Designing Extensible Theory Solvers

Cesare Tinelli

Frontiers of Combining Systems 2017

Sep 29, 2017

The University of Iowa

Credits

Based on joint work with

Andrew Reynolds, *Dejan Jovanović and Clark Barrett*

The Growth of SMT Solvers

More and more applications are leveraging SMT solvers

SMT solvers keep growing and evolving

E.g., they are now supporting many new theories

The Growth of SMT Solvers

More and more applications are leveraging SMT solvers

SMT solvers keep growing and evolving

E.g., they are now supporting many new theories

- unbounded strings with length constraints [39, 31],
- sequences with concatenation and extraction
- (co-)algebraic datatypes [33],
- finite sets with cardinality constraints [5],
- finite relations with transitive closure
- floating-point arithmetic [13]
- non-linear integer arithmetic
- non-linear real arithmetic (with transcendental functions)

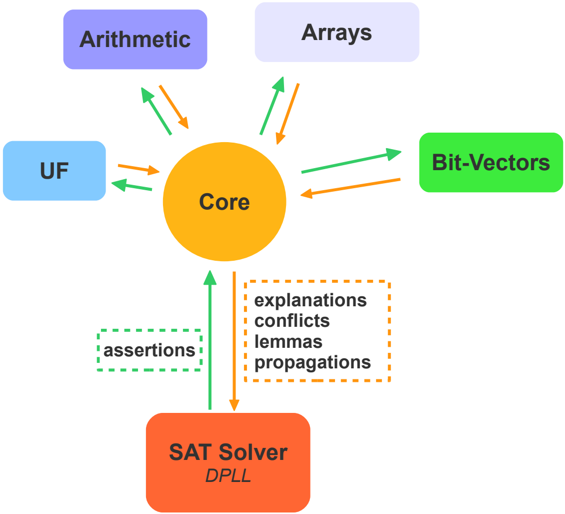
General architectures for SMT solvers

One general architecture, DPLL(T), is well understood and established

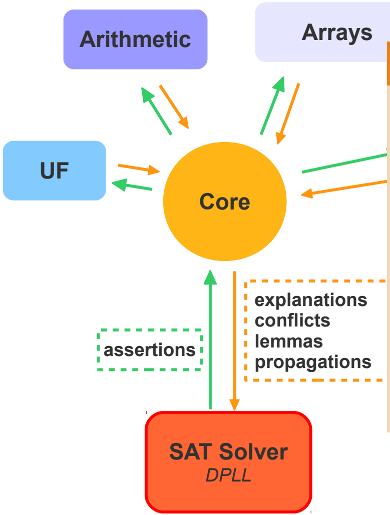
Its basic version is limited to quantifier-free formulas

T is the specific *background theory* supported by the solver

DPLL(T) architecture



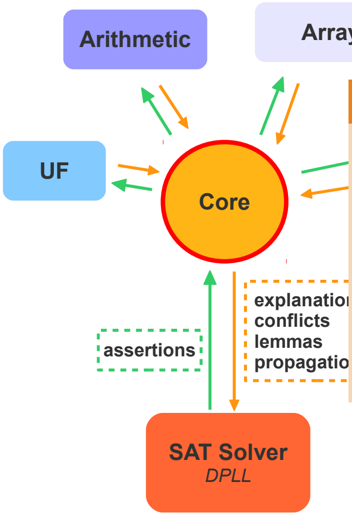
DPLL(T) architecture



SAT Engine

- Only sees Boolean skeleton of problem
- Builds partial model by assigning truth values to literals
- Sends these literals to the core as assertions

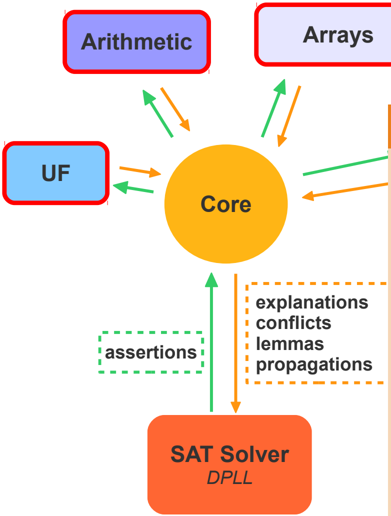
DPLL(T) architecture



Core

- Sends each assertion to the appropriate theory
- Sends deduced literals to other theories/SAT solver
- Handles theory combination

DPLL(T) architecture



Theory Solvers

- Decide T-satisfiability of conjunctions of theory literals
- Are backtrackable
- Generate *explanations*, lemmas and *conflicts*
- Propagate theory literals

The proliferation of theory solvers

New and established theory-specific subsolvers share several functionalities:

- simplifying/normalizing constraints
- reporting conflicts
- propagating literals
- returning lemmas
- producing explanations and proofs
- ...

The proliferation of theory solvers

New and established theory-specific subsolvers share several functionalities:

- simplifying/normalizing constraints
- reporting conflicts
- propagating literals
- returning lemmas
- producing explanations and proofs
- ...

