Splitting on Demand in SAT Modulo Theories

Clark Barrett, Robert Nieuwenhuis, Albert Oliveras, Cesare Tinelli

New York Univ. Techn. Univ. Catalonia Univ. of Iowa

LPAR'06
November 17th, 2006, Phom Penh (Cambodia)

Overview of the talk

- Introduction to SMT
  - Eager approach
  - Lazy approach: Boolean engine DPLL(X) + T-solver
- Inside the T-solver
  - What does DPLL(X) need from T-solver?
  - Splitting on Demand
- Use of Splitting on Demand for Nelson-Oppen
- Conclusions

Introduction to SMT

- Some problems are more naturally expressed in other logics than propositional logic, e.g:
  - Software verification needs reasoning about equality, arithmetic, data structures, ...
- SMT consists of deciding the satisfiability of a (ground) FO formula with respect to a background theory T
- Example (Equality with Uninterpreted Functions – EUF):

$$g(a)=c \quad \wedge \quad (\ f(g(a)) \neq f(c) \ \vee \ g(a)=d\ ) \quad \wedge \quad c \neq d$$

- Wide range of applications:
  - Predicate abstraction
  - Model checking
  - Equivalence checking
  - Static analysis
  - Scheduling
  - ...

SMT - Eager approach vs lazy approach

EAGER APPROACH:

- Methodology: translate problem into equisatisfiable propositional formula and use off-the-shelf SAT solver [Bryant, Velev, Pnueli, Lahiri, Seshia, Strichman, ...]
- Why "eager"? Search uses all theory information from the beginning
- Tools: UCLID [Lahiri, Seshia and Bryant]

LAZY APPROACH:

- Methodology: integration of a SAT-solver with a theory solver
- Why "lazy"? Theory information used lazily when checking T-consistency of propositional models
- Tools: CVC-Lite, Yices, MathSAT, TSAT+, Barcelogic ...

SMT - Lazy approach example

Consider EUF and

$$\underbrace{g(a)=c}_{1} \quad \wedge \quad (\underbrace{f(g(a))\neq f(c)}_{\overline{2}} \vee \underbrace{g(a)=d}_{3}) \quad \wedge \quad \underbrace{c\neq d}_{\overline{4}}$$

- Send $\{1, \ \overline{2}\vee 3, \ \overline{4}\}$ to SAT solver

SMT - Lazy approach example

Consider EUF and

$$\underbrace{g(a)=c}_{1} \quad \wedge \quad (\ \underbrace{f(g(a))\neq f(c)}_{\overline{2}} \vee \underbrace{g(a)=d}_{3}\ ) \quad \wedge \quad \underbrace{c\neq d}_{\overline{4}}$$

- Send $\{1,\ \ \overline{2}\vee 3,\ \ \overline{4}\}$ to SAT solver
  - SAT solver returns model $[1,\ \ \overline{2},\ \ \overline{4}]$

SMT - Lazy approach example

Consider EUF and

$$\underbrace{g(a)=c}_{1} \quad\land\quad (\,\underbrace{f(g(a))\neq f(c)}_{\overline{2}} \lor \underbrace{g(a)=d}_{3}\,) \quad\land\quad \underbrace{c\neq d}_{\overline{4}}$$

- Send $\{1,\ \overline{2} \lor 3,\ \overline{4}\}$ to SAT solver
  - SAT solver returns model $[1,\ \overline{2},\ \overline{4}]$
  - Theory solver says T-inconsistent

SMT - Lazy approach example

Consider EUF and

$$\underbrace{g(a)=c}_{1} \quad \wedge \quad (\underbrace{f(g(a))\neq f(c)}_{\overline{2}} \vee \underbrace{g(a)=d}_{3}) \quad \wedge \quad \underbrace{c\neq d}_{\overline{4}}$$

- Send $\{1,\ \overline{2}\vee 3,\ \overline{4}\}$ to SAT solver
  - SAT solver returns model $[1,\ \overline{2},\ \overline{4}]$
  - Theory solver says T-inconsistent
- Send $\{1,\ \overline{2}\vee 3,\ \overline{4},\ \overline{1}\vee 2\vee 4\}$ to SAT solver

