Outline

- Preliminaries
- Rewrite-based $T$-satisfiability procedures
- Recursive Data Structures
- Combination and complexity
- Discussion
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
    \[ \forall x, z, v. \; \text{select}\left(\text{store}(x, z, v), z\right) \equiv v, \]
    \[ \forall x, z, v, w. \; z \not\equiv w \Rightarrow \text{select}\left(\text{store}(x, z, v), w\right) \equiv \text{select}(x, w), \]
    \[ \forall x, y. \; (\forall z. \; \text{select}(x, z) \equiv \text{select}(y, z) \Rightarrow x \equiv y). \quad (\text{ext}) \]
Background

- First-order logic with equality
- Theories of data structures (or their combination)
  - Theory of arrays with or without extensionality
    \[
    \forall x, z, v. \ select(store(x, z, v), z) \simeq v, \\
    \forall x, z, v, w. \ z \neq w \Rightarrow \ select(store(x, z, v), w) \simeq \ select(x, w), \\
    \forall x, y. \ (\forall z. \ select(x, z) \simeq \ select(y, z) \Rightarrow x \simeq y). \quad (\text{ext})
    \]
  - Theory of nonempty lists
Background

- First-order logic with equality
- Theories of data structures (or their combination)
  - Theory of arrays with or without extensionality
    \[
    \forall x, z, v. \text{select}(\text{store}(x, z, v), z) \simeq v, \\
    \forall x, z, v, w. z \not\equiv w \Rightarrow \text{select}(\text{store}(x, z, v), w) \simeq \text{select}(x, w), \\
    \forall x, y. (\forall z. \text{select}(x, z) \simeq \text{select}(y, z) \Rightarrow x \simeq y). \quad (\text{ext})
    \]
  - Theory of nonempty lists
    \[
    \forall x. \text{cons}(\text{car}(x), \text{cdr}(x)) \simeq x, \\
    \forall x, y. \text{car}(\text{cons}(x, y)) \simeq x, \\
    \forall x, y. \text{cdr}(\text{cons}(x, y)) \simeq y.
    \]
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: $\mathcal{S}\mathcal{P}$.

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

- $T$-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 $\rightarrow$ arrays without extensionality.

- **Flattening**: flatten all ground literals to obtain a flat $\mathcal{T}$-satisfiability problem.
  Example: replace $f(f(a)) \approx b$ by $f(a) \approx a_1$ and $f(a_1) \approx b$.

- **Proof of termination**: prove that $\mathcal{SP}$ with a fair search plan terminates.
A three-step methodology

- \( T \)-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 \( T \)-satisfiability problem.
  Example: replace \( f(f(a)) \approx b \) by \( f(a) \approx a_1 \) and \( f(a_1) \approx b \).

- Proof of termination: prove that \( SP \) with a fair search plan terminates.

The first two steps can be performed automatically.
Termination/Complexity

What kind of clauses can $S_\infty$ contain?
Termination/Complexity

What kind of clauses can $S_{\infty}$ contain? Prove that these clauses belong to a finite number of finite categories.
Termination/Complexity

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

- **Termination:**
  - $S_\infty$ is finite.
  - $S\mathcal{P}$ with a fair search plan terminates.
Termination/Complexity

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

- **Termination**:
  - $S_\infty$ is finite.
  - $SP_\succ$ with a fair search plan terminates.

- **Complexity**:
  - Determine the number of clauses in each category.
  - Upper-bound on the size of $S_\infty$. 

Termination/Complexity

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

- **Termination:**
  - $S_\infty$ is finite.
  - $SP_\succ$ with a fair search plan terminates.

- **Complexity:**
  - Determine the number of clauses in each category.
  - Upper-bound on the size of $S_\infty$.

