Rewrite-based satisfiability procedures for Recursive Data Structures

Maria Paola Bonacina and Mnacho Echenim

August 21st 2006

Outline

Outline

- Preliminaries
- Rewrite-based $\mathcal{T}$-satisfiability procedures
- Recursive Data Structures
- Combination and complexity
- Discussion

Background

- First-order logic with equality
- Theories of data structures (or their combination)

Background

- First-order logic with equality
- Theories of data structures (or their combination)
  - Theory of arrays with or without extensionality

Background

- First-order logic with equality
- Theories of data structures (or their combination)
  - Theory of arrays with or without extensionality

$$\forall x, z, v.\ \mathrm{select}(\mathrm{store}(x, z, v), z) \simeq v,$$
$$\forall x, z, v, w.\ z \not\simeq w \Rightarrow \mathrm{select}(\mathrm{store}(x, z, v), w) \simeq \mathrm{select}(x, w),$$
$$\forall x, y.\ (\forall z.\ \mathrm{select}(x, z) \simeq \mathrm{select}(y, z) \Rightarrow x \simeq y). \quad \textbf{(ext)}$$

Background

- First-order logic with equality
- Theories of data structures (or their combination)
  - Theory of arrays with or without extensionality
    $\forall x, z, v.\ \mathrm{select}(\mathrm{store}(x, z, v), z) \simeq v,$
    $\forall x, z, v, w.\ z \not\simeq w \Rightarrow \mathrm{select}(\mathrm{store}(x, z, v), w) \simeq \mathrm{select}(x, w),$
    $\forall x, y.\ (\forall z.\ \mathrm{select}(x, z) \simeq \mathrm{select}(y, z) \Rightarrow x \simeq y).$ **(ext)**
  - Theory of nonempty lists

Background

- First-order logic with equality
- Theories of data structures (or their combination)
  - Theory of arrays with or without extensionality

    $\forall x, z, v.\ \mathrm{select}(\mathrm{store}(x, z, v), z) \simeq v,$

    $\forall x, z, v, w.\ z \not\simeq w \Rightarrow \mathrm{select}(\mathrm{store}(x, z, v), w) \simeq \mathrm{select}(x, w),$

    $\forall x, y.\ (\forall z.\ \mathrm{select}(x, z) \simeq \mathrm{select}(y, z) \Rightarrow x \simeq y).$ **(ext)**
  - Theory of nonempty lists

    $\forall x.\ \mathrm{cons}(\mathrm{car}(x), \mathrm{cdr}(x)) \simeq x,$

    $\forall x, y.\ \mathrm{car}(\mathrm{cons}(x, y)) \simeq x,$

    $\forall x, y.\ \mathrm{cdr}(\mathrm{cons}(x, y)) \simeq y.$

$\mathcal{T}$-satisfiability problems

Definition

Given the presentation $\mathcal{T}$ of a theory and a set S of ground literals, is $\mathcal{T} \cup S$ satisfiable ?

$\mathcal{T}$-satisfiability problems

Definition

Given the presentation $\mathcal{T}$ of a theory and a set S of ground literals, is $\mathcal{T} \cup S$ satisfiable ?

A standard approach : *"little" engines of proofs.*

- Principle
  - Theory is built into a dedicated inference engine.
  - Input of the inference engine : the set of ground literals.

$\mathcal{T}$-satisfiability problems

Definition

Given the presentation $\mathcal{T}$ of a theory and a set S of ground literals, is $\mathcal{T} \cup S$ satisfiable ?

A standard approach : *"little" engines of proofs.*

- Principle
  - Theory is built into a dedicated inference engine.
  - Input of the inference engine : the set of ground literals.
- Issues
  - Prove correctness, completeness.
  - How can these engines be efficiently combined ?

A new approach : *"big" engines of proofs*.

Idea : use *generic* theorem provers to solve $\mathcal{T}$-satisfiability problems.

A new approach : *"big" engines of proofs*.

Idea : use *generic* theorem provers to solve $\mathcal{T}$-satisfiability problems.

- Give $\mathcal{T} \cup S$ as an input.

A new approach : *"big" engines of proofs*.

Idea : use *generic* theorem provers to solve $\mathcal{T}$-satisfiability problems.

- Give $\mathcal{T} \cup S$ as an input.
- Advantages :
  - Correctness is guaranteed.
  - Combination is conceptually easy.

A new approach : *"big" engines of proofs.*

Idea : use *generic* theorem provers to solve $\mathcal{T}$-satisfiability problems.