SMT - Lazy approach example

Consider EUF and

$$\underbrace{g(a)=c}_{1} \quad \wedge \quad (\underbrace{f(g(a))\neq f(c)}_{\overline{2}} \vee \underbrace{g(a)=d}_{3}) \quad \wedge \quad \underbrace{c\neq d}_{\overline{4}}$$

- Send $\{1,\ \overline{2}\vee 3,\ \overline{4}\}$ to SAT solver
  - SAT solver returns model $[1,\ \overline{2},\ \overline{4}]$
  - Theory solver says T-inconsistent
- Send $\{1,\ \overline{2}\vee 3,\ \overline{4},\ \overline{1}\vee 2\vee 4\}$ to SAT solver
  - SAT solver returns model $[1, 2, 3, \overline{4}]$

SMT - Lazy approach example

Consider EUF and

$$\underbrace{g(a)=c}_{1} \quad \wedge \quad (\underbrace{f(g(a))\neq f(c)}_{\overline{2}} \vee \underbrace{g(a)=d}_{3}) \quad \wedge \quad \underbrace{c\neq d}_{\overline{4}}$$

- Send $\{1,\ \overline{2}\vee 3,\ \overline{4}\}$ to SAT solver
  - SAT solver returns model $[1,\ \overline{2},\ \overline{4}]$
  - Theory solver says T-inconsistent
- Send $\{1,\ \overline{2}\vee 3,\ \overline{4},\ \overline{1}\vee 2\vee 4\}$ to SAT solver
  - SAT solver returns model $[1, 2, 3, \overline{4}]$
  - Theory solver says T-inconsistent

SMT - Lazy approach example

Consider EUF and

$$\underbrace{g(a)=c}_{1} \quad \wedge \quad (\underbrace{f(g(a))\neq f(c)}_{\overline{2}} \vee \underbrace{g(a)=d}_{3}) \quad \wedge \quad \underbrace{c\neq d}_{\overline{4}}$$

- Send $\{1,\ \overline{2}\vee 3,\ \overline{4}\}$ to SAT solver
  - SAT solver returns model $[1,\ \overline{2},\ \overline{4}]$
  - Theory solver says T-inconsistent
- Send $\{1,\ \overline{2}\vee 3,\ \overline{4},\ \overline{1}\vee 2\vee 4\}$ to SAT solver
  - SAT solver returns model $[1, 2, 3, \overline{4}]$
  - Theory solver says T-inconsistent
- Send $\{1,\ \overline{2}\vee 3,\ \overline{4},\ \overline{1}\vee 2\vee 4,\ \overline{1}\vee\overline{2}\vee\overline{3}\vee 4\}$ to SAT solver

SMT - Lazy approach example

Consider EUF and

$$\underbrace{g(a)=c}_{1} \quad \wedge \quad (\ \underbrace{f(g(a))\neq f(c)}_{\overline{2}} \ \vee \ \underbrace{g(a)=d}_{3}\ ) \quad \wedge \quad \underbrace{c\neq d}_{\overline{4}}$$

- Send $\{1,\ \ \overline{2}\vee 3,\ \ \overline{4}\}$ to SAT solver
  - SAT solver returns model $[1,\ \ \overline{2},\ \ \overline{4}]$
  - Theory solver says T-inconsistent
- Send $\{1,\ \ \overline{2}\vee 3,\ \ \overline{4},\ \ \overline{1}\vee 2\vee 4\}$ to SAT solver
  - SAT solver returns model $[1, 2, 3, \overline{4}]$
  - Theory solver says T-inconsistent
- Send $\{1,\ \ \overline{2}\vee 3,\ \ \overline{4},\ \ \overline{1}\vee 2\vee 4,\ \ \overline{1}\vee\overline{2}\vee\overline{3}\vee 4\}$ to SAT solver
  - SAT solver detects it UNSATISFIABLE

SMT - Lazy approach optimizations

Several optimizations for enhancing efficiency:

- Check T-consistency only of full propositional models

SMT - Lazy approach optimizations

Several optimizations for enhancing efficiency:

- ~~Check T-consistency only of full propositional models~~
- Check T-consistency of partial assignment while being built

SMT - Lazy approach optimizations

Several optimizations for enhancing efficiency:

- ~~Check T-consistency only of full propositional models~~
- Check T-consistency of partial assignment while being built

- Given a T-inconsistent assignment M, add $\neg M$ as a clause

SMT - Lazy approach optimizations

Several optimizations for enhancing efficiency:

- ~~Check T-consistency only of full propositional models~~
- Check T-consistency of partial assignment while being built

- ~~Given a T-inconsistent assignment M, add $\neg M$ as a clause~~
- Given a T-inconsistent assignment M, identify a T-inconsistent subset $M_0 \subseteq M$ and add $\neg M_0$ as a clause

SMT - Lazy approach optimizations

Several optimizations for enhancing efficiency:

- ~~Check T-consistency only of full propositional models~~
- Check T-consistency of partial assignment while being built

- ~~Given a T-inconsistent assignment M, add $\neg M$ as a clause~~
- Given a T-inconsistent assignment M, identify a T-inconsistent subset $M_0 \subseteq M$ and add $\neg M_0$ as a clause

- Upon a T-inconsistency, add clause and restart

SMT - Lazy approach optimizations

Several optimizations for enhancing efficiency:

- ~~Check T-consistency only of full propositional models~~
- Check T-consistency of partial assignment while being built

- ~~Given a T-inconsistent assignment M, add $\neg M$ as a clause~~
- Given a T-inconsistent assignment M, identify a T-inconsistent subset $M_0 \subseteq M$ and add $\neg M_0$ as a clause

- ~~Upon a T-inconsistency, add clause and restart~~
- Upon a T-inconsistency, use the conflicting clause $\neg M_0$ to backjump to some point where the assignment was still T-consistent, as in SAT-solvers.

Overview of the talk

- Introduction to SMT
  - Eager approach
  - Lazy approach: Boolean engine DPLL(X) + T-solver
- Inside the T-solver $\Leftarrow$
  - What does DPLL(X) need from T-solver?
  - Splitting on Demand
- Use of Splitting on Demand for Nelson-Oppen
- Conclusions

What does DPLL(X) need from T-Solver?

- T-consistency check of a set of literals M, with:
  - Explain of T-inconsistency: find (small) T-inconsistent subset of M [minimal wrt. size?, wrt. $\subseteq$?]
  - Incrementality: if l is added to M, check for $M\,l$ faster than reprocessing $M\,l$ from scratch.
- Theory propagation: find input T-consequences of M, with:
  - Explain T-Propagate of l: find (small) subset of M that T-entails l (needed in conflict analysis).
- Backtrack n: undo last n literals added

> PAPER FOCUSES only on T-consistency checks

A T-Solver for EUF

- Theory solvers can usually be described using inference rules
- The input **conjunction** of literals $\mathcal{C}$ is saturated wrt the rules
- $\mathcal{C}$ is unsat iff $\perp$ has been derived

A T-Solver for EUF

- Theory solvers can usually be described using inference rules
- The input **conjunction** of literals $\mathcal{C}$ is saturated wrt the rules
- $\mathcal{C}$ is unsat iff $\perp$ has been derived

A congruence closure algorithm (a solver for EUF) can be described with the following rules:

Reflexitivy:

$$\frac{}{t = t}$$

Symmetry:

$$\frac{u = t}{t = u}$$

Transitivity:

$$\frac{t = u \quad u = v}{t = v}$$

Monotonicity:

$$\frac{t_1 = u_1 \quad \dots \quad t_n = u_n}{f(t_1, \dots, t_n) = f(u_1, \dots, u_n)}$$

Contradiction:

$$\frac{t = u \quad t \neq u}{\perp}$$

A T-solver for difference logic (in $\mathbb{R}$)

Atoms are of the form $x \bowtie y + d$, being x and y variables, d a real constant and $\bowtie \in \{<, \leq\}$, or of the form $x = y + d$.