There is a need to express their common features from both a formal and an engineering perspective

Our experience with developing theory solvers

Lesson 1

Term simplification is crucial for performance and scalability

Our experience with developing theory solvers

Lesson 1

Term simplification is crucial for performance and scalability

Lesson 2

New theory solvers can often be built on top of existing solvers

Stratified solvers

In general, a theory solver can be built in layers:

- lower layers are simpler/more efficient than higher layers
- higher layers implement a larger fragment of the constraint language
- higher layers increase the solver's *refutation recall*
- abstraction and refinement can be used to connect the layers

Refutation Recall?

Solvers are classified in theory along these binary dimensions:

- refutation soundness
- refutation completeness
- solution soundness
- solution completeness
- termination

Refutation Recall?

Solvers are classified in theory along these binary dimensions:

- refutation soundness
- refutation completeness
- solution soundness
- solution completeness
- termination

In practice,

- most solvers are refutation and solution sound
- many solvers are refutation or solution incomplete
- solvers for newer theories are rarely terminating

Refutation Recall?

Solvers are classified in theory along these binary dimensions:

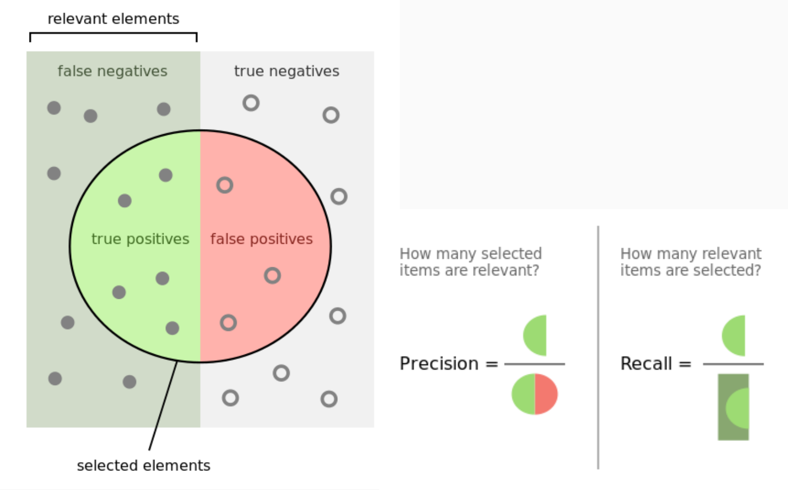
- refutation soundness
- refutation completeness
- solution soundness
- solution completeness
- termination

Problem

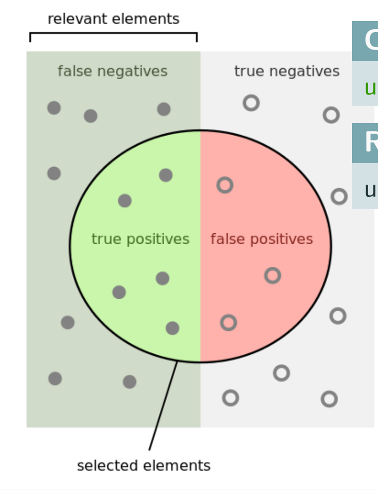
These binary dimensions are too coarse for proper analysis!

- most solvers are refutation and solution sound
- many solvers are refutation or solution incomplete
- solvers for newer theories are rarely terminating

Information Retrieval to the rescue



Information Retrieval to the rescue

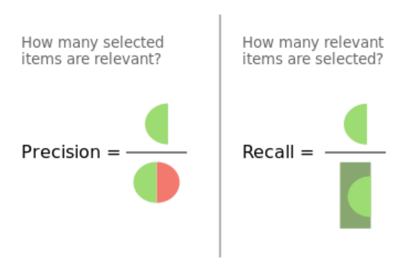


Classification problem

unsat vs. sat, unknown, timeout

Relevant elements

unsat input problems in fragment



Back to theory solvers

Challenge

How to extend modularly a theory solver for fragment of a theory T to a larger fragment of T
while

1. maintaining precision at 100%
2. increasing recall over larger fragment

Focus of this talk

Theories T with signature

$$\Sigma_T = \Sigma_T^{\mathrm{b}} \cup \Sigma_T^{\mathrm{e}}$$

with Σ_T^{b} a *basic signature* and Σ^{e} an *extension signature*

Focus of this talk

Theories T with signature

$$\Sigma_T = \Sigma_T^{\mathrm{b}} \cup \Sigma_T^{\mathrm{e}}$$

with Σ_T^{b} a *basic signature* and Σ^{e} an *extension signature*

Assumptions

1. Σ_T^{b} and Σ_T^{e} share sorts but not function symbols
2. extension symbols in formulas are applied only to vars
3. A bjective mapping

$$\xi : Z \to \{\, f(\bar{x}) \mid f \in \Sigma_T^{\mathrm{e}} \,\}$$

with Z a distinguished set of *abstraction* variables

Example

$\Sigma_{\mathrm{A}}^{\mathrm{b}}$ basic signature for integer arithmetic (Int, $\cdot$, $+$, $-$, 0, 1, $\ldots$)