- Give $\mathcal{T} \cup S$ as an input.
- Advantages :
  - Correctness is guaranteed.
  - Combination is conceptually easy.
- Issue : proof of *termination*.

A new approach : *"big" engines of proofs.*

Idea : use *generic* theorem provers to solve $\mathcal{T}$-satisfiability problems.

- Give $\mathcal{T} \cup S$ as an input.
- Advantages :
  - Correctness is guaranteed.
  - Combination is conceptually easy.
- Issue : proof of *termination*.

Current line of work : rewrite-based inference systems.

Outline

- Preliminaries
- Rewrite-based $\mathcal{T}$-satisfiability procedures
- Recursive Data Structures
- Combination and complexity
- Discussion

The Superposition Calculus : $\mathcal{SP}$.

Consists of :

- *Expansion rules :*
  - Superposition/Paramodulation, Reflection, Equational Factoring.

The Superposition Calculus : $\mathcal{SP}$.

Consists of :

- *Expansion rules :*
  - Superposition/Paramodulation, Reflection, Equational Factoring.
- *Contraction rules :*
  - Subsumption, Simplification, Deletion.

The Superposition Calculus : $\mathcal{SP}$.

Consists of :

- *Expansion rules :*
  - Superposition/Paramodulation, Reflection, Equational Factoring.
- *Contraction rules :*
  - Subsumption, Simplification, Deletion.

Associated with a *complete simplification ordering* $\succ$ on terms, literals and clauses : $\mathcal{SP}_{\succ}$

The Superposition Calculus : $\mathcal{SP}$.

Consists of :

- *Expansion rules :*
  - Superposition/Paramodulation, Reflection, Equational Factoring.
- *Contraction rules :*
  - Subsumption, Simplification, Deletion.

Associated with a *complete simplification ordering* $\succ$ on terms, literals and clauses : $\mathcal{SP}_\succ$

Definition

Derivation : $S_0 \vdash_{\mathcal{SP}_\succ} S_1 \vdash_{\mathcal{SP}_\succ} \ldots \vdash_{\mathcal{SP}_\succ} S_i \ldots$

Set of *persistent* clauses : $S_\infty = \bigcup_{j \geq 0} \bigcap_{i > j} S_i$.

A three-step methodology

- *$\mathcal{T}$-reduction* : obtain equisatisfiable problem by removing/transforming literals.

  Example : arrays with extensionality $\rightarrow$ arrays without extensionality.

A three-step methodology

- *$\mathcal{T}$-reduction* : obtain equisatisfiable problem by removing/transforming literals.

  Example : arrays with extensionality $\rightarrow$ arrays without extensionality.

- *Flattening* : flatten all ground literals to obtain a *flat* $\mathcal{T}$-satisfiability problem.

  Example : replace $f(f(a)) \simeq b$ by $f(a) \simeq a_1$ and $f(a_1) \simeq b$.

A three-step methodology

- *$\mathcal{T}$-reduction* : obtain equisatisfiable problem by removing/transforming literals.

  Example : arrays with extensionality $\longrightarrow$ arrays without extensionality.

- *Flattening* : flatten all ground literals to obtain a *flat* $\mathcal{T}$-satisfiability problem.

  Example : replace $f(f(a)) \simeq b$ by $f(a) \simeq a_1$ and $f(a_1) \simeq b$.

- *Proof of termination* : prove that $\mathcal{SP}_{\succ}$ with a fair search plan terminates.

A three-step methodology

- *$\mathcal{T}$-reduction* : obtain equisatisfiable problem by removing/transforming literals.

  Example : arrays with extensionality $\rightarrow$ arrays without extensionality.

- *Flattening* : flatten all ground literals to obtain a *flat* $\mathcal{T}$-satisfiability problem.

  Example : replace $f(f(a)) \simeq b$ by $f(a) \simeq a_1$ and $f(a_1) \simeq b$.

- *Proof of termination* : prove that $\mathcal{SP}_{\succ}$ with a fair search plan terminates.

The first two steps can be performed automatically.

Termination/Complexity

What kind of clauses can S_∞ contain ?

Termination/Complexity

What kind of clauses can S_∞ contain ? Prove that these clauses belong to a finite number of finite categories.

Termination/Complexity

What kind of clauses can S_∞ contain ? Prove that these clauses belong to a finite number of finite categories.