Transitivity:

$$\frac{x \leq z + c \quad z \bowtie y + d}{x \bowtie y + (c + d)} \qquad \frac{x < z + c \quad z \bowtie y + d}{x < y + (c + d)}$$

Equality treatment:

$$\frac{x \leq y + c \quad y \leq x - c}{x = y + c} \qquad \frac{x = y + c}{x \leq y + c, y \leq x - c}$$

Contradiction:

$$\frac{x < x + c}{\perp} \;\; (\text{if } c \leq 0) \qquad \frac{x = y + c \qquad x \neq y + c}{\perp}$$

A T-solver for difference logic (in $\mathbb{Z}$)

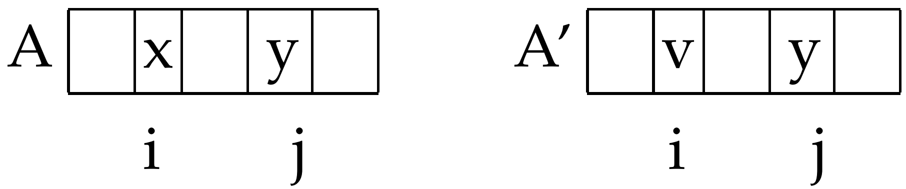
- Consider the unsatisfiable set of literals
$$\{1 \leq x - y,\ \ x - y \leq 2,\ \ x \neq y + 1,\ \ x \neq y + 2\}$$
- Saturation wrt the previous inference rules only adds
$$\{y \leq y + 1\}.$$
- To obtain a (refutationally) complete inference system:
  - Add splitting rule:
$$\frac{x \neq y + c}{x < y + c \qquad x > y + c}$$
  - Or add splitting rules of the form:
$$\frac{c \leq x - y \quad x - y \leq (c + k)}{x - y = c \qquad x - y = c + 1 \qquad \ldots \qquad x - y = c + k}$$
- This may give an exponential amount of work, but problem is NP-hard anyway.

Other theories requiring case-splitting

T-solvers requiring internal case-splitting are common:

- Theory of arrays:

$$\frac{read(write(A,i,v),j) = read(A,j)}{i \neq j \qquad\qquad i = j\,,\; read(A,j) = v}$$



- Fragments of set theory:

$$\frac{S_1 \neq S_2}{e \in S_1, e \notin S_2 \qquad e \notin S_1, e \in S_2}$$

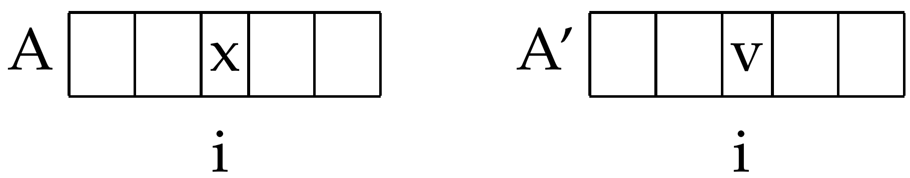
This type of solvers are much more difficult to implement than "deterministic" ones

Other theories requiring case-splitting

T-solvers requiring internal case-splitting are common:

- Theory of arrays:

$$\frac{read(write(A,i,v),j) = read(A,j)}{i \neq j \qquad i = j,\ read(A,j) = v}$$



- Fragments of set theory:

$$\frac{S_1 \neq S_2}{e \in S_1, e \notin S_2 \qquad e \notin S_1, e \in S_2}$$

This type of solvers are much more difficult to implement than "deterministic" ones

Our proposal: splitting on demand

INFORMALLY:

- IDEA: pass theory case-splits to the DPLL engine as clauses
- BENEFITS:
  - Split-backtrack infrastructure is not duplicated
  - Allow flexibility in T-reasoning (cheap computations first)

Our proposal: splitting on demand

INFORMALLY:

- IDEA: pass theory case-splits to the DPLL engine as clauses
- BENEFITS:
  - Split-backtrack infrastructure is not duplicated
  - Allow flexibility in T-reasoning (cheap computations first)

FORMALLY:

- Given initial state $\emptyset \parallel F$, consider $\mathcal{L}$ the **finite** set of all literals that might need case splitting.
- Modify T-Learn: also clauses with literals form $\mathcal{L}$ may be learned.
- $\mathcal{L}$ avoids termination problems (under certain conditions)
- T-solvers complete only when all atoms in $\mathcal{L}$ are decided