$\Sigma_{\mathrm{A}}^{\mathrm{e}}$ extension signature for integer arithmetic ($\times$)

$$\Sigma_{\mathrm{A}} = \Sigma_{\mathrm{A}}^{\mathrm{b}} \cup \Sigma_{\mathrm{A}}^{\mathrm{e}}$$

$$\begin{aligned}
\mathsf{F} &= \{\, x_6 + x_5 \times x_3 \approx x_5,\; x_5 - 3 \approx x_1 \times x_2 \vee x_5 > 4 \,\} \\
\mathsf{F}_{\mathrm{b}} &= \{\, x_6 + z_{5,3} \approx x_5,\; x_5 - 3 \approx z_{1,2} \vee x_5 > 4 \,\} \\
\mathsf{F}_{\mathrm{e}} &= \{\, z_{5,3} \approx x_5 \times x_3,\; z_{1,2} \approx x_1 \times x_2 \,\} \\
\xi &= \{\, z_{5,3} \mapsto x_5 \times x_3,\; z_{1,2} \mapsto x_1 \times x_2,\; \ldots \,\} \\
\lceil \mathsf{F}_{\mathrm{b}} \rceil &= \mathsf{F}_{\mathrm{b}}\xi \;=\; \mathsf{F}
\end{aligned}$$

Example

Σ_A^b basic signature for integer arithmetic (Int, $\cdot$, $+$, $-$, 0, 1, $\ldots$)

Σ_A^e extension signature for integer arithmetic ($\times$)

$$\Sigma_A = \Sigma_A^b \cup \Sigma_A^e$$

$$\begin{aligned}
F &= \{\, x_6 + x_5 \times x_3 \approx x_5,\ x_5 - 3 \approx x_1 \times x_2 \vee x_5 > 4 \,\} \\
F_b &= \{\, x_6 + z_{5,3} \approx x_5,\ x_5 - 3 \approx z_{1,2} \vee x_5 > 4 \,\} \\
F_e &= \{\, z_{5,3} \approx x_5 \times x_3,\ z_{1,2} \approx x_1 \times x_2 \,\} \\
\xi &= \{\, z_{5,3} \mapsto x_5 \times x_3,\ z_{1,2} \mapsto x_1 \times x_2,\ \ldots \,\} \\
\lceil F_b \rceil &= F_b\xi \ = \ F
\end{aligned}$$

Observe

$$\lceil F_b \rceil \ = \ F \ \equiv_A \ \exists z_{5,3} \exists z_{1,2}\, F_b \wedge F_e$$

Abstract DPLL(T)

Abstractly, the core of a DPLL(T) solver maintains two evolving data structures:

1. A *context* M, a sequence of literals from a set $\mathcal{L}$
2. A *clause set* F, a set of clauses over $\mathcal{L}$

M is initially empty

F is initially a CNF of input formula

$\mathcal{L}$ is finite and includes all literals in initial F

Basic theory solver in DPLL(T) systems

$$
\begin{array}{lcl}
\texttt{type}\ \text{contex} & = & \text{literal sequence} \\
\texttt{type}\ \text{response} & = & \texttt{Learn of}\ \text{clause} \mid \texttt{Infer of}\ \text{literal} \\
 & \mid & \texttt{Sat}\ \text{of model} \mid \texttt{Unknown}
\end{array}
$$

Solve$_T$(M): context $\to$ response

`if` $\varphi = \ell_1 \vee \ldots \vee \ell_n,\ \ \models_T \varphi,\ \ \mathsf{M} \not\models_{\mathrm{p}} \varphi$ for some $\ell_1, \ldots, \ell_n \subseteq \mathcal{L}$

`Learn(`φ`)`

`else if` $\mathsf{M} \models_T \ell$ for some $\ell \in \mathcal{L} \setminus \mathsf{M}$

`Infer(`ℓ`)`

`else if` $\mathcal{I} \models \mathsf{M}$ for some T-model $\mathcal{I}$

`Sat(`$\mathcal{I}$`)`

`else`

`Unknown`

Leveraging the state of the art

Current theory solvers have functionalities that can be leveraged to handle extended contexts $\mathsf{M} = \mathsf{M}_{\mathrm{b}} \cup \mathsf{M}_{\mathrm{e}}$:

- Computing an congruence relation $\approx_{\mathsf{M}}$ over terms in $\mathcal{T}(\mathsf{M})$, where $s \approx_{\mathsf{M}} t$ only if $\mathsf{M} \models_{\mathcal{T}} s \approx t$
- Computing *simplified* forms $t{\downarrow}$ of terms t, where $\models_{\mathcal{T}} t \approx t{\downarrow}$

Static simplification

In DPLL(T) architectures, simplified forms are useful to

theory solvers: to reduce the number of cases

Ex: $(t_1 < t2)\!\downarrow = p > 0$

the SAT engine: to abstract different atoms by the same var

Ex: $\{(x \times 2 > 8), \neg(4 < x)\}\!\downarrow = \{(x > 4), \neg(x > 4)\}$

However, they are mostly applied once, to the input formula

Dynamic simplification

Claim

It is helpful to apply the same simplification technique dynamically (as M changes) and modulo $\approx_{\mathsf{M}}$

