Logically Constrained Term Rewriting Systems

Cynthia Kop and Naoki Nishida

Sept 20, 2013

Logically Constrained Term Rewriting Systems

Cynthia Kop and Naoki Nishida

Sept 20, 2013

Frontiers of Combining Systems

Frontiers of Combining Systems

Term Rewriting

and

SMT

Term Rewriting: Traditional Example

$$\begin{aligned} \mathsf{sum}(0) &\to 0 \\ \mathsf{sum}(\mathsf{s}(x)) &\to \mathsf{plus}(\mathsf{s}(x), \mathsf{sum}(x)) \end{aligned}$$

Term Rewriting: Traditional Example

$$
\begin{array}{rcl}
\mathsf{sum}(0) & \to & 0 \\
\mathsf{sum}(\mathsf{s}(x)) & \to & \mathsf{plus}(\mathsf{s}(x), \mathsf{sum}(x)) \\
\mathsf{plus}(0, y) & \to & y \\
\mathsf{plus}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{plus}(x, y))
\end{array}
$$

Term Rewriting: Traditional Example

$$\begin{array}{rcl} \mathsf{sum}(0) & \to & 0 \\ \mathsf{sum}(\mathsf{s}(x)) & \to & \mathsf{plus}(\mathsf{s}(x), \mathsf{sum}(x)) \\ \mathsf{plus}(0, y) & \to & y \\ \mathsf{plus}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{plus}(x, y)) \end{array}$$

$$\mathsf{sum}(\mathsf{s}(0))$$

Term Rewriting: Traditional Example

$$\begin{array}{rcl} \mathsf{sum}(0) & \to & 0 \\ \mathsf{sum}(\mathsf{s}(x)) & \to & \mathsf{plus}(\mathsf{s}(x), \mathsf{sum}(x)) \\ \mathsf{plus}(0, y) & \to & y \\ \mathsf{plus}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{plus}(x, y)) \end{array}$$

$$\mathsf{sum}(\mathsf{s}(0))$$

Term Rewriting: Traditional Example

$$\begin{aligned} \mathsf{sum}(0) &\to 0 \\ \mathsf{sum}(\mathsf{s}(x)) &\to \mathsf{plus}(\mathsf{s}(x), \mathsf{sum}(x)) \\ \mathsf{plus}(0, y) &\to y \\ \mathsf{plus}(\mathsf{s}(x), y) &\to \mathsf{s}(\mathsf{plus}(x, y)) \end{aligned}$$

$$\begin{aligned} &\mathsf{sum}(\mathsf{s}(0)) \\ &\to \mathsf{plus}(\mathsf{s}(0), \mathsf{sum}(0)) \end{aligned}$$

Term Rewriting: Traditional Example

$$\begin{aligned} \mathsf{sum}(0) &\to 0 \\ \mathsf{sum}(\mathsf{s}(x)) &\to \mathsf{plus}(\mathsf{s}(x), \mathsf{sum}(x)) \\ \mathsf{plus}(0, y) &\to y \\ \mathsf{plus}(\mathsf{s}(x), y) &\to \mathsf{s}(\mathsf{plus}(x, y)) \end{aligned}$$

$$\begin{aligned} &\mathsf{sum}(\mathsf{s}(0)) \\ &\to \mathsf{plus}(\mathsf{s}(0), \mathsf{sum}(0)) \end{aligned}$$

Term Rewriting: Traditional Example

$$
\begin{array}{rcl}
\mathsf{sum}(0) & \to & 0 \\
\mathsf{sum}(\mathsf{s}(x)) & \to & \mathsf{plus}(\mathsf{s}(x), \mathsf{sum}(x)) \\
\mathsf{plus}(0, y) & \to & y \\
\mathsf{plus}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{plus}(x, y))
\end{array}
$$

$$
\begin{array}{l}
\mathsf{sum}(\mathsf{s}(0)) \\
\quad \to \mathsf{plus}(\mathsf{s}(0), \mathsf{sum}(0))
\end{array}
$$