- Termination :
  - S_∞ is finite.
  - $\mathcal{SP}_\succ$ with a fair search plan terminates.

Termination/Complexity

What kind of clauses can S_∞ contain ? Prove that these clauses belong to a finite number of finite categories.

- Termination :
  - S_∞ is finite.
  - $\mathcal{SP}_\succ$ with a fair search plan terminates.
- Complexity :
  - Determine the number of clauses in each category.
  - Upper-bound on the size of S_∞.

Termination/Complexity

What kind of clauses can S_∞ contain ? Prove that these clauses belong to a finite number of finite categories.

- Termination :
  - S_∞ is finite.
  - $\mathcal{SP}_\succ$ with a fair search plan terminates.
- Complexity :
  - Determine the number of clauses in each category.
  - Upper-bound on the size of S_∞.

Some upper-bounds on the sizes of S_∞ :

- Theory of non-empty lists : $O(n^2)$
- Theory of arrays : $O(2^{n^2})$

Theories covered by this approach

- Equality
- Lists (*à la* Shostak, *à la* Nelson and Oppen, possibly empty)
- Arrays, records and finite sets, with or without extensionality

Theories covered by this approach

- Equality
- Lists (*à la* Shostak, *à la* Nelson and Oppen, possibly empty)
- Arrays, records and finite sets, with or without extensionality

What about acyclic lists ?

Theories covered by this approach

- Equality
- Lists (*à la* Shostak, *à la* Nelson and Oppen, possibly empty)
- Arrays, records and finite sets, with or without extensionality

What about acyclic lists ?

- An infinite presentation : $\mathrm{cdr}(x) \not\simeq x$, $\mathrm{cdr}(\mathrm{cdr}(x)) \not\simeq x$, etc...
- How do we ensure termination ?

Outline

- Preliminaries
- Rewrite-based $\mathcal{T}$-satisfiability procedures
- Recursive Data Structures
- Combination and complexity
- Discussion

Definition

A *constructor* of arity k, and k *selectors* of arity 1, where $k \geq 1$.

Definition

A *constructor* of arity k, and k *selectors* of arity 1, where $k \geq 1$.

$k = 1$: *integer offsets*

$$\mathrm{p}(\mathrm{s}(x)) \simeq x$$

$$\mathrm{s}(\mathrm{p}(x)) \simeq x$$

$$\{\mathrm{s}^i(x) \not\simeq x \mid i > 0\}$$

Definition

A *constructor* of arity k, and k *selectors* of arity 1, where $k \geq 1$.

$k = 2$: *acyclic lists*

$$\mathrm{cons}(\mathrm{car}(x), \mathrm{cdr}(x)) \simeq x$$

$$\{\mathrm{car}(\mathrm{cons}(x, y)) \simeq x,\ \mathrm{cdr}(\mathrm{cons}(x, y)) \simeq y\}$$

$$\{\mathrm{car}(x) \not\simeq x,\ \mathrm{cdr}(x) \not\simeq x,\ \mathrm{car}(\mathrm{cdr}(x)) \not\simeq x \ldots\}$$

Definition

A *constructor* of arity k, and k *selectors* of arity 1, where $k \geq 1$.

For any $k \geq 1$:

$$\mathrm{cons}(\mathrm{sel}_1(x), \ldots, \mathrm{sel}_k(x)) \simeq x$$

$$\{\mathrm{sel}_i(\mathrm{cons}(x_1, \ldots, x_k)) \simeq x_i \mid 1 \leq i \leq k\}$$

$$\{t[x] \not\simeq x\}$$

Definition

A *constructor* of arity k, and k *selectors* of arity 1, where $k \geq 1$.

For any $k \geq 1$:

$$\mathrm{cons}(\mathrm{sel}_1(x), \ldots, \mathrm{sel}_k(x)) \simeq x$$

$$\{\mathrm{sel}_i(\mathrm{cons}(x_1, \ldots, x_k)) \simeq x_i \mid 1 \leq i \leq k\}$$

$\mathcal{R}$

$$\{t[x] \not\simeq x\}\ \mathrm{Ac}$$

Definition

A *constructor* of arity k, and k *selectors* of arity 1, where $k \geq 1$.

For any $k \geq 1$:

$$\left.\begin{array}{l} \mathrm{cons}(\mathrm{sel}_1(x), \ldots, \mathrm{sel}_k(x)) \simeq x \\ \{\mathrm{sel}_i(\mathrm{cons}(x_1, \ldots, x_k)) \simeq x_i \mid 1 \leq i \leq k\} \end{array}\right] \mathcal{R}$$

$$\{t[x] \not\simeq x\} \ \mathrm{Ac}$$

There can be no termination result on $S \cup \mathrm{Ac} \cup \mathcal{R}$.