Some upper-bounds on the sizes of $S_\infty$:

- 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: $\text{cdr}(x) \not\approx x$, $\text{cdr}(	ext{cdr}(x)) \not\approx x$, etc...
- How do we ensure termination?
Outline

- Preliminaries
- Rewrite-based $\mathcal{T}$-satisfiability procedures
- Recursive Data Structures
- Combination and complexity
- Discussion
A constructor of arity $k$, and $k$ selectors of arity 1, where $k \geq 1$. 
A constructor of arity $k$, and $k$ selectors of arity 1, where $k \geq 1$.

$k = 1$ : integer offsets

\[
p(s(x)) \simeq x
\]
\[
s(p(x)) \simeq x
\]
\[
\{s^i(x) \not\simeq x \mid i > 0\}
\]
Definition

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

$k = 2$ : *acyclic lists*

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

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

$$\{\text{car}(x) \not\simeq x, \ \text{cdr}(x) \not\simeq x, \ \text{car(cdr}(x)) \not\simeq x \ldots\}$$
**Definition**

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

For any $k \geq 1$:

- $$\text{cons}(\text{sel}_1(x), \ldots, \text{sel}_k(x)) \simeq x$$
- $$\{ \text{sel}_i(\text{cons}(x_1, \ldots, x_k)) \simeq x_i \mid 1 \leq i \leq k \}$$
- $$\{ t[x] \not\simeq x \}$$
A constructor of arity $k$, and $k$ selectors of arity 1, where $k \geq 1$.

For any $k \geq 1$:

\[
\text{cons}(\text{sel}_1(x), \ldots, \text{sel}_k(x)) \simeq x \quad \text{R} \\
\{\text{sel}_i(\text{cons}(x_1, \ldots, x_k)) \simeq x_i \mid 1 \leq i \leq k\} \\
\{t[x] \not\simeq x\} \quad \text{Ac}
\]
Definition

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

For any $k \geq 1$:

$\text{cons}(\text{sel}_1(x), \ldots, \text{sel}_k(x)) \simeq x \quad \text{for any } k \geq 1$

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

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

There can be no termination result on $S \cup \text{Ac} \cup \mathcal{R}$. 
A constructor of arity $k$, and $k$ selectors of arity 1, where $k \geq 1$.

For any $k \geq 1$:

$$\text{cons}(\text{sel}_1(x), \ldots, \text{sel}_k(x)) \simeq x \ \{\text{sel}_i(\text{cons}(x_1, \ldots, x_k)) \simeq x_i \mid 1 \leq i \leq k\}$$

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

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

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

Given $S \cup Ac \cup R$, apply two steps:

- “Discard” the axioms in $R$,
- Use $Ac(n) = \{ t[x] \not\equiv x \ | \ depth(t) \leq n \}$ instead of $Ac$. 
Equisatisfiable sets

Given $S \cup Ac \cup R$, apply two steps:

- "Discard" the axioms in $R$,
- Use $Ac(n) = \{ t[x] \neq x \mid \text{depth}(t) \leq n \}$ instead of $Ac$.

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

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

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

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

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

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

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

- “Discard” the axioms in $\mathcal{R}$,
- Use $Ac(n) = \{ t[x] \neq x \mid \text{depth}(t) \leq n \}$ instead of $Ac$.

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

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

$S \cup \text{Ac} \cup \mathcal{R}$ and $S' \cup \text{Ac}$ are not equisatisfiable.
Getting rid of $\mathcal{R}$

$S \cup Ac \cup \mathcal{R}$ and $S' \cup Ac$ 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 $\mathcal{R}$

$S \cup \mathcal{A}c \cup \mathcal{R}$ and $S' \cup \mathcal{A}c$ 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 $\mathcal{R}$

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

Example for $k = 1$

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

$S' = \{s(c_1) \simeq c, \ s(c_2) \simeq c, \ c_1 \not\approx c_2\}$. 
Getting rid of $\mathcal{R}$