Example

Consider again Diff. Logic over $\mathbb{Z}$ and the formula:

$$\underbrace{x \leq y+1}_{1} \quad \wedge \quad (\underbrace{x < y}_{2} \vee \underbrace{x \neq y+1}_{\overline{3}}) \quad \wedge \quad \underbrace{x \neq y}_{\overline{4}}$$

$$\emptyset \parallel \quad 1, \;\; 2 \vee \overline{3}, \;\; \overline{4} \qquad\qquad \Rightarrow \qquad (\text{UnitPropagate x } 2)$$

Example

Consider again Diff. Logic over $\mathbb{Z}$ and the formula:

$$\underbrace{x \le y+1}_{1} \;\wedge\; (\underbrace{x < y}_{2} \vee \underbrace{x \ne y+1}_{\overline{3}}) \;\wedge\; \underbrace{x \ne y}_{\overline{4}}$$

$$\emptyset \;\|\; 1,\; 2 \vee \overline{3},\; \overline{4} \quad \Rightarrow \quad (\text{UnitPropagate x } 2)$$

$$1\,\overline{4} \;\|\; 1,\; 2 \vee \overline{3},\; \overline{4} \quad \Rightarrow \quad (T\text{-Learn with } 5 \equiv x > y)$$

Example

Consider again Diff. Logic over $\mathbb{Z}$ and the formula:

$$\underbrace{x \leq y+1}_{1} \;\wedge\; (\underbrace{x < y}_{2} \vee \underbrace{x \neq y+1}_{\overline{3}}) \;\wedge\; \underbrace{x \neq y}_{\overline{4}}$$

$$\begin{array}{rcllll}
\emptyset & \| & 1,\; 2 \vee \overline{3},\; \overline{4} & & \Rightarrow & (\text{UnitPropagate x } 2) \\
1\,\overline{4} & \| & 1,\; 2 \vee \overline{3},\; \overline{4} & & \Rightarrow & (T\text{-Learn with } 5 \equiv x > y) \\
1\,\overline{4} & \| & 1,\; 2 \vee \overline{3},\; \overline{4}, & 4 \vee 2 \vee 5 & \Rightarrow & (\text{Decide})
\end{array}$$

Example

Consider again Diff. Logic over $\mathbb{Z}$ and the formula:

$$\underbrace{x \leq y+1}_{1} \;\wedge\; (\underbrace{x < y}_{2} \vee \underbrace{x \neq y+1}_{\overline{3}}) \;\wedge\; \underbrace{x \neq y}_{\overline{4}}$$

$$\begin{array}{rclcl}
\emptyset & \| & 1,\; 2 \vee \overline{3},\; \overline{4} & \Rightarrow & (\text{UnitPropagate x } 2) \\
1\,\overline{4} & \| & 1,\; 2 \vee \overline{3},\; \overline{4} & \Rightarrow & (T\text{-Learn with } 5 \equiv x > y) \\
1\,\overline{4} & \| & 1,\; 2 \vee \overline{3},\; \overline{4},\; 4 \vee 2 \vee 5 & \Rightarrow & (\text{Decide}) \\
1\,\overline{4}\,5 & \| & 1,\; 2 \vee \overline{3},\; \overline{4},\; 4 \vee 2 \vee 5 & \Rightarrow & (\text{T-Propagate x } 2)
\end{array}$$

Example

Consider again Diff. Logic over $\mathbb{Z}$ and the formula:

$$\underbrace{x \leq y+1}_{1} \quad \wedge \quad (\underbrace{x < y}_{2} \vee \underbrace{x \neq y+1}_{\overline{3}}) \quad \wedge \quad \underbrace{x \neq y}_{\overline{4}}$$

$$\begin{array}{rclcl}
\emptyset & \| & 1,\ \ 2 \vee \overline{3},\ \ \overline{4} & \Rightarrow & (\text{UnitPropagate x 2}) \\
1\ \overline{4} & \| & 1,\ \ 2 \vee \overline{3},\ \ \overline{4} & \Rightarrow & (T\text{-Learn with } 5 \equiv x > y) \\
1\ \overline{4} & \| & 1,\ \ 2 \vee \overline{3},\ \ \overline{4},\ \ 4 \vee 2 \vee 5 & \Rightarrow & (\text{Decide}) \\
1\ \overline{4}\ 5 & \| & 1,\ \ 2 \vee \overline{3},\ \ \overline{4},\ \ 4 \vee 2 \vee 5 & \Rightarrow & (\text{T-Propagate x 2}) \\
1\ \overline{4}\ 5\ \overline{2}\ 3 & \| & 1,\ \ 2 \vee \overline{3},\ \ \overline{4},\ \ 4 \vee 2 \vee 5 & \Rightarrow & (\text{Backjump})
\end{array}$$