Definition

A *constructor* of arity k, and k *selectors* of arity 1, where $k \geq 1$.

For any $k \geq 1$:

$$\left.\begin{array}{l} \mathrm{cons}(\mathrm{sel}_1(x), \ldots, \mathrm{sel}_k(x)) \simeq x \\ \{\mathrm{sel}_i(\mathrm{cons}(x_1, \ldots, x_k)) \simeq x_i \mid 1 \leq i \leq k\} \end{array}\right] \mathcal{R}$$

$\{t[x] \not\simeq x\}$ Ac

There can be no termination result on $S \cup \mathrm{Ac} \cup \mathcal{R}$.

Does there exist a finite set that is equisatisfiable to $S \cup \mathrm{Ac} \cup \mathcal{R}$?

Equisatisfiable sets

Given $S \cup \mathrm{Ac} \cup \mathcal{R}$, apply two steps :

- "Discard" the axioms in $\mathcal{R}$,
- Use $\mathrm{Ac}(n) = \{t[x] \not\simeq x \mid \mathrm{depth}(t) \leq n\}$ instead of Ac.

Equisatisfiable sets

Given $S \cup \mathrm{Ac} \cup \mathcal{R}$, apply two steps :

- "Discard" the axioms in $\mathcal{R}$,
- Use $\mathrm{Ac}(n) = \{t[x] \not\simeq x \mid \mathrm{depth}(t) \leq n\}$ instead of Ac.

Getting rid of $\mathcal{R}$: remove every cons symbol.

- Principle : replace every $\mathrm{cons}(c_1, \ldots, c_k) \simeq c$ by $\mathrm{sel}_1(c) \simeq c_1, \ldots, \mathrm{sel}_k(c) \simeq c_k$.

Equisatisfiable sets

Given $S \cup \mathrm{Ac} \cup \mathcal{R}$, apply two steps :

- "Discard" the axioms in $\mathcal{R}$,
- Use $\mathrm{Ac}(n) = \{t[x] \not\simeq x \mid \mathrm{depth}(t) \leq n\}$ instead of Ac.

Getting rid of $\mathcal{R}$: remove every cons symbol.

- Principle : replace every $\mathrm{cons}(c_1, \ldots, c_k) \simeq c$ by $\mathrm{sel}_1(c) \simeq c_1, \ldots, \mathrm{sel}_k(c) \simeq c_k$.
- We obtain $S' \cup \mathrm{Ac} \cup \mathcal{R}$.

Equisatisfiable sets

Given $S \cup \mathrm{Ac} \cup \mathcal{R}$, apply two steps :

- "Discard" the axioms in $\mathcal{R}$,
- Use $\mathrm{Ac}(n) = \{t[x] \not\simeq x \mid \mathrm{depth}(t) \leq n\}$ instead of Ac.

Getting rid of $\mathcal{R}$: remove every cons symbol.

- Principle : replace every $\mathrm{cons}(c_1, \ldots, c_k) \simeq c$ by $\mathrm{sel}_1(c) \simeq c_1, \ldots, \mathrm{sel}_k(c) \simeq c_k$.
- We obtain $S' \cup \mathrm{Ac} \cup \mathcal{R}$.
- cons does not appear in $S' \cup \mathrm{Ac}$.

Getting rid of $\mathcal{R}$

$S \cup \mathrm{Ac} \cup \mathcal{R}$ and $S' \cup \mathrm{Ac}$ are not equisatisfiable.

Getting rid of $\mathcal{R}$

$S \cup \mathrm{Ac} \cup \mathcal{R}$ and $S' \cup \mathrm{Ac}$ are not equisatisfiable.

Example for $k = 1$

$$S = \{s(c_1) \simeq c,\ p(c) \simeq c_2,\ c_1 \not\simeq c_2\}.$$

Getting rid of $\mathcal{R}$

$S \cup \mathrm{Ac} \cup \mathcal{R}$ and $S' \cup \mathrm{Ac}$ are not equisatisfiable.