Context-dependent simplification

Assume $\bar{x} \approx_{\mathsf{M}} \bar{s}$ for variables $\bar{x}$ and terms $\bar{s}$ from $\mathcal{T}(\mathsf{M})$. Then,

$$\mathsf{M} \models_{T} t \approx (t\sigma)\downarrow$$

where $\sigma = \{\, \bar{x} \mapsto \bar{s} \,\}$ (called a *derivable substitution*)

Context-dependent simplification

Assume $\bar{x} \approx_{\mathsf{M}} \bar{s}$ for variables $\bar{x}$ and terms $\bar{s}$ from $\mathcal{T}(\mathsf{M})$. Then,

$$\mathsf{M} \models_T t \approx (t\sigma){\downarrow}$$

where $\sigma = \{\, \bar{x} \mapsto \bar{s} \,\}$ (called a *derivable substitution*)

Reduction to basics

Now suppose $t = f(\bar{x})$ and $z \approx f(\bar{x}) \in \mathsf{M}$

If $(t\sigma){\downarrow}$ is a Σ^{b}_T-term, then

$$z \approx f(\bar{x}) \text{ can be simplified to } z \approx (t\sigma){\downarrow}$$

and handled by the basic solver

Context-dependent simplification

Assume $\bar{x} \approx_{\mathsf{M}} \bar{s}$ for variables $\bar{x}$ and terms $\bar{s}$ from $\mathcal{T}(\mathsf{M})$. Then,

$$\mathsf{M} \models_T t \approx (t\sigma){\downarrow}$$

where $\sigma = \{\, \bar{x} \mapsto \bar{s} \,\}$ (called a *derivable substitution*)

Example

Let $\mathsf{M} = \{\, u \approx z_{1,1},\ y_1 \approx w + 2,\ y_1 - w \approx 2,\ z_{1,1} \approx y_1 \times y_1 \,\}$

$\sigma = \{\, y_1 \mapsto 3 \,\}$ is a derivable substitution

Suppose $(y_1 \times y_1)\sigma{\downarrow} = (3 \times 3){\downarrow} = 9$

Then the theory solver can infer the (basic) equality $u \approx 9$

Context-dependent simplification

Assume $\bar{x} \approx_{\mathsf{M}} \bar{s}$ for variables $\bar{x}$ and terms $\bar{s}$ from $\mathcal{T}(\mathsf{M})$. Then,

$$\mathsf{M} \models_{\mathcal{T}} t \approx (t\sigma){\downarrow}$$

where $\sigma = \{\, \bar{x} \mapsto \bar{s} \,\}$ (called a *derivable substitution*)

Example

Let $\mathrm{M_b} = \{\, x_1 \not\approx x_2,\, w \approx 4 \cdot z,\, y \approx 2 \cdot z \,\}$
and $\mathrm{M_e} = \{\, x_1 \approx y \times y,\, x_2 \approx w \times z \,\}$

Then $\sigma = \{\, w \mapsto 4 \cdot z,\, y \mapsto 2 \cdot z \,\}$ is a derivable substitution

Moreover, $(y \times y)\sigma{\downarrow} = ((2 \cdot z) \times (2 \cdot z)){\downarrow} = 4 \cdot (z \times z)$
$(w \times z)\sigma{\downarrow} = ((4 \cdot z) \times z){\downarrow} = 4 \cdot (z \times z)$

Thus, the solver can infer $x_1 \approx x_2$ from $\mathrm{M_b}$

Model-based refinement

What to do if no (more) simplifications apply to $\mathsf{M} = \mathsf{M}_{\mathrm{b}} \cup \mathsf{M}_{\mathrm{e}}$?

Model-based refinement

What to do if no (more) simplifications apply to $\mathsf{M} = \mathsf{M}_{\mathrm{b}} \cup \mathsf{M}_{\mathrm{e}}$?

Observation

M_{b} is a conservative abstraction of M in the basic language

Model-based refinement

What to do if no (more) simplifications apply to $\mathsf{M} = \mathsf{M}_{\mathrm{b}} \cup \mathsf{M}_{\mathrm{e}}$?

Observation

M_{b} is a conservative abstraction of M in the basic language

Abstraction refinement

1. If the basic solver $\mathsf{Solve}^{\mathrm{b}}_{T}$ finds M_{b} unsat then M is unsat

Model-based refinement

What to do if no (more) simplifications apply to $\mathsf{M} = \mathsf{M}_{\mathrm{b}} \cup \mathsf{M}_{\mathrm{e}}$?