Example

Consider again Diff. Logic over $\mathbb{Z}$ and the formula:

$$\underbrace{x \leq y+1}_{1} \;\wedge\; (\underbrace{x < y}_{2} \vee \underbrace{x \neq y+1}_{\overline{3}}) \;\wedge\; \underbrace{x \neq y}_{\overline{4}}$$

$$\begin{array}{rcllll}
\emptyset & \| & 1,\ 2 \vee \overline{3},\ \overline{4} & & \Rightarrow & (\text{UnitPropagate x } 2) \\
1\ \overline{4} & \| & 1,\ 2 \vee \overline{3},\ \overline{4} & & \Rightarrow & (T\text{-Learn with } 5 \equiv x > y) \\
1\ \overline{4} & \| & 1,\ 2 \vee \overline{3},\ \overline{4}, & 4 \vee 2 \vee 5 & \Rightarrow & (\text{Decide}) \\
1\ \overline{4}\ 5 & \| & 1,\ 2 \vee \overline{3},\ \overline{4}, & 4 \vee 2 \vee 5 & \Rightarrow & (\text{T-Propagate x } 2) \\
1\ \overline{4}\ 5\ \overline{2}\ 3 & \| & 1,\ 2 \vee \overline{3},\ \overline{4}, & 4 \vee 2 \vee 5 & \Rightarrow & (\text{Backjump}) \\
1\ \overline{4}\ \overline{5} & \| & 1,\ 2 \vee \overline{3},\ \overline{4}, & 4 \vee 2 \vee 5 & \Rightarrow & (\text{UnitPropagate})
\end{array}$$

Example

Consider again Diff. Logic over $\mathbb{Z}$ and the formula:

$$\underbrace{x \leq y+1}_{1} \;\wedge\; (\underbrace{x < y}_{2} \vee \underbrace{x \neq y+1}_{\overline{3}}) \;\wedge\; \underbrace{x \neq y}_{\overline{4}}$$

$\emptyset$	$\parallel$	$1,\ 2 \vee \overline{3},\ \overline{4}$	$\Rightarrow$	(UnitPropagate x 2)
$1\ \overline{4}$	$\parallel$	$1,\ 2 \vee \overline{3},\ \overline{4}$	$\Rightarrow$	(T-Learn with $5 \equiv x > y$)
$1\ \overline{4}$	$\parallel$	$1,\ 2 \vee \overline{3},\ \overline{4},\ 4 \vee 2 \vee 5$	$\Rightarrow$	(Decide)
$1\ \overline{4}\ 5$	$\parallel$	$1,\ 2 \vee \overline{3},\ \overline{4},\ 4 \vee 2 \vee 5$	$\Rightarrow$	(T-Propagate x 2)
$1\ \overline{4}\ 5\ \overline{2}\ 3$	$\parallel$	$1,\ 2 \vee \overline{3},\ \overline{4},\ 4 \vee 2 \vee 5$	$\Rightarrow$	(Backjump)
$1\ \overline{4}\ \overline{5}$	$\parallel$	$1,\ 2 \vee \overline{3},\ \overline{4},\ 4 \vee 2 \vee 5$	$\Rightarrow$	(UnitPropagate)
$1\ \overline{4}\ \overline{5}\ 2$	$\parallel$	$1,\ 2 \vee \overline{3},\ \overline{4},\ 4 \vee 2 \vee 5$		(Model found)

Overview of the talk

- Introduction to SMT
  - Eager approach
  - Lazy approach: Boolean engine DPLL(X) + T-solver
- Inside the T-solver
  - What does DPLL(X) need from T-solver?
  - Splitting on Demand
- Use of Splitting on Demand for Nelson-Oppen $\Leftarrow$
- Conclusions