Example for $k = 1$

$$S = \{s(c_1) \simeq c,\ p(c) \simeq c_2,\ c_1 \not\simeq c_2\}.$$
$$S' = \{s(c_1) \simeq c,\ s(c_2) \simeq c,\ c_1 \not\simeq c_2\}.$$

Getting rid of $\mathcal{R}$

$S \cup \mathrm{Ac} \cup \mathcal{R}$ and $S' \cup \mathrm{Ac}$ are not equisatisfiable.

Example for $k = 1$

$S = \{s(c_1) \simeq c,\ p(c) \simeq c_2,\ c_1 \not\simeq c_2\}$. $S \cup \mathrm{Ac} \cup \mathcal{R}$ is unsatisfiable

$S' = \{s(c_1) \simeq c,\ s(c_2) \simeq c,\ c_1 \not\simeq c_2\}$.

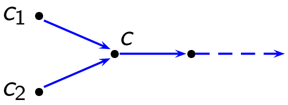
Getting rid of $\mathcal{R}$

$S \cup \mathrm{Ac} \cup \mathcal{R}$ and $S' \cup \mathrm{Ac}$ are not equisatisfiable.

Example for $k = 1$

$S = \{s(c_1) \simeq c,\ p(c) \simeq c_2,\ c_1 \not\simeq c_2\}$. $S \cup \mathrm{Ac} \cup \mathcal{R}$ is unsatisfiable

$S' = \{s(c_1) \simeq c,\ s(c_2) \simeq c,\ c_1 \not\simeq c_2\}$. $S' \cup \mathrm{Ac}$ is satisfiable



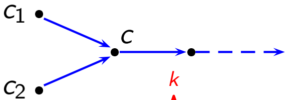
Getting rid of $\mathcal{R}$

$S \cup \mathrm{Ac} \cup \mathcal{R}$ and $S' \cup \mathrm{Ac}$ are not equisatisfiable.

Example for $k = 1$

$S = \{s(c_1) \simeq c,\ p(c) \simeq c_2,\ c_1 \not\simeq c_2\}$. $S \cup \mathrm{Ac} \cup \mathcal{R}$ is unsatisfiable

$S' = \{s(c_1) \simeq c,\ s(c_2) \simeq c,\ c_1 \not\simeq c_2\}$. $S' \cup \mathrm{Ac}$ is satisfiable



ext : $\bigwedge_{i=1}^{k} \mathrm{sel}_i(x) \simeq \mathrm{sel}_i(y) \Rightarrow x \simeq y$

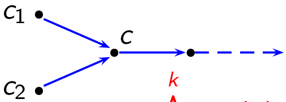
Getting rid of $\mathcal{R}$

$S \cup \mathrm{Ac} \cup \mathcal{R}$ and $S' \cup \mathrm{Ac}$ are not equisatisfiable.

Example for $k = 1$

$S = \{s(c_1) \simeq c,\ p(c) \simeq c_2,\ c_1 \not\simeq c_2\}$. $S \cup \mathrm{Ac} \cup \mathcal{R}$ is unsatisfiable

$S' = \{s(c_1) \simeq c,\ s(c_2) \simeq c,\ c_1 \not\simeq c_2\}$. $S' \cup \mathrm{Ac}$ is satisfiable



$$\textbf{ext} : \bigwedge_{i=1}^{k} \mathrm{sel}_i(x) \simeq \mathrm{sel}_i(y) \Rightarrow x \simeq y$$

Theorem

$S \cup \mathrm{Ac} \cup \mathcal{R}$ and $S' \cup \mathrm{Ac} \cup \{\textbf{ext}\}$ are equisatisfiable.

Replacing Ac by $\mathrm{Ac}(n)$

We want $S' \cup \{\mathbf{ext}\} \cup \mathrm{Ac}$ and $S' \cup \{\mathbf{ext}\} \cup \mathrm{Ac}(n)$ to be equisatisfiable.

Replacing Ac by $\mathrm{Ac}(n)$

We want $S' \cup \{\mathbf{ext}\} \cup \mathrm{Ac}$ and $S' \cup \{\mathbf{ext}\} \cup \mathrm{Ac}(n)$ to be equisatisfiable.

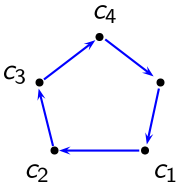
Example : $S' = \{s(c_1) \simeq c_2,\ s(c_2) \simeq c_3,\ s(c_3) \simeq c_4\}$

Replacing Ac by $\mathrm{Ac}(n)$

We want $S' \cup \{\mathbf{ext}\} \cup \mathrm{Ac}$ and $S' \cup \{\mathbf{ext}\} \cup \mathrm{Ac}(n)$ to be equisatisfiable.