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

Example for $k = 1$

\[
S = \{ s(c_1) \simeq c, \ p(c) \simeq c_2, \ c_1 \not\simeq c_2 \}. \quad S \cup \text{Ac} \cup \mathcal{R} \text{ is unsatisfiable}
\]

\[
S' = \{ s(c_1) \simeq c, \ s(c_2) \simeq c, \ c_1 \not\simeq c_2 \}. \quad S' \cup \text{Ac} \text{ is satisfiable}
\]
Getting rid of $\mathcal{R}$

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

Example for $k = 1$

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

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

\[
\text{ext} : \bigwedge_{i=1}^{k} \text{sel}_i(x) \simeq \text{sel}_i(y) \Rightarrow x \simeq y
\]
Getting rid of $R$

$S \cup Ac \cup R$ and $S' \cup Ac$ are not equisatisfiable.

Example for $k = 1$

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

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

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

Theorem

$S \cup Ac \cup R$ and $S' \cup Ac \cup \{\text{ext}\}$ are equisatisfiable.
Replacing $Ac$ by $Ac(n)$

We want $S' \cup \{\text{ext}\} \cup Ac$ and $S' \cup \{\text{ext}\} \cup Ac(n)$ to be equisatisfiable.
Replacing Ac by Ac(n)

We want \( S' \cup \{\text{ext}\} \cup \text{Ac} \) and \( S' \cup \{\text{ext}\} \cup \text{Ac}(n) \) 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 $Ac(n)$

We want $S' \cup \{\text{ext}\} \cup Ac$ and $S' \cup \{\text{ext}\} \cup Ac(n)$ 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 $S' \cup \{\text{ext}\} \cup Ac(4)$:
Replacing $\text{Ac}$ by $\text{Ac}(n)$

We want $S' \cup \{\text{ext}\} \cup \text{Ac}$ and $S' \cup \{\text{ext}\} \cup \text{Ac}(n)$ 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 $S' \cup \{\text{ext}\} \cup \text{Ac}$:

```
  c_4
  /|
 / |
 c_3
 /  |
/   |
c_2
  |
  |
  c_1
```
Replacing $Ac$ by $Ac(n)$

We want $S' \cup \{\text{ext}\} \cup Ac$ and $S' \cup \{\text{ext}\} \cup Ac(n)$ 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 $S' \cup \{\text{ext}\} \cup Ac$:

```
  ┌───────┐
  │      │
  │      │  c_4
  │      │
  │      │
  └───┬───┘
       │   
    c_3 

  ┌───────┐
  │      │
  │      │  c_2
  │      │
  │      │
  └───┬───┘
       │   
    c_1
```
Replacing $Ac$ by $Ac(n)$

We want $S' \cup \{\text{ext}\} \cup Ac$ and $S' \cup \{\text{ext}\} \cup Ac(n)$ 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 $S' \cup \{\text{ext}\} \cup Ac$:

```
   c4
  /   \
 /     \
 c3     c2
  |     |
  |     |
  |     |
  c1
```
Replacing $\text{Ac}$ by $\text{Ac}(n)$

We want $S' \cup \{\text{ext}\} \cup \text{Ac}$ and $S' \cup \{\text{ext}\} \cup \text{Ac}(n)$ 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 $S' \cup \{\text{ext}\} \cup \text{Ac}$:

![Diagram](image)

Theorem

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

Maria Paola Bonacina and Mnacho Echenim

Rewrite-based satisfiability procedures for Recursive Data Structures
Summary of the reductions

Input : set $S$ of ground literals.

$$S \cup R \cup Ac$$
Summary of the reductions

Input: set $S$ of ground literals.

$$S \cup \mathcal{R} \cup \mathcal{A}$$

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

$$\text{sel}_1(c) \simeq c_1, \ldots, \text{sel}_k(c) \simeq c_k$$
Summary of the reductions

Input: set $S$ of ground literals.

