Structural Resolution

Katya Komendantskaya (joint work with Patricia Johann)

School of Computing, University of Dundee, UK

02 September 2015

Resolution rule

- Propositional:

$$\frac{C \vee A \quad \neg A \vee D}{C \vee D}$$

Resolution rule

- Propositional:

$$\frac{C \vee A \quad \neg A \vee D}{C \vee D}$$

- First-order:

$$\frac{C \vee A \quad \neg B \vee D}{\theta(C) \vee \theta(D)},$$

if θ is a unifier of A and B (i.e., $\theta(A) = \theta(B)$.

Restricted Resolution by term-matching

$$\frac{C \vee A \quad \neg B \vee D}{\theta(C) \vee D},$$

where θ is a matcher of A and B (i.e., $\theta(A) = B$).

Restricted Resolution by term-matching

$$\frac{C \vee A \quad \neg B \vee D}{\theta(C) \vee D},$$

where θ is a matcher of A and B (i.e., $\theta(A) = B$).

... incomplete relative to the usual resolution rule...

Structural Resolution

1. (the "term-matching rule"):

$$\frac{C \vee A \quad \neg B \vee D}{\theta(C) \vee D},$$

where θ is a matcher of A and B (i.e., $\theta(A) = B$).

Structural Resolution

1. (the "term-matching rule"):

$$\frac{C \vee A \quad \neg B \vee D}{\theta(C) \vee D},$$

   where θ is a matcher of A and B (i.e., $\theta(A) = B$).

2. plus the rule

$$\frac{C \vee A \quad \neg B \vee D}{C \vee A, \quad \theta(\neg B) \vee \theta(D)},$$

   where θ is a unifier of A and B.

SLD-resolution

... is an instance of standard resolution rule:

$$\frac{C_1 \vee \ldots \vee C_n \vee A \quad \neg B \vee \neg D_1 \vee \ldots \vee \neg D_k}{\theta(C_1 \vee \ldots \vee C_n) \vee \theta(\neg D_1 \vee \ldots \vee \neg D_k)},$$

if θ is a unifier of A and B (i.e., $\theta(A) = \theta(B)$.

Example 1

Example

1.$\texttt{nat(0)} \leftarrow$

2.$\texttt{nat(s(x))} \leftarrow \texttt{nat(x)}$

Computes $x \mapsto 0$, and other natural numbers with the use of backtracking.

$$\leftarrow \texttt{nat(x)}$$

$$|$$

$$\square$$

Example 2

Nat2:

Example

1.$\texttt{nat(s(x))} \leftarrow \texttt{nat(x)}$
2.$\texttt{nat(0)} \leftarrow$

Computes $x \mapsto s(s(s\ldots)))$, the first limit ordinal.

$$\begin{array}{c} \leftarrow \texttt{nat}(\texttt{x}) \\ | \\ \leftarrow \texttt{nat}(\texttt{x}') \\ | \\ \leftarrow \texttt{nat}(\texttt{x}'') \\ | \\ \ldots \end{array}$$

Example 3

Program **Stream**:

Example

```
1.bit(0) ←
2.bit(1) ←
3.stream(scons(x,y)) ←
        bit(x), stream(y)
```

Computes an infinite stream.

$$\leftarrow \mathtt{stream}(\mathtt{scons}(x, y))$$

$$\leftarrow \mathtt{bit}(x), \mathtt{stream}(y)$$

$$\leftarrow \mathtt{stream}(y)$$

$$\leftarrow \mathtt{bit}(x_1), \mathtt{stream}(y_1)$$

$$\leftarrow \mathtt{stream}(y_1)$$

$$\vdots$$

Problems...

What does it mean if your program does not terminate?

Problems...

What does it mean if your program does not terminate?

- May be it is a corecursive program, like **Stream**...

Problems...

What does it mean if your program does not terminate?

- May be it is a corecursive program, like **Stream**...
- May be it is a recursive program, but badly ordered, like **Nat2**...

Problems...

What does it mean if your program does not terminate?

- May be it is a corecursive program, like **Stream**...
- May be it is a recursive program, but badly ordered, like **Nat2**...
- Or may be it is a recursive program with coinductive interpretation? (again, **Nat2**)

Problems...

What does it mean if your program does not terminate?

- May be it is a corecursive program, like **Stream**...
- May be it is a recursive program, but badly ordered, like **Nat2**...
- Or may be it is a recursive program with coinductive interpretation? (again, **Nat2**)
- Or may be it is just some bad loop without particular computational meaning:

$$bad(x) \leftarrow bad(x)$$

Problems...

What does it mean if your program does not terminate?

- May be it is a corecursive program, like **Stream**...
- May be it is a recursive program, but badly ordered, like **Nat2**...
- Or may be it is a recursive program with coinductive interpretation? (again, **Nat2**)
- Or may be it is just some bad loop without particular computational meaning:

$$bad(x) \leftarrow bad(x)$$

We are missing a theory to talk about such things...