Observation

M_{b} is a conservative abstraction of M in the basic language

Abstraction refinement

1. If the basic solver $\mathsf{Solve}_T^{\mathrm{b}}$ finds M_{b} unsat then M is unsat
2. If $\mathsf{Solve}_T^{\mathrm{b}}$ finds a T-model $\mathcal{I}$ s.t. $\mathcal{I} \models \mathsf{M}_{\mathrm{b}}$
   2.1 If $\mathcal{I} \models \mathsf{M}_{\mathrm{e}}$ then M is sat
   2.2 Otherwise, add to F a *refinement lemma*,
   a Σ^{b}-clause $\varphi\xi$ s.t. $\mathsf{M}_{\mathrm{e}} \models_T \varphi$ and $\mathcal{I} \not\models \varphi$

Model-based refinement

What to do if no (more) simplifications apply to $\mathsf{M} = \mathsf{M}_{\mathrm{b}} \cup \mathsf{M}_{\mathrm{e}}$?

Refinement Example

Let $\mathsf{M}_{\mathrm{b}} = \{\, z \not\approx 0 \,\}$ and $\mathsf{M}_{\mathrm{e}} = \{\, z \approx y \times y \,\}$

Let $\mathcal{I}$ be a model of IA satisfying M_{b} with $\mathcal{I}(z) = -1$

A refinement lemma for $(\mathsf{M}, \mathcal{I})$ is $z \geq 0$

Model-based refinement

What to do if no (more) simplifications apply to $\mathrm{M} = \mathrm{M}_\mathrm{b} \cup \mathrm{M}_\mathrm{e}$?

Refinement Example

Let $\mathrm{M}_\mathrm{b} = \{\, z \not\approx 0 \,\}$ and $\mathrm{M}_\mathrm{e} = \{\, z \approx y \times y \,\}$

Let $\mathcal{I}$ be a model of IA satisfying M_b with $\mathcal{I}(z) = -1$

A refinement lemma for $(\mathrm{M}, \mathcal{I})$ is $z \geq 0$

Note

$\lceil z \geq 0 \rceil = y \times y \geq 0$ is valid in IA

A Strategy for Extended Theory Solvers

$\text{Solve}^{\mathrm{e}}_{T}(\ \mathsf{M}_{\mathrm{b}} \cup \mathsf{M}_{\mathrm{e}}\ )$: Perform the following steps

1. **(Context-Dependent Simplification)**
2. **(Basic Solver)**
3. **(Model-Based Refinement)**

A Strategy for Extended Theory Solvers

$\mathsf{Solve}^{\mathrm{e}}_{\mathcal{T}}(\ \mathrm{M_b} \cup \mathrm{M_e}\ )$: Perform the following steps

1. **(Context-Dependent Simplification)**
   While there is a $\sigma = \{\,\bar{y} \mapsto \bar{s}\,\}$
   with $\bar{y}, \bar{s} \in \mathcal{T}(\mathrm{M_b})$ and $\mathrm{M_b} \models_{\mathcal{T}} \bar{y} \approx \bar{s}$
   do
   1.1 **(Ext-Reduce)**
   1.2 **(Ext-Equal)**
2. **(Basic Solver)**
3. **(Model-Based Refinement)**

A Strategy for Extended Theory Solvers

$\mathsf{Solve}^{\mathrm{e}}_{\mathcal{T}}(\ \mathrm{M_b} \cup \mathrm{M_e}\ )$: Perform the following steps

1. **(Context-Dependent Simplification)**
   While there is a $\sigma = \{\, \bar{y} \mapsto \bar{s} \,\}$
   with $\bar{y}, \bar{s} \in \mathcal{T}(\mathrm{M_b})$ and $\mathrm{M_b} \models_{\mathcal{T}} \bar{y} \approx \bar{s}$
   do
   1.1 **(Ext-Reduce)**
   If there is a $x \approx t \in \mathrm{M_e}$ s.t. $s = (t\sigma)\!\downarrow$ is basic and $x \approx s \in \mathcal{L}$
   return $\mathsf{Infer}(x \approx s)$
   1.2 **(Ext-Equal)**
2. **(Basic Solver)**
3. **(Model-Based Refinement)**

A Strategy for Extended Theory Solvers

$\mathrm{Solve}^{\mathrm{e}}_{T}(\ \mathrm{M_b} \cup \mathrm{M_e}\ )$: Perform the following steps

1. **(Context-Dependent Simplification)**
   While there is a $\sigma = \{\, \bar{y} \mapsto \bar{s} \,\}$
   with $\bar{y}, \bar{s} \in \mathcal{T}(\mathrm{M_b})$ and $\mathrm{M_b} \models_T \bar{y} \approx \bar{s}$
   do
   1.1 **(Ext-Reduce)**
   1.2 **(Ext-Equal)**
   If there are $x_1 \approx t_1, x_2 \approx t_2 \in \mathrm{M_e}$ s.t.
   $(t_1\sigma)\!\downarrow = (t_2\sigma)\!\downarrow$ and $x_1 \approx x_2 \in \mathcal{L}$
   return $\mathsf{Infer}(x_1 \approx x_2)$
2. **(Basic Solver)**
3. **(Model-Based Refinement)**