Nelson-Oppen: combination of theories

- SMT problems usually involve more than one theory:

$$a = b+2 \;\wedge\; A = write(B, a+1, 4) \;\wedge\; (\, read(A, b+3) = 2 \vee f(a-1) \neq f(b+1)\,)$$

- Combination problem:

  INPUT:
  - Two theories T_1 and T_2.
  - A T_1-solver and a T_2-solver

  OUTPUT:
  - A $(T_1 \cup T_2)-$solver

- Nelson-Oppen provides a combination procedure if:
  - Theories are signature disjoint
  - Theories are stably-infinite

Nelson-Oppen: example

$$\Gamma = \{ f(f(x) - f(y)) \neq f(z), \quad x \leq y, \quad y + z \leq x, \quad z \geq 0 \}$$

1. Purify literals: introduce new variables
   $w_1 = f(x), w_2 = f(y), w_3 = w_1 - w_2$
2. Now we get
   $\Gamma_{\mathbb{R}} = \{x \leq y, y + z \leq x, z \geq 0, w_3 = w_1 - w_2\}$ and
   $\Gamma_{\mathbb{E}} = \{f(w_3) \neq f(z), w_1 = f(x), w_2 = f(y)\}$
   with shared variables $\{x, y, z, w_1, w_2, w_3\}$.
3. N-O: Γ SAT in the combined theory iff exists arrangement $\mathcal{A}$ (for each pair of shared variables, say whether they are equal or distinct) such that $\Gamma_{\mathbb{R}} \wedge \mathcal{A}$ is $T_{\mathbb{R}}$-SAT and $\Gamma_{\mathbb{E}} \wedge \mathcal{A}$ is $T_{\mathbb{E}}$-SAT.

Ideal implementation: T-solvers exchange entailed equations until fix point or unsatisfiability is detected by a single T-solver.

Nelson-Oppen: example(2)

$\Gamma_{\mathbb{R}} = \{x \leq y, \quad y + z \leq x, \quad z \geq 0, \quad w_3 = w_1 - w_2\}$

$\Gamma_{\mathbb{E}} = \{f(w_3) \neq f(z), \quad w_1 = f(x), \quad w_2 = f(y)\}$

with shared variables $\{x, y, z, w_1, w_2, w_3\}$.

Arrangement $\mathcal{A}$ (init. empty) is seen by both solvers:

- $T_{\mathbb{R}}$-solver detects $x = y$ is entailed (and added to $\mathcal{A}$)

Nelson-Oppen: example(2)

$\Gamma_{\mathbb{R}} = \{x \leq y, \quad y + z \leq x, \quad z \geq 0, \quad w_3 = w_1 - w_2\}$

$\Gamma_{\mathbb{E}} = \{f(w_3) \neq f(z), \quad w_1 = f(x), \quad w_2 = f(y)\}$

with shared variables $\{x, y, z, w_1, w_2, w_3\}$.

Arrangement $\mathcal{A}$ (init. empty) is seen by both solvers:

- $T_{\mathbb{R}}$-solver detects $x = y$ is entailed (and added to $\mathcal{A}$)
- $T_{\mathbb{E}}$-solver detects $w_1 = w_2$ is entailed (and added to $\mathcal{A}$)

Nelson-Oppen: example(2)

$\Gamma_{\mathbb{R}} = \{x \leq y, \quad y + z \leq x, \quad z \geq 0, \quad w_3 = w_1 - w_2\}$

$\Gamma_{\mathbb{E}} = \{f(w_3) \neq f(z), \quad w_1 = f(x), \quad w_2 = f(y)\}$

with shared variables $\{x, y, z, w_1, w_2, w_3\}$.