Example : $S' = \{s(c_1) \simeq c_2,\ s(c_2) \simeq c_3,\ s(c_3) \simeq c_4\}$

A model of $S' \cup \{\mathbf{ext}\} \cup \mathrm{Ac}(4)$:

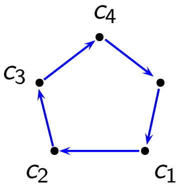


Replacing Ac by $\mathrm{Ac}(n)$

We want $S' \cup \{\mathbf{ext}\} \cup \mathrm{Ac}$ and $S' \cup \{\mathbf{ext}\} \cup \mathrm{Ac}(n)$ to be equisatisfiable.

Example : $S' = \{s(c_1) \simeq c_2,\ s(c_2) \simeq c_3,\ s(c_3) \simeq c_4\}$

A model of $S' \cup \{\mathbf{ext}\} \cup \mathrm{Ac}$:

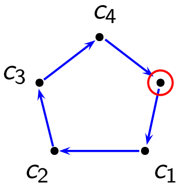


Replacing Ac by $\mathrm{Ac}(n)$

We want $S' \cup \{\textbf{ext}\} \cup \mathrm{Ac}$ and $S' \cup \{\textbf{ext}\} \cup \mathrm{Ac}(n)$ to be equisatisfiable.

Example : $S' = \{s(c_1) \simeq c_2,\ s(c_2) \simeq c_3,\ s(c_3) \simeq c_4\}$

A model of $S' \cup \{\textbf{ext}\} \cup \mathrm{Ac}$:

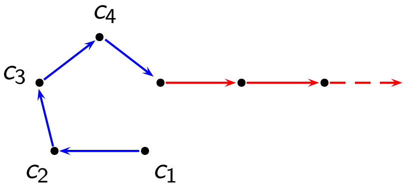


Replacing Ac by $\mathrm{Ac}(n)$

We want $S' \cup \{\mathbf{ext}\} \cup \mathrm{Ac}$ and $S' \cup \{\mathbf{ext}\} \cup \mathrm{Ac}(n)$ to be equisatisfiable.

Example : $S' = \{s(c_1) \simeq c_2,\ s(c_2) \simeq c_3,\ s(c_3) \simeq c_4\}$

A model of $S' \cup \{\mathbf{ext}\} \cup \mathrm{Ac}$:

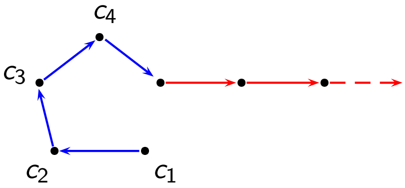


Replacing Ac by $\mathrm{Ac}(n)$

We want $S' \cup \{\mathbf{ext}\} \cup \mathrm{Ac}$ and $S' \cup \{\mathbf{ext}\} \cup \mathrm{Ac}(n)$ to be equisatisfiable.

Example : $S' = \{s(c_1) \simeq c_2,\ s(c_2) \simeq c_3,\ s(c_3) \simeq c_4\}$

A model of $S' \cup \{\mathbf{ext}\} \cup \mathrm{Ac}$:



Theorem

Sufficient condition : $n \geq$ number of selectors in S'.

Summary of the reductions

Input : set S of ground literals.

$$S \cup \mathcal{R} \cup \mathrm{Ac}$$

Summary of the reductions

Input : set S of ground literals.

$$S \cup \mathcal{R} \cup \mathrm{Ac}$$

replace $\mathrm{cons}(c_1, \ldots, c_k) \simeq c$ with

$$\mathrm{sel}_1(c) \simeq c_1, \ldots, \mathrm{sel}_k(c) \simeq c_k$$

Summary of the reductions

Input : set S of ground literals.

$$S \cup \mathcal{R} \cup \mathrm{Ac}$$

$$\updownarrow$$

replace $\mathrm{cons}(c_1, \ldots, c_k) \simeq c$ with

$\mathrm{sel}_1(c) \simeq c_1, \ldots, \mathrm{sel}_k(c) \simeq c_k$

$$S' \cup \{\textbf{ext}\} \cup \mathrm{Ac}$$

Summary of the reductions

Input : set S of ground literals.