A Strategy for Extended Theory Solvers

$\text{Solve}^{\mathrm{e}}_{T}(\; \mathsf{M}_{\mathrm{b}} \cup \mathsf{M}_{\mathrm{e}} \;)$: Perform the following steps

1. **(Context-Dependent Simplification)**
2. **(Basic Solver)**
3. **(Model-Based Refinement)**

A Strategy for Extended Theory Solvers

$\mathsf{Solve}^{\mathrm{e}}_{\mathcal{T}}(\ \mathsf{M}_{\mathrm{b}} \cup \mathsf{M}_{\mathrm{e}}\ )$: Perform the following steps

1. **(Context-Dependent Simplification)**
2. **(Basic Solver)**
   Let $res = \mathsf{Solve}^{\mathrm{b}}_{\mathcal{T}}(\mathsf{M}_{\mathrm{b}})$
   Unless $res = \mathsf{Sat}(\mathcal{I})$ return res
3. **(Model-Based Refinement)**

A Strategy for Extended Theory Solvers

$\mathsf{Solve}^{\mathrm{e}}_{\mathcal{T}}(\ \mathsf{M}_{\mathrm{b}} \cup \mathsf{M}_{\mathrm{e}}\ )$: Perform the following steps

1. **(Context-Dependent Simplification)**
2. **(Basic Solver)**
   Let $\mathit{res} = \mathsf{Solve}^{\mathrm{b}}_{\mathcal{T}}(\mathsf{M}_{\mathrm{b}})$
3. **(Model-Based Refinement)**

A Strategy for Extended Theory Solvers

$\text{Solve}^{\mathrm{e}}_{\mathcal{T}}(\ \mathsf{M}_{\mathrm{b}} \cup \mathsf{M}_{\mathrm{e}}\ )$: Perform the following steps

1. **(Context-Dependent Simplification)**
2. **(Basic Solver)**
   Let $\mathit{res} = \text{Solve}^{\mathrm{b}}_{\mathcal{T}}(\mathsf{M}_{\mathrm{b}})$
3. **(Model-Based Refinement)**
   If $\mathit{res} = \text{Sat}(\mathcal{I})$
   3.1 **(Check)**
   return res if $\mathcal{I} \models \mathsf{M}_{\mathrm{e}}$
   3.2 **(Refine)**
   return $\text{Learn}(\lceil\varphi\rceil)$ if there is a ref. lemma φ s.t. $\mathcal{L}\mathit{it}(\varphi) \subseteq \mathcal{L}$
   3.3 **(Give up)**
   return Unknown

A Strategy for Extended Theory Solvers

$\text{Solve}^{\mathrm{e}}_{\mathcal{T}}(\ \mathsf{M}_{\mathrm{b}} \cup \mathsf{M}_{\mathrm{e}}\ )$: Perform the following steps

1. **(Context-Dependent Simplification)**
2. **(Basic Solver)**
3. **(Model-Based Refinement)**

Extending

a theory of string with concatenation and length

in the CVC4 solver

An extended theory of strings

Basic signature:

$\Sigma_S^b = (\,\mathsf{Int}, \mathsf{String}, \circ, |\_|, \epsilon, \mathsf{a}, \mathsf{ab}, \ldots)$

Extension signature:

$\Sigma_S^e = (\ \mathsf{substr}, \mathsf{contains}, \mathsf{indexof}, \mathsf{replace}, \ldots)$

Full signature:

$\Sigma_A = \Sigma_S^b \cup \Sigma_S^e$

CVC4 has an efficient and competitive theory solver for the basic theory

We recently worked an extending it to the full theory

Context-based simplification for strings

Simplification rules are highly non-trivial (2,000 LOC in C++)

Sample reductions:

$$
\begin{array}{rcl}
\mathsf{contains}(y \circ x \circ \mathsf{abc},\ x \circ \mathsf{a})\downarrow & = & \top \\
\mathsf{contains}(\mathsf{abcde},\ \mathsf{d} \circ x \circ \mathsf{a})\downarrow & = & \bot \\
\mathsf{contains}(\mathsf{a} \circ x,\ \mathsf{b} \circ x \circ \mathsf{a})\downarrow & = & \bot \\
\mathsf{indexof}(\mathsf{a} \circ x \circ \mathsf{b},\ \mathsf{b}, 0)\downarrow & = & 1 + \mathsf{indexof}(x, \mathsf{b}, 0) \\
\mathsf{indexof}(\mathsf{abc} \circ x,\ \mathsf{a} \circ x,\ 1)\downarrow & = & -1 \\
\mathsf{replace}(\mathsf{a} \circ x,\ \mathsf{b}, y)\downarrow & = & \mathsf{a} \circ \mathsf{replace}(x, \mathsf{b}, y) \\
\mathsf{replace}(x, y, y)\downarrow & = & x \\
\mathsf{substr}(x \circ \mathsf{abcd},\ 1 + |x|,\ 2)\downarrow & = & \mathsf{bc}
\end{array}
$$

