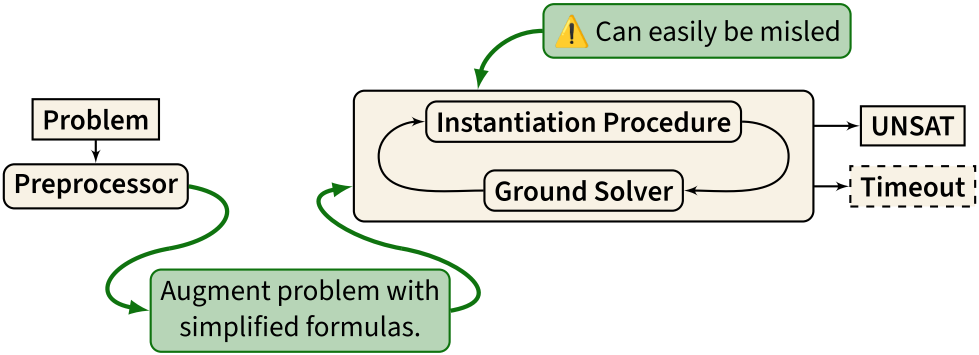
How SMT Solving Works: The Instantiation Loop



How SMT Solving Works: The Instantiation Loop



How SMT Solving Works: The Instantiation Loop



Variants

When to use the rule?

1. **Standard**: remove first quantified subformula.
2. **Eager**: remove subformulas even if they don't start with a quantifier.
3. **Solitary Variable**: remove subformulas with a variable that occurs in no other subformula.

Variants

When to use the rule?

1. **Standard**: remove first quantified subformula.
2. **Eager**: remove subformulas even if they don't start with a quantifier.
3. **Solitary Variable**: remove subformulas with a variable that occurs in no other subformula.

Deletion: remove the second premise (incomplete).
Can be combined with the three variants above.

Experimental Results: Baseline Strategies

vs. Default		Standard	Eager	Solitary	Standard+Del.	Eager+Del.	Solitary+Del.	Total
Solved	31 690	31 927	31 772	31 928	31 733	21 405	21 823	32 151
		+237	+82	**+238**	+43	−10 285	−9 867	+461
Gained		282	**315**	285	291	115	255	475
Lost		**45**	233	47	248	10 400	10 122	14
vs. Virtual Best								
Gained	32 633	83	80	85	**86**	32	76	**125**
Unique		0	**18**	0	5	2	6	

180 s timeout, 38 717 benchmarks, unsat.

Experimental Results: Baseline Strategies

vs. Default		Standard	Eager	Solitary	Standard+Del.	Eager+Del.	Solitary+Del.	Total
Solved	31 690	31 927	31 772	31 928	31 733	21 405	21 823	32 151
		+237	+82	**+238**	+43	−10 285	−9 867	+461
Gained		282	**315**	285	291	115	255	475
Lost		**45**	233	47	248	10 400	10 122	14
vs. Virtual Best								
Gained	32 633	83	80	85	**86**	32	76	**125**
Unique		0	**18**	0	5	2	6	

180 s timeout, 38 717 benchmarks, unsat.

Experimental Results: Baseline Strategies

vs. Default		Standard	Eager	Solitary	Standard+Del.	Eager+Del.	Solitary+Del.	Total
Solved	31 690	31 927	31 772	31 928	31 733	21 405	21 823	32 151
		+237	+82	**+238**	+43	−10 285	−9 867	+461
Gained		282	**315**	285	291	115	255	475
Lost		**45**	233	47	248	10 400	10 122	14
vs. Virtual Best								
Gained	32 633	83	80	85	**86**	32	76	**125**
Unique		0	**18**	0	5	2	6	

180 s timeout, 38 717 benchmarks, unsat.

Experimental Results: Baseline Strategies

vs. Default		Standard	Eager	Solitary	Standard+Del.	Eager+Del.	Solitary+Del.	Total
Solved	31 690	31 927	31 772	31 928	31 733	21 405	21 823	32 151
		+237	+82	**+238**	+43	−10 285	−9 867	+461
Gained		282	**315**	285	291	115	255	475
Lost		**45**	233	47	248	10 400	10 122	14
vs. Virtual Best								
Gained	32 633	83	80	85	**86**	32	76	**125**
Unique		0	**18**	0	5	2	6	

180 s timeout, 38 717 benchmarks, unsat.