Term Rewriting: Traditional Example

$$
\begin{array}{rcl}
\mathsf{sum}(0) & \to & 0 \\
\mathsf{sum}(\mathsf{s}(x)) & \to & \mathsf{plus}(\mathsf{s}(x), \mathsf{sum}(x)) \\
\mathsf{plus}(0, y) & \to & y \\
\mathsf{plus}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{plus}(x, y))
\end{array}
$$

$$
\begin{array}{l}
\mathsf{sum}(\mathsf{s}(0)) \\
\quad \to \mathsf{plus}(\mathsf{s}(0), \mathsf{sum}(0)) \\
\quad \to \mathsf{plus}(\mathsf{s}(0), 0)
\end{array}
$$

Term Rewriting: Traditional Example

$$
\begin{array}{rcl}
\mathsf{sum}(0) & \to & 0 \\
\mathsf{sum}(\mathsf{s}(x)) & \to & \mathsf{plus}(\mathsf{s}(x), \mathsf{sum}(x)) \\
\mathsf{plus}(0, y) & \to & y \\
\mathsf{plus}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{plus}(x, y))
\end{array}
$$

$$
\begin{array}{l}
\mathsf{sum}(\mathsf{s}(0)) \\
\quad \to \mathsf{plus}(\mathsf{s}(0), \mathsf{sum}(0)) \\
\quad \to \mathsf{plus}(\mathsf{s}(0), 0)
\end{array}
$$

Term Rewriting: Traditional Example

$$\begin{aligned}
\mathsf{sum}(0) &\to 0 \\
\mathsf{sum}(\mathsf{s}(x)) &\to \mathsf{plus}(\mathsf{s}(x), \mathsf{sum}(x)) \\
\mathsf{plus}(0, y) &\to y \\
\mathsf{plus}(\mathsf{s}(x), y) &\to \mathsf{s}(\mathsf{plus}(x, y))
\end{aligned}$$

$$\begin{aligned}
&\mathsf{sum}(\mathsf{s}(0)) \\
&\to \mathsf{plus}(\mathsf{s}(0), \mathsf{sum}(0)) \\
&\to \mathsf{plus}(\mathsf{s}(0), 0) \\
&\to \mathsf{s}(\mathsf{plus}(0, 0))
\end{aligned}$$

Term Rewriting: Traditional Example

$$\begin{array}{rcl} \mathsf{sum}(0) & \to & 0 \\ \mathsf{sum}(\mathsf{s}(x)) & \to & \mathsf{plus}(\mathsf{s}(x), \mathsf{sum}(x)) \\ \mathsf{plus}(0, y) & \to & y \\ \mathsf{plus}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{plus}(x, y)) \end{array}$$

$$\begin{array}{l} \mathsf{sum}(\mathsf{s}(0)) \\ \quad \to \mathsf{plus}(\mathsf{s}(0), \mathsf{sum}(0)) \\ \quad \to \mathsf{plus}(\mathsf{s}(0), 0) \\ \quad \to \mathsf{s}(\mathsf{plus}(0, 0)) \end{array}$$

Term Rewriting: Traditional Example

$$
\begin{array}{rcl}
\mathsf{sum}(0) & \to & 0 \\
\mathsf{sum}(\mathsf{s}(x)) & \to & \mathsf{plus}(\mathsf{s}(x), \mathsf{sum}(x)) \\
\mathsf{plus}(0, y) & \to & y \\
\mathsf{plus}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{plus}(x, y))
\end{array}
$$

$$
\begin{array}{l}
\mathsf{sum}(\mathsf{s}(0)) \\
\quad \to \mathsf{plus}(\mathsf{s}(0), \mathsf{sum}(0)) \\
\quad \to \mathsf{plus}(\mathsf{s}(0), 0) \\
\quad \to \mathsf{s}(\mathsf{plus}(0, 0)) \\
\quad \to \mathsf{s}(0)
\end{array}
$$