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

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

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

$$S' \cup \{\text{ext}\} \cup \text{Ac}$$
Summary of the reductions

Input: set $S$ of ground literals.

\[ S \cup \mathcal{R} \cup \mathcal{A}_c \]

replace $\text{cons}(c_1, \ldots, c_k) \simeq c$ with
\[ \text{sel}_1(c) \simeq c_1, \ldots, \text{sel}_k(c) \simeq c_k \]

\[ S' \cup \{\text{ext}\} \cup \mathcal{A}_c \]

\[ n = \text{number of selectors in } S' \]
Summary of the reductions

Input: set $S$ of ground literals.

\[
S \cup R \cup Ac \\
\leftrightarrow \\
S' \cup \{\text{ext}\} \cup Ac
\]

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

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

\[
S' \cup \{\text{ext}\} \cup Ac(n)
\]
Summary of the reductions

Input: set $S$ of ground literals.

\[
S \cup \mathcal{R} \cup \text{Ac}
\]

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

\[
\text{sel}_1(c) \simeq c_1, \ldots, \text{sel}_k(c) \simeq c_k
\]

\[
S' \cup \{\text{ext}\} \cup \text{Ac}
\]

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

\[
S' \cup \{\text{ext}\} \cup \text{Ac}(n)
\]

Theorem

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

- Preliminaries
- Rewrite-based $\mathcal{T}$-satisfiability procedures
- Recursive Data Structures
- Combination and complexity
- Discussion
Suppose $SP_{\succ}$ with a fair search plan terminates on $T_1 \cup S_1$ and $T_2 \cup S_2$.

Under what conditions do we have termination on $T_1 \cup T_2 \cup S$?
Combination

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

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

- 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...
Suppose $SP_\succ$ 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 $S_\infty$:

- $\text{ln } \{\text{ext} \} \cup \text{Ac}(n) : O(n)$ if $k = 1$, $O(k^n)$ if $k \geq 2$. 
Complexity

Clauses in $S_\infty$:

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

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

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

$S_\infty$ 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

\[ S \cup \mathcal{R} \cup \text{Ac} \]

\[ S' \cup \{\text{ext}\} \cup \text{Ac} \leftrightarrow \rightarrow S' \cup \{\text{ext}\} \cup \text{Ac}(n) \]
Solving the complexity issue

\[ S \cup \mathcal{R} \cup \text{Ac} \]

\[ S' \cup \{\text{ext}\} \cup \text{Ac} \quad \leftrightarrow \quad S' \cup \{\text{ext}\} \cup \text{Ac}(n) \]

\[ S \cup \mathcal{R} \cup \text{Ac}(n) \]
Solving the complexity issue

\[
S \cup \mathcal{R} \cup \text{Ac} \\
S' \cup \{\text{ext}\} \cup \text{Ac} \\
S \cup \mathcal{R} \cup \text{Ac}(n) \\
S' \cup \{\text{ext}\} \cup \text{Ac}(n)
\]

Clauses in \(S_\infty\) for \(k = 1\):

- In \(\mathcal{R} \cup \text{Ac}(n)\) : \(O(n)\).
- Flat ground literals : \(O(n^2)\).
- Generated clauses : \(O(n^2)\).
Solving the complexity issue

\[ S \cup \mathcal{R} \cup \text{Ac} \]

\[ S' \cup \{\text{ext}\} \cup \text{Ac} \]

\[ S \cup \mathcal{R} \cup \text{Ac}(n) \]

\[ S' \cup \{\text{ext}\} \cup \text{Ac}(n) \]

Clauses in \( S_\infty \) for \( k = 1 \):
- \( \text{In } \mathcal{R} \cup \text{Ac}(n) : O(n) \).
- 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

\[ S \cup \mathcal{R} \cup \text{Ac} \]

\[ S' \cup \{\text{ext}\} \cup \text{Ac} \quad \leftrightarrow \quad S' \cup \{\text{ext}\} \cup \text{Ac}(n) \]

Clauses in \( S_\infty \) for \( k = 1 \):
- \( \text{In} \ \mathcal{R} \cup \text{Ac}(n) : O(n) \).
- 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.