Experimental Results: Baseline Strategies

vs. Default		Standard	Eager	Solitary	Standard+Del.	Eager+Del.	Solitary+Del.	Total
Solved	31 690	31 927	31 772	31 928	31 733	21 405	21 823	32 151
		+237	+82	**+238**	+43	−10 285	−9 867	+461
Gained		282	**315**	285	291	115	255	475
Lost		**45**	233	47	248	10 400	10 122	14
vs. Virtual Best								
Gained	32 633	83	80	85	**86**	32	76	**125**
Unique		0	**18**	0	5	2	6	

180 s timeout, 38 717 benchmarks, unsat.

Experimental Results: Baseline Strategies

vs. Default		Standard	Eager	Solitary	Standard+Del.	Eager+Del.	Solitary+Del.	Total
Solved	31 690	31 927	31 772	31 928	31 733	21 405	21 823	32 151
		+237	+82	**+238**	+43	−10 285	−9 867	+461
Gained		282	**315**	285	291	115	255	475
Lost		**45**	233	47	248	10 400	10 122	14
vs. Virtual Best								
Gained	32 633	83	80	85	**86**	32	76	**125**
Unique		0	**18**	0	5	2	6	

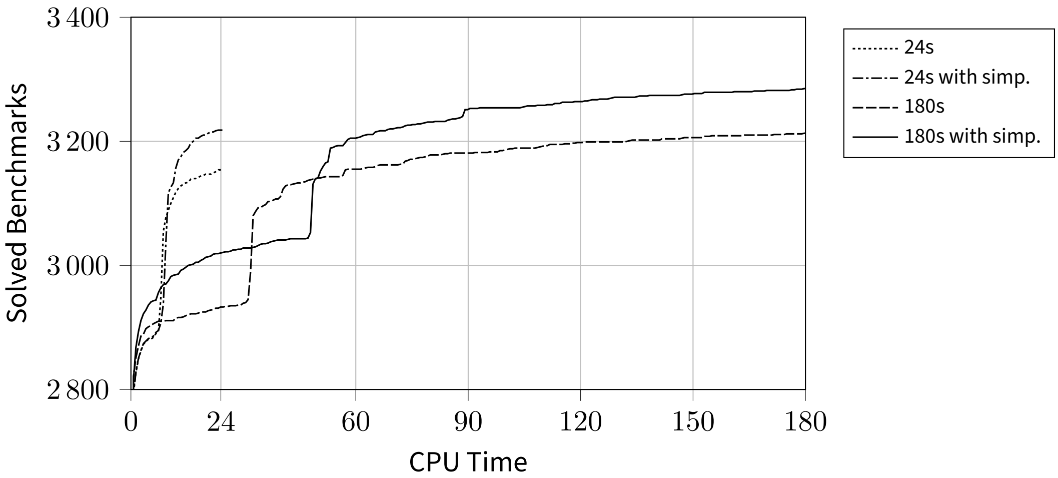
180 s timeout, 38 717 benchmarks, unsat.