Term Rewriting: Traditional Example

$$\begin{aligned} \mathsf{sum}(0) &\to 0 \\ \mathsf{sum}(\mathsf{s}(x)) &\to \mathsf{plus}(\mathsf{s}(x), \mathsf{sum}(x)) \\ \mathsf{plus}(0, y) &\to y \\ \mathsf{plus}(\mathsf{s}(x), y) &\to \mathsf{s}(\mathsf{plus}(x, y)) \end{aligned}$$

Term Rewriting: Less Traditional Example

Sum Using Integers

Integers: $0, \mathsf{s}(0), \mathsf{s}(\mathsf{s}(0)), \ldots, \mathsf{p}(0), \mathsf{p}(\mathsf{p}(0)), \ldots$

Term Rewriting: Less Traditional Example

Sum Using Integers

Integers: $0, \mathsf{s}(0), \mathsf{s}(\mathsf{s}(0)), \ldots, \mathsf{p}(0), \mathsf{p}(\mathsf{p}(0)), \ldots$

$$\begin{aligned} \mathsf{sum}(x) &\to \mathsf{sumhelp}(\mathsf{geq}(0, x), x) \\ \mathsf{sumhelp}(\mathsf{true}, x) &\to 0 \\ \mathsf{sumhelp}(\mathsf{false}, \mathsf{s}(x)) &\to \mathsf{plus}(\mathsf{s}(x), \mathsf{sum}(x)) \end{aligned}$$

Term Rewriting: Less Traditional Example

Sum Using Integers

Integers: $0, \mathsf{s}(0), \mathsf{s}(\mathsf{s}(0)), \ldots, \mathsf{p}(0), \mathsf{p}(\mathsf{p}(0)), \ldots$

$$
\begin{array}{rcl}
\mathsf{sum}(x) & \to & \mathsf{sumhelp}(\mathsf{geq}(0, x), x) \\
\mathsf{sumhelp}(\mathsf{true}, x) & \to & 0 \\
\mathsf{sumhelp}(\mathsf{false}, \mathsf{s}(x)) & \to & \mathsf{plus}(\mathsf{s}(x), \mathsf{sum}(x)) \\
\mathsf{plus}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{plus}(x, y)) \\
\mathsf{plus}(\mathsf{p}(x), y) & \to & \mathsf{p}(\mathsf{plus}(x, y)) \\
\mathsf{plus}(0, y) & \to & y
\end{array}
$$

Term Rewriting: Less Traditional Example

Sum Using Integers

Integers: $0, \mathsf{s}(0), \mathsf{s}(\mathsf{s}(0)), \ldots, \mathsf{p}(0), \mathsf{p}(\mathsf{p}(0)), \ldots$

$$
\begin{array}{rcl}
\mathsf{sum}(x) & \to & \mathsf{sumhelp}(\mathsf{geq}(0, x), x) \\
\mathsf{sumhelp}(\mathsf{true}, x) & \to & 0 \\
\mathsf{sumhelp}(\mathsf{false}, \mathsf{s}(x)) & \to & \mathsf{plus}(\mathsf{s}(x), \mathsf{sum}(x)) \\
\mathsf{plus}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{plus}(x, y)) \\
\mathsf{plus}(\mathsf{p}(x), y) & \to & \mathsf{p}(\mathsf{plus}(x, y)) \\
\mathsf{plus}(0, y) & \to & y \\
\mathsf{s}(\mathsf{p}(x)) & \to & x \\
\mathsf{p}(\mathsf{s}(x)) & \to & x
\end{array}
$$