Nat in S-resolution

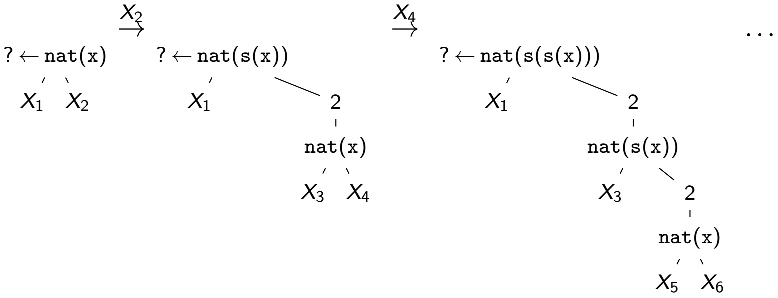
1.$\texttt{nat(0)} \leftarrow$
2.$\texttt{nat(s(x))} \leftarrow \texttt{nat(x)}$

$$\begin{array}{c} ? \leftarrow \texttt{nat(X)} \\ / \quad \backslash \\ X_1 \quad X_2 \end{array} \quad \xrightarrow{X_1} \quad \begin{array}{c} ? \leftarrow \texttt{nat(0)} \\ / \quad \backslash \\ 1 \quad X_2 \end{array}$$

Nat2 in S-resolution

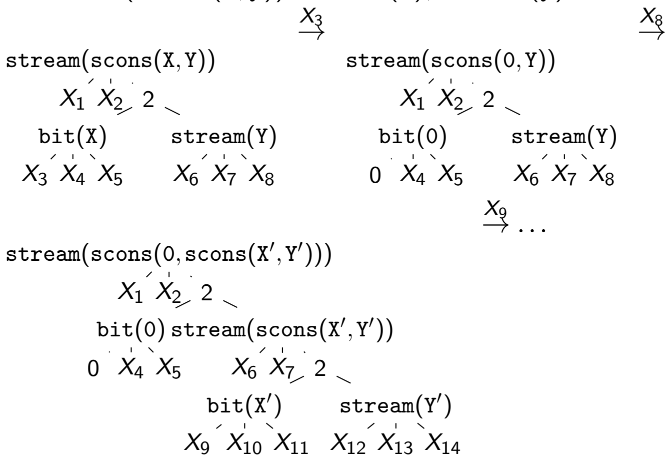
1.$\texttt{nat(0)} \leftarrow$

2.$\texttt{nat(s(x))} \leftarrow \texttt{nat(x)}$



Streams in S-resolution

0. $\texttt{bit}(\texttt{0}) \leftarrow$
1. $\texttt{bit}(\texttt{1}) \leftarrow$
2. $\texttt{stream}(\texttt{scons}(\texttt{x},\texttt{y})) \leftarrow \texttt{bit}(\texttt{x}), \texttt{stream}(\texttt{y})$



New theory of universal productivity for resolution

A program P is productive, if it gives rise to rewriting trees of finite size.

New theory of universal productivity for resolution

A program P is productive, if it gives rise to rewriting trees of finite size.

In the class of Productive LPs, we can further distinguish:

- finite LP that give rise only to finite derivations,
- inductive LPs for which derivations can be finite or infinite;
- coinductive LPs all derivations for which are infinite

New theory of universal productivity for resolution

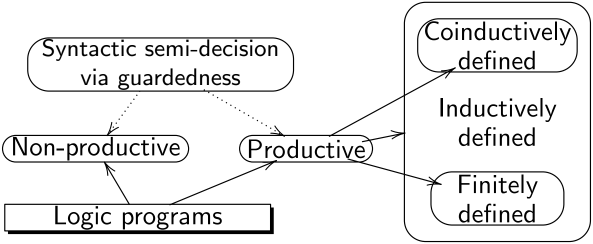
A program P is productive, if it gives rise to rewriting trees of finite size.

In the class of Productive LPs, we can further distinguish:

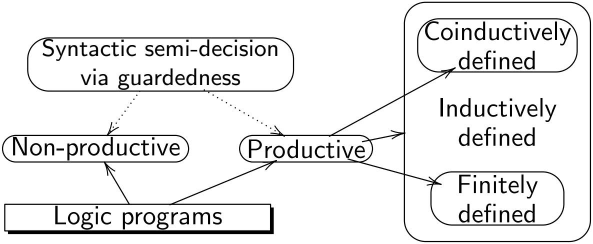
- finite LP that give rise only to finite derivations,
- inductive LPs for which derivations can be finite or infinite;
- coinductive LPs all derivations for which are infinite

Nat and Nat2	Infinite streams.	Bad recursion.
$\mathtt{nat(s(x))} \leftarrow \mathtt{nat(x)}$ $\mathtt{nat(0)} \leftarrow$ inductive definition	$\mathtt{stream(scons(x,y))} \leftarrow \mathtt{stream(y)}$ coinductive definition	$\mathtt{bad(x)} \leftarrow \mathtt{bad(x)}$ non-well-founded
Productive inductive program	Productive coinductive program	Non-productive program
finite rewriting trees, possibly infinite derivations	finite rewriting trees, necessarily infinite derivations	infinite rewriting trees

Theory of universal Productivity in LP!



Theory of universal Productivity in LP!



This and more

... in my poster session