Arrangement $\mathcal{A}$ (init. empty) is seen by both solvers:

- $T_{\mathbb{R}}$-solver detects $x = y$ is entailed (and added to $\mathcal{A}$)
- $T_{\mathbb{E}}$-solver detects $w_1 = w_2$ is entailed (and added to $\mathcal{A}$)
- $T_{\mathbb{R}}$-solver detects $z = w_3$ is entailed (and added to $\mathcal{A}$)

Nelson-Oppen: example(2)

$\Gamma_{\mathbb{R}} = \{x \leq y, \quad y + z \leq x, \quad z \geq 0, \quad w_3 = w_1 - w_2\}$
$\Gamma_{\mathbb{E}} = \{f(w_3) \neq f(z), \quad w_1 = f(x), \quad w_2 = f(y)\}$
with shared variables $\{x, y, z, w_1, w_2, w_3\}$.
Arrangement $\mathcal{A}$ (init. empty) is seen by both solvers:

- $T_{\mathbb{R}}$-solver detects $x = y$ is entailed (and added to $\mathcal{A}$)
- $T_{\mathbb{E}}$-solver detects $w_1 = w_2$ is entailed (and added to $\mathcal{A}$)
- $T_{\mathbb{R}}$-solver detects $z = w_3$ is entailed (and added to $\mathcal{A}$)
- $T_{\mathbb{E}}$-solver detects $\Gamma_{\mathbb{E}} \wedge \mathcal{A}$ is $T_{\mathbb{E}}$-unsat

Nelson-Oppen: example(2)

$\Gamma_{\mathbb{R}} = \{x \leq y, \quad y + z \leq x, \quad z \geq 0, \quad w_3 = w_1 - w_2\}$
$\Gamma_{\mathbb{E}} = \{f(w_3) \neq f(z), \quad w_1 = f(x), \quad w_2 = f(y)\}$
with shared variables $\{x, y, z, w_1, w_2, w_3\}$.
Arrangement $\mathcal{A}$ (init. empty) is seen by both solvers:

- $T_{\mathbb{R}}$-solver detects $x = y$ is entailed (and added to $\mathcal{A}$)
- $T_{\mathbb{E}}$-solver detects $w_1 = w_2$ is entailed (and added to $\mathcal{A}$)
- $T_{\mathbb{R}}$-solver detects $z = w_3$ is entailed (and added to $\mathcal{A}$)
- $T_{\mathbb{E}}$-solver detects $\Gamma_{\mathbb{E}} \wedge \mathcal{A}$ is $T_{\mathbb{E}}$-unsat

But exchanging entailed equalities does not always suffice:

$$\Gamma_{\mathbb{Z}} = \{1 \leq x - y, \quad x - y \leq 2, \quad w_1 = y + 1, \quad w_2 = y + 2\}$$

$$\Gamma_{\mathbb{E}} = \{f(x) \neq f(w_1), \quad f(x) \neq f(w_2)\}$$

is UNSAT, but no equation is entailed.

Nelson-Oppen with non-convex theories

Why didn't it work with

$\Gamma_{\mathbb{Z}} = \{1 \leq x - y, \quad x - y \leq 2, \quad w_1 = y + 1, \quad w_2 = y + 2\}$

$\Gamma_{\mathbb{E}} = \{f(x) \neq f(w_1), \quad f(x) \neq f(w_2)\}$?

Nelson-Oppen with non-convex theories

Why didn't it work with

$\Gamma_{\mathbb{Z}} = \{1 \leq x - y, \quad x - y \leq 2, \quad w_1 = y + 1, \quad w_2 = y + 2\}$

$\Gamma_{\mathbb{E}} = \{f(x) \neq f(w_1), \quad f(x) \neq f(w_2)\}$?

- $\Gamma_{\mathbb{Z}}$ does not entail any equation between shared variables
- But $\Gamma_{\mathbb{Z}} \models_T \; x = w_1 \vee x = w_2$ (non-convex theory)
- For non-convex theories, DISJUNCTIONS of equalities should be communicated. Possibilities:
  - Send clauses from solver to solver
  - Force DPLL(X) to split on equalities between shared variables [DTC]
  - Send clauses from solvers to DPLL(X) only as necessary [DTC,Splitting on Demand]

Overview of the talk

- Introduction to SMT
  - Eager approach
  - Lazy approach: Boolean engine DPLL(X) + T-solver
- Inside the T-solver
  - What does DPLL(X) need from T-solver?
  - Splitting on Demand
- Use of Splitting on Demand for Nelson-Oppen
- Conclusions $\Leftarrow$

Conclusions

- Expensive theories easily dealt with the appropriate infrastructure
- This infrastructure allows greater flexibility
- Nelson-Oppen easily accommodated