Model-based refinement

When a S-model $\mathcal{I}$ satisfying M_{b} falsifies M,
the extended solver

1. identifies *relevant* falsified equations $z \approx f(\bar{x})$ in M
2. expands $z \approx f(\bar{x})$ lazily based on recursive axioms for extension functions

Built-in axioms for extension string operators

$$\llbracket\, x \approx \mathsf{substr}(y, n, m) \,\rrbracket \;\equiv$$
$$\mathsf{ite}(\; 0 \leq n < |y| \wedge 0 < m,$$
$$y \approx z_1 \circ x \circ z_2 \wedge |z_1| \approx n \wedge |z_2| \approx |y| \dot{-} m, \;\; x \approx \epsilon \;)$$

$$\llbracket\, x \approx \mathsf{contains}(y, z) \,\rrbracket \;\equiv$$
$$(x \approx \bot) \Leftrightarrow \bigwedge_{n=0}^{K} n \leq |y| - |z| \Rightarrow \neg\llbracket\, z \approx \mathsf{substr}(y, n, |z|) \,\rrbracket$$

$$\llbracket\, x \approx \mathsf{replace}(y, z, w) \,\rrbracket \;\equiv$$
$$\mathsf{ite}(\; z \not\approx \epsilon \wedge \llbracket\, \top \approx \mathsf{contains}(y, z) \,\rrbracket,$$
$$x \approx z_1 \circ w \circ z_2 \wedge y \approx z_1 \circ z \circ z_2 \wedge \llbracket\, |z_1| \approx \mathsf{indexof}(y, z, 0) \,\rrbracket,$$
$$x \approx y \;)$$

$$\llbracket\, x \approx \mathsf{indexof}(y, z, n) \,\rrbracket \;\equiv\; \ldots$$

Experimental evaluation

$25,386$ benchmarks generated by PyEx

PyEx is an SMT-based symbolic execution engine for Python

Benchmarks heavily involve string functions in the extended signature

Experimental evaluation

Compared our implementation in CVC4 against the string solvers in Z3-STR and Z3

Both use eager reductions to handle extended string functions

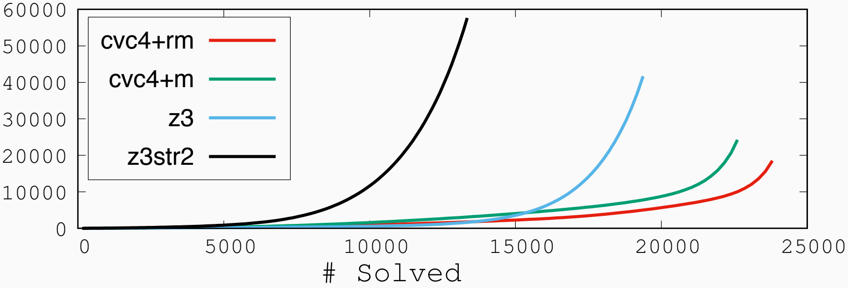
Tested two configurations of CVC4:

1. **cvc4+m**, which uses model-based refinement (**m**)
2. **cvc4+sm**, which also uses context-dependent simplification (**s**)

30*s* timeout for each benchmark

Experimental results

	PyEx-c (5557)		**PyEx-z3** (8399)		**PyEx-z32** (11430)		**Total** (25386)	
Solver	#	time	#	time	#	time	#	time
cvc4+sm	**5485**	52m	**11298**	2h33m	**7019**	1h43m	**23802**	5h8m
cvc4+m	5377	1h8m	10355	2h29m	6879	3h6m	22611	6h44m
z3	4695	2h44m	8415	5h18m	6258	3h30m	19368	11h33m
z3str2	3291	3h47m	5908	7h24m	4136	4h48m	13335	16h1m



Another application

Lightweight extension

of linear arithmetic theory solver to non-linear arithmetic

in CVC4

Integer and real arithmetic

Basic signature:

$\Sigma^{\mathrm{b}}_{\mathrm{A}} = (\,\mathsf{Int}, \mathsf{Real}, +, -, \cdot, 0, 1, ..., 1/2, 1/3, ..., \,)$

Extension signature:

$\Sigma^{\mathrm{e}}_{\mathrm{A}} = (\ \times\ )$

Full signature:

$\Sigma_{\mathrm{A}} = \Sigma^{\mathrm{b}}_{\mathrm{A}} \cup \Sigma^{\mathrm{e}}_{\mathrm{A}}$

CVC4 has an efficient and competitive theory solver for the basic theory based on several methods

We working an extending it to the full theory

Integer and real arithmetic

Context-dependent simplification linearizes non-linear terms when their variables become equivalent to constants

Integer and real arithmetic

Context-dependent simplification linearizes non-linear terms when their variables become equivalent to constants

All literals are first normalized to the form $p \sim 0$
where $\sim$ is a relational operator and
p a sum of monomials of the form $c \cdot x_1 \times \ldots \times x_n$

Integer and real arithmetic

All computed derivable substitutions σ are into constants

They are constructed from linear equalities in $\mathrm{M_b}$ by a Gaussian elimination process