Experimental Results: Baseline Strategies

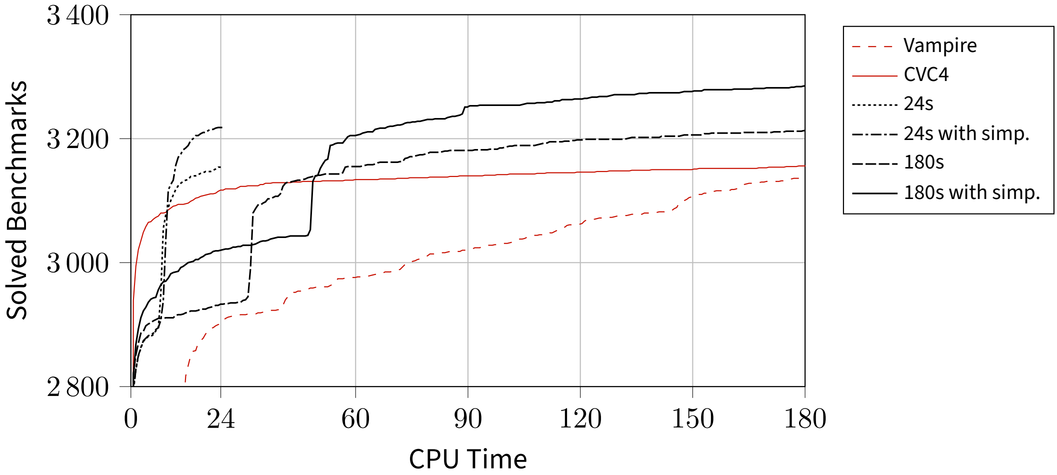
vs. Default		Standard	Eager	Solitary	Standard+Del.	Eager+Del.	Solitary+Del.	Total
Solved	31 690	31 927	31 772	31 928	31 733	21 405	21 823	32 151
		+237	+82	**+238**	+43	−10 285	−9 867	+461
Gained		282	**315**	285	291	115	255	475
Lost		**45**	233	47	248	10 400	10 122	14
vs. Virtual Best								
Gained	32 633	83	80	85	**86**	32	76	**125**
Unique		0	**18**	0	5	2	6	

180 s timeout, 38 717 benchmarks, unsat.

Experimental Results: Schedules (Only Uninterpreted Functions)



Experimental Results: Schedules (Only Uninterpreted Functions)



Conclusion

Quantifier Simplification

- Small things can have big effects.
- We can learn from others.
- The nested structure is tricky.
- It can be exploited,
  but we must be careful.

Conclusion

Quantifier Simplification

- Small things can have big effects.
- We can learn from others.
- The nested structure is tricky.
- It can be exploited,
  but we must be careful.

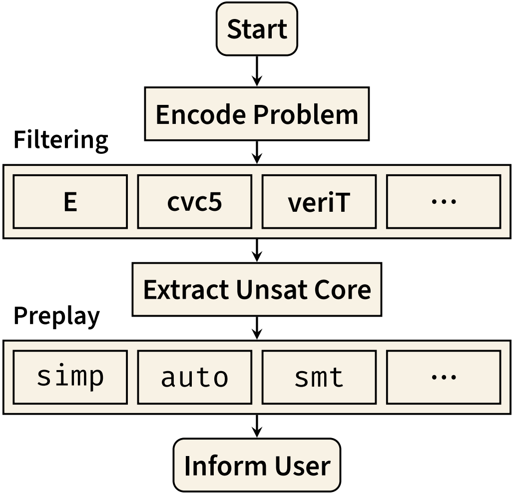
Outlook

Quantifier Reasoning

- Simplification as inprocessing:
  Simplify after each instantiation round.
- More ideas from superposition.
- Can we add those in a granular manner?

Overall Conclusion

- SMT solvers are heterogeneous.
- Many knobs to tweak.
- Specialized solvers can be very useful.
- Practical improvements are hard.



Some Speculation

- Here an expert improved SMT solving for an application.
- Could users adapt solvers? Could specialists contribute to SMT solving?
- What would a “white box” SMT solver look like?

Outlook

Some Speculation

- Here an expert improved SMT solving for an application.
- Could users adapt solvers? Could specialists contribute to SMT solving?
- What would a “white box” SMT solver look like?

Programmable Solver

- Users can adapt the solver to their needs using a DSL.
- Some users already “program” solvers using triggers.
- Idea: DSL based on term rewriting.

Library Solver

- Library Solver: SMT solver as a set of libraries.
- Users pick and choose.
- Potential for tighter integration.

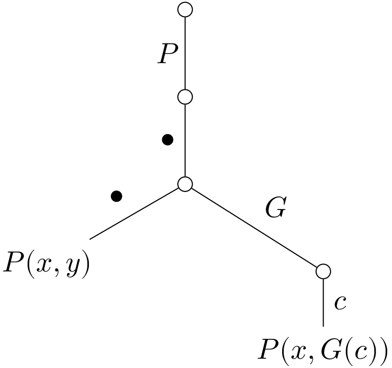
Thank You!