Term Rewriting: Less Traditional Example

Sum Using Integers

Integers: $0, \mathsf{s}(0), \mathsf{s}(\mathsf{s}(0)), \ldots, \mathsf{p}(0), \mathsf{p}(\mathsf{p}(0)), \ldots$

$$
\begin{array}{rcl}
\mathsf{sum}(x) & \to & \mathsf{sumhelp}(\mathsf{geq}(0, x), x) \\
\mathsf{sumhelp}(\mathsf{true}, x) & \to & 0 \\
\mathsf{sumhelp}(\mathsf{false}, \mathsf{s}(x)) & \to & \mathsf{plus}(\mathsf{s}(x), \mathsf{sum}(x)) \\
\mathsf{plus}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{plus}(x, y)) \\
\mathsf{plus}(\mathsf{p}(x), y) & \to & \mathsf{p}(\mathsf{plus}(x, y)) \\
\mathsf{plus}(0, y) & \to & y \\
\mathsf{s}(\mathsf{p}(x)) & \to & x \\
\mathsf{p}(\mathsf{s}(x)) & \to & x
\end{array}
$$

Term Rewriting: Less Traditional Example

Sum Using Integers

Integers: $0, \mathsf{s}(0), \mathsf{s}(\mathsf{s}(0)), \ldots, \mathsf{p}(0), \mathsf{p}(\mathsf{p}(0)), \ldots$

$$
\begin{aligned}
\mathsf{geq}(x, y) &\to \mathsf{geqhelp}(x, y, 0, 0) \\
\mathsf{geqhelp}(\mathsf{s}(x), y, z, a) &\to \mathsf{geqhelp}(x, y, \mathsf{s}(z), a) \\
\mathsf{geqhelp}(\mathsf{p}(x), y, z, a) &\to \mathsf{geqhelp}(x, y, z, \mathsf{s}(a)) \\
\mathsf{geqhelp}(0, \mathsf{s}(x), y, z) &\to \mathsf{geqhelp}(0, x, y, \mathsf{s}(z)) \\
\mathsf{geqhelp}(0, \mathsf{p}(x), y, z) &\to \mathsf{geqhelp}(0, x, \mathsf{s}(y), z) \\
\mathsf{geqhelp}(0, 0, \mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{geqhelp}(0, 0, x, y) \\
\mathsf{geqhelp}(0, 0, x, 0) &\to \mathsf{true} \\
\mathsf{geqhelp}(0, 0, 0, \mathsf{s}(x)) &\to \mathsf{false}
\end{aligned}
$$

Term Rewriting with Constraints

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & \mathsf{plus}(x, \mathsf{sum}(\mathsf{plus}(x, -1)) & [x > 0] \end{array}$$

Term Rewriting with Constraints

$$\begin{array}{rcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & \mathsf{plus}(x, \mathsf{sum}(\mathsf{plus}(x, -1)) & [x > 0] \\ \mathsf{plus}(x, y) & \to & z & [x + y = z] \end{array}$$

Term Rewriting with Constraints and Internal Calculation

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & x + \mathsf{sum}(x-1) & [x > 0] \end{array}$$

Modelling Imperative Programs

```
int sum(int x) {
  int i = 0, z = 0;
  for (i = 0; i <= x; i++)
    z += i;
  return z;
}
```

Modelling Imperative Programs

```
int sum(int x) {
  int i = 0, z = 0;
  for (i = 0; i <= x; i++)
    z += i;
  return z;
}
```

$$
\begin{array}{rcll}
\mathsf{sum}(x) & \to & \mathsf{u}(x,0,0) & \\
\mathsf{u}(x,i,z) & \to & \mathsf{u}(x,i+1,z+i) & [i \leq x] \\
\mathsf{u}(x,i,z) & \to & z & [\neg(i \leq x)]
\end{array}
$$

Modelling Imperative Programs

```
int fac(x) {
  if (x <= 0) return 1;
  return x * fac(x-1);
}
int facsum(int x) {
  int i = 0, z = 0;
  for (i = 0; i <= x; i++) z += fac(i);
  return z;
}
```