Example

If $\mathrm{M_b} = \{ x + y \approx 4,\ x - y \approx -2,\ \dots \}$ then $\sigma = \{ x \mapsto 1,\ y \mapsto 3 \}$ will be computed

Model-based refinement

When a A-model $\mathcal{I}$ satisfying M_{b} falsifies M,
the extended solver

1. identifies ***relevant*** falsified equations $z \approx t_1 \times t_2$ in M
2. adds selected instances of (candidate) axiom templates for extension functions

Templates for model-based refinement in $\mathbb{A}$

Sign

$$t_1 \sim_1 0 \wedge t_2 \sim_2 0 \Rightarrow z \sim 0$$

Magnitude

$$|t_1| \sim_1 |s_1| \ \wedge \ |t_2| \sim_2 |s_2| \ \Rightarrow \ |z| \sim |s_1 \times s_2|$$

where $(s_1 \times s_2)\!\downarrow\, \in \mathcal{T}(\mathrm{M_e})$

Multiply

$$t_1 \sim_1 p \ \wedge \ t_2 \sim_2 0 \ \Rightarrow \ z \sim (t_2 \times p)$$

where $\deg(t_1) \geq \deg(p)$ and $(t_1 \sim_1 p)\!\downarrow\, \in \mathrm{M_b}$

t_1, t_2, s_1, s_2 are monomials, p is a polynomial

$\sim_1, \sim_2, \sim \in \{\approx, >, <, \leq, \geq\}$

Experimental evaluation

All benchmarks QF_NRA and QF_NIA from SMT-LIB 2

Tested two configurations of CVC4:

1. **cvc4+m**, which uses model-based refinement (**m**)
2. **cvc4+sm**, which also uses context-dependent simplification (**s**)

Experimental evaluation

All benchmarks QF_NRA and QF_NIA from SMT-LIB 2

Tested two configurations of CVC4:

1. **cvc4+m**, which uses model-based refinement (**m**)
2. **cvc4+sm**, which also uses context-dependent simplification (**s**)

QF_NRA

Compared against YICES2, Z3 and RASAT

RASAT has an incomplete interval-based solver

Z3 and YICES2 have complete solvers based on NLSAT/MCSAT

Experimental evaluation

All benchmarks QF_NRA and QF_NIA from SMT-LIB 2

Tested two configurations of CVC4:

1. **cvc4+m**, which uses model-based refinement (**m**)
2. **cvc4+sm**, which also uses context-dependent simplification (**s**)

QF_NIA

Compared against YICES2, Z3, and APROVE

APROVE relies on bit-blasting

Z3 relies on bit-blasting aided by linear and interval reasoning

YICES2 extends NLSAT with branch-and-bound

Experimental results

QF_NIA	**aprove**		**calypto**		**lranker**		**lctes**		**leipzig**		**mcm**		**uauto**		**ulranker**		**Total**	
	#	time	#	time	#	time	#	time	#	time	#	time	#	time	#	time	#	time
yices	**8706**	1761	**173**	83	98	102	0	0	92	30	4	32	**7**	0	**32**	11	**9112**	2021
z3	8253	7636	172	146	93	767	0	0	157	173	**16**	180	**7**	0	32	43	8730	8947
cvc4+m	8234	4799	164	43	**111**	52	**1**	0	69	589	0	0	6	0	32	84	8617	5569
cvc4+sm	8190	3723	170	61	108	57	**1**	0	68	375	3	107	7	1	32	86	8579	4413
AProVE	8028	3819	72	110	3	2	0	0	**157**	169	0	0	0	0	6	4	8266	4106

QF_NRA	**hong**		**hycomp**		**kissing**		**lranker**		**mtarski**		**uauto**		**zankl**		**Total**	
	#	time	#	time	#	time	#	time	#	time	#	time	#	time	#	time
z3	9	16	**2442**	3903	**27**	443	235	1165	**7707**	370	**60**	175	87	23	**10567**	6098
yices	7	59	2379	594	10	0	213	3110	7640	707	50	210	**91**	61	10390	4744
raSat	20	1	1933	409	12	32	0	0	6998	504	0	0	54	52	9017	999
cvc4+sm	**20**	0	2246	718	5	0	**623**	8375	5434	3711	11	31	33	36	8372	12874
cvc4+m	**20**	0	2236	491	6	0	603	6677	5440	3532	10	33	31	25	8346	10761

$60s$ timeout for each benchmark

Conclusions

- General modular approach for theory solver extensions
- Extended constraints processed with context-dependent simplification and model-based refinement techniques
- Provides new light-weight solutions for handling constraints in the theory of strings and in non-linear arithmetic
- Experimental data shows that the approach is
  - highly effective for strings
  - conferms some advantages over the state of the art in non-linear arithmetic

Future work

Use approach in part to develop further theory extensions

Extensions of interest include

- a stratified approach for floating-point constraints
- commonly used type conversion functions (e.g., bv_to_int, int_to_str)
- transcendental functions in real arithmetic
- catamorphisms on algebraic datatypes
- HOL constraints