Implementation

- We have to perform many unifiability tests.
- We can use the standard index data structures used by theorem provers.
- In our case: a non-perfect discrimination tree
- and a subsequent unifiability check.
- By treating strongly quantified variables as constants we can avoid creating any new symbols for skolemization!

Non-Perfect Discrimination Tree

Contains:
$\forall x.\, P(x, y)$ as $[P \bullet \bullet]$
$\forall x.\, P(x, G(c))$ as $[P \bullet G\, c]$



Non-Perfect Discrimination Tree

Contains:

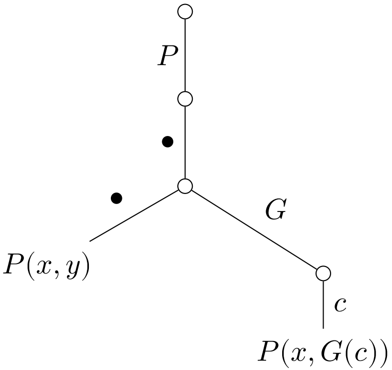
$\forall x.\, P(x, y)$ as $[P \bullet \bullet]$

$\forall x.\, P(x, G(c))$ as $[P \bullet G\, c]$

Lookup:

$\forall x.\, P(c, x)$ as $[P c \bullet]$

matches both formulas



Non-Perfect Discrimination Tree

Contains:

$\forall x.\, P(x, y)$ as $[P \bullet \bullet]$

$\forall x.\, P(x, G(c))$ as $[P \bullet G\, c]$

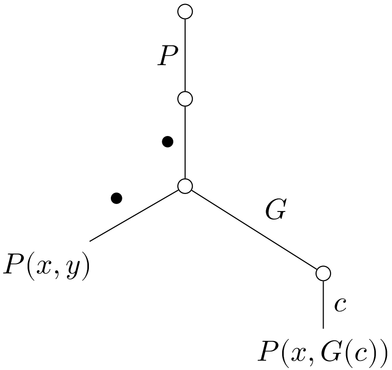
Lookup:

$\forall x. \exists z.\, P(x, z)$

not $[P \bullet s_1 \bullet]$

but $[P \bullet z]$

matches only $P(x, y)$



`verit-schedgen` a Toolbox to Work With Schedules

- Multiple tools to work with **static** strategy schedules
- Can generate schedules
- Focus on simplicity and stability
- Implemented in Python
  - with few extra dependencies
- Available at `https://gitlab.uliege.be/verit/schedgen`

What is a strategy?

- A **strategy** is a full parameterization of the system
- For an SMT solver:
    - select preprocessing methods
    - select instantiation procedures
    - set limits for instantiation procedures
    - …

What is a strategy schedule?

- A finite list $[(t_1, s_1), \ldots, (t_n, s_n)]$
- t_i are time limits
- $s_i \in S$ are strategies
- $\sum_i t_i \leq T$ is the total timeout
- We require that the t_i are from finite set TS of allowed time slices
- In the following $\mathcal{S} = \mathrm{TS} \times S$
- Furthermore, we have training benchmarks (denoted b)

Encoding

$$
\begin{aligned}
T &\geq \sum_{(t,s)\in\mathcal{S}} \dot{x}_{(t,s)} t \\
\dot{x}_s &= \sum_{1\leq i\leq n} \dot{x}_{(t_i,s)} \\
x_b &= \sum_{\dot{x}\in X_b} \dot{x} \text{ with } X_b := \left\{ \dot{x}_{(t,s)} \mid (t,s) \in \mathcal{S} \text{ and } s \text{ solves } b \text{ in time} \leq t \right\} \\
\dot{x}_b |X_b| &\geq x_b \\
\dot{x}_b &\leq x_b + 0.5 \\
\text{maximize} & \sum_{b\in B} \dot{x}_b
\end{aligned}
$$

What's in the box?