$$\begin{array}{ll} S \cup \mathcal{R} \cup \mathrm{Ac} & \\ \big\updownarrow & \text{replace } \mathrm{cons}(c_1, \ldots, c_k) \simeq c \text{ with} \\ & \mathrm{sel}_1(c) \simeq c_1, \ldots, \mathrm{sel}_k(c) \simeq c_k \\ S' \cup \{\mathbf{ext}\} \cup \mathrm{Ac} & \end{array}$$

n = number of selectors in S'

Summary of the reductions

Input : set S of ground literals.

$$S \cup \mathcal{R} \cup \mathrm{Ac}$$

$$\updownarrow \quad \text{replace } \mathrm{cons}(c_1, \ldots, c_k) \simeq c \text{ with } \mathrm{sel}_1(c) \simeq c_1, \ldots, \mathrm{sel}_k(c) \simeq c_k$$

$$S' \cup \{\textbf{ext}\} \cup \mathrm{Ac}$$

$$\updownarrow \quad n = \text{number of selectors in } S'$$

$$S' \cup \{\textbf{ext}\} \cup \mathrm{Ac}(n)$$

Summary of the reductions

Input : set S of ground literals.

$$
\begin{array}{cl}
S \cup \mathcal{R} \cup \mathrm{Ac} & \\
\updownarrow & \text{replace } \mathrm{cons}(c_1, \ldots, c_k) \simeq c \text{ with} \\
 & \mathrm{sel}_1(c) \simeq c_1, \ldots, \mathrm{sel}_k(c) \simeq c_k \\
S' \cup \{\textbf{ext}\} \cup \mathrm{Ac} & \\
\updownarrow & n = \text{number of selectors in } S' \\
S' \cup \{\textbf{ext}\} \cup \mathrm{Ac}(n) &
\end{array}
$$

Theorem

$\mathcal{SP}_{\succ}$ with a fair search plan terminates on $S' \cup \{\textbf{ext}\} \cup \mathrm{Ac}(n)$.

Outline

- Preliminaries
- Rewrite-based $\mathcal{T}$-satisfiability procedures
- Recursive Data Structures
- Combination and complexity
- Discussion

Combination

Suppose $\mathcal{SP}_{\succ}$ with a fair search plan terminates on $\mathcal{T}_1 \cup S_1$ and $\mathcal{T}_2 \cup S_2$.

Under what conditions do we have termination on $\mathcal{T}_1 \cup \mathcal{T}_2 \cup S$?

Combination

Suppose $\mathcal{SP}_{\succ}$ with a fair search plan terminates on $\mathcal{T}_1 \cup S_1$ and $\mathcal{T}_2 \cup S_2$.

Under what conditions do we have termination on $\mathcal{T}_1 \cup \mathcal{T}_2 \cup S$?

- Sufficient condition : variable-inactivity
  (Armando, Bonacina, Ranise, Schulz. 2005)
  - Intuition : no paramodulations from variables across theories.

Combination

Suppose $\mathcal{SP}_{\succ}$ with a fair search plan terminates on $\mathcal{T}_1 \cup S_1$ and $\mathcal{T}_2 \cup S_2$.

Under what conditions do we have termination on $\mathcal{T}_1 \cup \mathcal{T}_2 \cup S$?

- Sufficient condition : variable-inactivity
  (Armando, Bonacina, Ranise, Schulz. 2005)
  - Intuition : no paramodulations from variables across theories.
- The following theories can all be combined together :
  - Equality, Lists, Arrays, Records...

Combination

Suppose $\mathcal{SP}_{\succ}$ with a fair search plan terminates on $\mathcal{T}_1 \cup S_1$ and $\mathcal{T}_2 \cup S_2$.

Under what conditions do we have termination on $\mathcal{T}_1 \cup \mathcal{T}_2 \cup S$?

- Sufficient condition : variable-inactivity
  (Armando, Bonacina, Ranise, Schulz. 2005)
  - Intuition : no paramodulations from variables across theories.
- The following theories can all be combined together :
  - Equality, Lists, Arrays, Records...

Theorem

Any theory of recursive data structures can be combined with the ones mentioned above.

Complexity

Clauses in S_∞ :

- In $\{\mathbf{ext}\} \cup \mathrm{Ac}(n)$: $O(n)$ if $k = 1$, $O(k^n)$ if $k \geq 2$.

Complexity

Clauses in S_∞ :

- In $\{\textbf{ext}\} \cup \mathrm{Ac}(n)$: $O(n)$ if $k = 1$, $O(k^n)$ if $k \geq 2$.
- Flat ground literals : $O(n^2)$.