Modelling Imperative Programs

```
int fac(x) {
  if (x <= 0) return 1;
  return x * fac(x-1);
}
int facsum(int x) {
  int i = 0, z = 0;
  for (i = 0; i <= x; i++) z += fac(i);
  return z;
}
```

$$
\begin{array}{rcll}
\mathsf{sum}(x) & \to & \mathsf{u}(x,0,0) & \\
\mathsf{u}(x,i,z) & \to & \mathsf{u}(x,i+1,z+\mathsf{fac}(i)) & [i \leq x] \\
\mathsf{u}(x,i,z) & \to & z & [\neg(i \leq x)] \\
\mathsf{fac}(x) & \to & x * \mathit{fac}(x-1) & [\neg(x \leq 0)] \\
\mathsf{fac}(x) & \to & 1 & [x \leq 0]
\end{array}
$$

Definitions of Rewriting with Integers and/or Constraints

- integer TRSs
- $\mathbb{Z}$-TRSs
- (Nagoya) Constrained TRSs

Definitions of Rewriting with Integers and/or Constraints

- integer TRSs
- $\mathbb{Z}$-TRSs
- (Nagoya) Constrained TRSs

Integer TRSs: Idea

- infinite signature: add all integers, $\top, \bot$ and operators $+, -, *, /, \%, >, \geq, <, \leq, =, \neq, \wedge, \Rightarrow$
- infinite rules: add eg rules
  $1 + 1 \to 2$
  $6 \geq 9 \to \text{false}$
  $\text{true} \wedge \text{true} \to \text{true}$

Integer TRSs: Example

$$\begin{aligned} \mathsf{sum}(x) &\to \mathsf{sumhelp}(0 \geq x, x) \\ \mathsf{sumhelp}(\mathsf{true}, x) &\to 0 \\ \mathsf{sumhelp}(\mathsf{false}, x) &\to x + \mathsf{sum}(x - 1) \end{aligned}$$

Integer TRSs: Example

$$\begin{aligned} \mathsf{sum}(x) &\to \mathsf{sumhelp}(0 \geq x, x) \\ \mathsf{sumhelp}(\mathsf{true}, x) &\to 0 \\ \mathsf{sumhelp}(\mathsf{false}, x) &\to x + \mathsf{sum}(x - 1) \end{aligned}$$

(plus an infinite set of rules generally omitted)

Integer TRSs: Downsides

Integer TRSs: Downsides

- infinitely many rules: requires dedicated techniques

Integer TRSs: Downsides

- infinitely many rules: requires dedicated techniques
  $\Longrightarrow$ hard to extend

Integer TRSs: Downsides

- infinitely many rules: requires dedicated techniques
  $\Longrightarrow$ hard to extend
- properties like termination rely on constraints

Integer TRSs: Downsides

- infinitely many rules: requires dedicated techniques
  $\Longrightarrow$ hard to extend
- properties like termination rely on constraints
  $\Longrightarrow$ we need some way to keep track of "x gets closer to 0"

Definitions of Rewriting with Integers and/or Constraints

- integer TRSs
- $\mathbb{Z}$-TRSs
- (Nagoya) Constrained TRSs

$\mathbb{Z}$-TRSs: Ideas

- Pre-defined symbols: $0, 1 : \text{int},\ +, - : [\text{int} \times \text{int}] \Rightarrow \text{int}$
- User-given symbols with sorts (e.g. sum)
- Constraints using symbols $=, >, \geq, \neg, \wedge$, variables and int-symbols

$\mathbb{Z}$-TRSs: Example

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & x + \mathsf{sum}(x-1) & [x > 0] \end{array}$$

$\mathbb{Z}$-TRSs: Example

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & x + \mathsf{sum}(x-1) & [x > 0] \end{array}$$

Reduces for instance $\mathsf{sum}(1+1)$ to $(1+1) + ((1+1)-1) + 0$.