- `schedgen-optimize` – generate schedules
- `schedgen-finalize` – generate scripts from a schedule and a template
- `schedgen-simulate` – calculate the benchmarks solved by a schedule
- `schedgen-query` – list unsolved benchmarks, compare schedules
- `schedgen-visualize` – inspect a schedule visually

Walkthrough: Input Data

```
benchmark    ; logic ; strategy      ; solved ; time
base01.smt2 ; UF    ; base-strategy ; yes    ; 0.5189
base02.smt2 ; UF    ; base-strategy ; yes    ; 0.2164
base03.smt2 ; UF    ; base-strategy ; yes    ; 0.1754
...
```

This is artificial example data. All exampes are included in the source code repository.

Walkthrough: `schedgen-optimize`

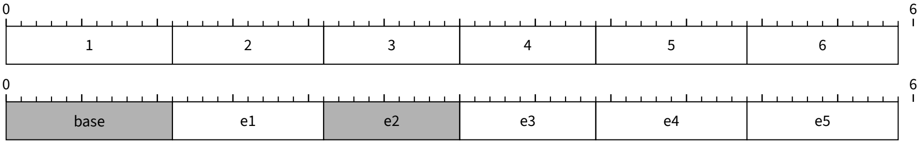
```
$ schedgen-optimize.py
    -l UF --epsilon 0.1 -t 6 \
    -s 0.5 1.0 2 3 4 5 6 \
      --pre-schedule one_second_schedule.csv \
      --pre-schedule-time 1 \
    -c -d contrib/example_data.csv \
        contrib/example_schedule.csv
```

Walkthrough: Generated Schedule

```
time  ; strategy
1.100 ; base-strategy
1.000 ; extra01
0.900 ; extra02
...
```

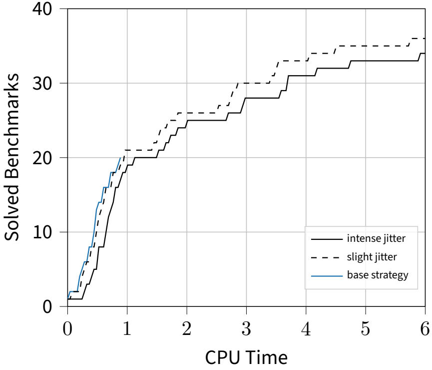
Walkthrough: Visualize

```
$ schedgen-visualize.py -t 6 -p out.pgf \
    -a contrib/example_shorthand.csv \
       contrib/example_schedule.csv
```



Walkthrough: Simulate

```
$ schedgen-simulate.py -l UF -t 6 \
    -c -d contrib/example_data.csv \
    --mu 0.05 --sigma 0.01 --seed 1 \
    contrib/example_schedule.csv simulation_1.txt
```



Walkthrough: Query

```
$ schedgen-query.py -c -d contrib/example_data.csv \
     -q unsolved contrib/example_schedule.csv
   special01.smt2
   unsolved.smt2
```

- `compare` Solved by virtual best solver, but not the schedule
- `best` Virtual best solver (score and solved benchmarks)
- `schedule` Schedule performance (score and solved benchmarks)

Does it work?

- SMT-COMP 2020, 2021, 2022
- Isabelle/HOL `smt` tactic: best strategy, three complementary strategies
  - Best: only timeslice is 3 s, generate 3 s schedule
  - Complementary: same, but 9 s schedule,
- Evaluate new features: generate schedules with and without

Does it work?

Solved	Split 1	Split 2	Split 3	Split 4	Split 5	Arith. Mean (σ)	
virtual best	1355	1318	1328	1293	1338	1326	(23.1)
generated	**1349**	**1306**	**1317**	**1283**	**1326**	**1316**	**(24.4)**
greedy	1340	1303	1314	1275	1326	1312	(24.7)
best strategy	1311	1267	1280	1243	1299	1280	(26.7)
PAR-2 score						Arith. Mean (σ)	
virtual best	160 501	174 213	170 347	182 938	167 371	171 074	(8 316)
generated	**164 388**	**179 811**	**175 453**	**187 851**	**172 102**	**175 921**	**(8 736)**
greedy	169 183	183 040	178 817	192 482	173 655	179 435	(8 974)
best strategy	176 844	192 438	187 772	201 248	180 966	187 854	(9 606)

9000 benchmarks. Five splits of 7200 training benchmarks and 1800 evaluation benchmarks.