Complexity

Clauses in S_∞ :

- In $\{\textbf{ext}\} \cup \mathrm{Ac}(n)$: $O(n)$ if $k = 1$, $O(k^n)$ if $k \geq 2$.
- Flat ground literals : $O(n^2)$.
- Generated clauses : $O(2^{n^2})$.

Complexity

Clauses in S_∞ :

- In $\{\textbf{ext}\} \cup \mathrm{Ac}(n)$: $O(n)$ if $k = 1$, $O(k^n)$ if $k \geq 2$.
- Flat ground literals : $O(n^2)$.
- Generated clauses : $O(2^{n^2})$.

S_∞ contains an exponential number of clauses.

The rewrite-based satisfiability procedure is exponential for every $k \geq 1$.

Discussion

- Rewrite-based satisfiability procedures for Recursive Data Structures
  - Uniformity : same technique for every $k \geq 1$
  - Combination
  - Complexity issue : exponentiality for $k = 1$

Discussion

- Rewrite-based satisfiability procedures for Recursive Data Structures
  - Uniformity : same technique for every $k \geq 1$
  - Combination
  - Complexity issue : exponentiality for $k = 1$
- Breaking news : the complexity issue is solved.

Solving the complexity issue

$$
\begin{tikzcd}
S \cup \mathcal{R} \cup \mathrm{Ac} \arrow[d, leftrightarrow] & \\
S' \cup \{\textbf{ext}\} \cup \mathrm{Ac} \arrow[r, leftrightarrow] & S' \cup \{\textbf{ext}\} \cup \mathrm{Ac}(n)
\end{tikzcd}
$$

Solving the complexity issue

$$
\begin{tikzcd}
S \cup \mathcal{R} \cup \mathrm{Ac} \arrow[d, leftrightarrow] & & S \cup \mathcal{R} \cup \mathrm{Ac}(n) \arrow[d, leftrightarrow] \\
S' \cup \{\textbf{ext}\} \cup \mathrm{Ac} \arrow[rr, leftrightarrow] & & S' \cup \{\textbf{ext}\} \cup \mathrm{Ac}(n)
\end{tikzcd}
$$

Solving the complexity issue

$$
\begin{array}{ccc}
S \cup \mathcal{R} \cup \mathrm{Ac} & & S \cup \mathcal{R} \cup \mathrm{Ac}(n) \\
\updownarrow & & \updownarrow \\
S' \cup \{\textbf{ext}\} \cup \mathrm{Ac} & \longleftrightarrow & S' \cup \{\textbf{ext}\} \cup \mathrm{Ac}(n)
\end{array}
$$

Clauses in S_∞ for $k = 1$:

- In $\mathcal{R} \cup \mathrm{Ac}(n)$: $O(n)$.
- Flat ground literals : $O(n^2)$.
- Generated clauses : $O(n^2)$.

Solving the complexity issue

$$\begin{tikzcd}
S \cup \mathcal{R} \cup \mathrm{Ac} \arrow[d, leftrightarrow] & & {\color{red} S \cup \mathcal{R} \cup \mathrm{Ac}(n)} \arrow[d, leftrightarrow] \\
S' \cup \{\mathbf{ext}\} \cup \mathrm{Ac} \arrow[rr, leftrightarrow] & & S' \cup \{\mathbf{ext}\} \cup \mathrm{Ac}(n)
\end{tikzcd}$$

Clauses in S_∞ for $k = 1$:

- In $\mathcal{R} \cup \mathrm{Ac}(n)$: $O(n)$.
- Flat ground literals : $O(n^2)$.
- Generated clauses : $O(n^2)$.

The rewrite-based satisfiability procedure is polynomial for $k = 1$.

Solving the complexity issue

$$\begin{array}{ccc} S \cup \mathcal{R} \cup \mathrm{Ac} & & S \cup \mathcal{R} \cup \mathrm{Ac}(n) \\ \updownarrow & & \updownarrow \\ S' \cup \{\mathbf{ext}\} \cup \mathrm{Ac} & \longleftrightarrow & S' \cup \{\mathbf{ext}\} \cup \mathrm{Ac}(n) \end{array}$$

Clauses in S_∞ for $k = 1$:

- In $\mathcal{R} \cup \mathrm{Ac}(n)$: $O(n)$.
- Flat ground literals : $O(n^2)$.
- Generated clauses : $O(n^2)$.

The rewrite-based satisfiability procedure is polynomial for $k = 1$.

Thank you for your attention.