$\mathbb{Z}$-TRSs: Downsides

$\mathbb{Z}$-TRSs: Downsides

- formalism is fundamentally limited to the integers

$\mathbb{Z}$-TRSs: Downsides

- formalism is fundamentally limited to the integers
- encoded numbers (and equality modulo calculation) not entirely intuitive

$\mathbb{Z}$-TRSs: Downsides

- formalism is fundamentally limited to the integers
- encoded numbers (and equality modulo calculation) not entirely intuitive
- technical restrictions on reduction

$\mathbb{Z}$-TRSs: Downsides

- formalism is fundamentally limited to the integers
- encoded numbers (and equality modulo calculation) not entirely intuitive
- technical restrictions on reduction
  (e.g. for a rule $\mathsf{f}(x) \to x + 1$, a term $\mathsf{f}(\mathsf{f}(0))$ can only be reduced innermost)

Definitions of Rewriting with Integers and/or Constraints

- integer TRSs
- $\mathbb{Z}$-TRSs
- (Nagoya) Constrained TRSs

(Nagoya) Constrained TRSs: Ideas

- arbitrary underlying set (although usually integers)

(Nagoya) Constrained TRSs: Ideas

- arbitrary underlying set (although usually integers)
- values of set are encoded (e.g. $\mathsf{s}^n(0)$, $\mathsf{p}^m(0)$)

(Nagoya) Constrained TRSs: Ideas

- arbitrary underlying set (although usually integers)
- values of set are encoded (e.g. $\mathsf{s}^n(0)$, $\mathsf{p}^m(0)$)
- rules for standard functions (like addition) are also encoded

(Nagoya) Constrained TRSs: Ideas

- arbitrary underlying set (although usually integers)
- values of set are encoded (e.g. $\mathsf{s}^n(0)$, $\mathsf{p}^m(0)$)
- rules for standard functions (like addition) are also encoded
- in constraints: logical symbols and predicate symbols

(Nagoya) Constrained TRSs: Ideas

- arbitrary underlying set (although usually integers)
- values of set are encoded (e.g. $\mathsf{s}^n(0)$, $\mathsf{p}^m(0)$)
- rules for standard functions (like addition) are also encoded
- in constraints: logical symbols and predicate symbols

(Nagoya) Constrained TRSs: Example

$$\begin{array}{rcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(\mathsf{s}(x)) & \to & \mathsf{plus}(\mathsf{s}(x), \mathsf{sum}(x)) & [x \geq 0] \end{array}$$

(Nagoya) Constrained TRSs: Example

$$
\begin{array}{rcll}
\mathsf{sum}(x) & \to & 0 & [x \leq 0] \\
\mathsf{sum}(\mathsf{s}(x)) & \to & \mathsf{plus}(\mathsf{s}(x), \mathsf{sum}(x)) & [x \geq 0] \\
\mathsf{plus}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{plus}(x, y)) & \\
\mathsf{plus}(\mathsf{p}(x), y) & \to & \mathsf{p}(\mathsf{plus}(x, y)) & \\
\mathsf{plus}(0, y) & \to & y & \\
\mathsf{s}(\mathsf{p}(x)) & \to & x & \\
\mathsf{p}(\mathsf{s}(x)) & \to & x &
\end{array}
$$

(Nagoya) Constrained TRSs: Example

$$
\begin{array}{rcll}
\mathsf{sum}(x) & \to & 0 & [x \leq 0] \\
\mathsf{sum}(\mathsf{s}(x)) & \to & \mathsf{plus}(\mathsf{s}(x), \mathsf{sum}(x)) & [x \geq 0] \\
\mathsf{plus}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{plus}(x, y)) & \\
\mathsf{plus}(\mathsf{p}(x), y) & \to & \mathsf{p}(\mathsf{plus}(x, y)) & \\
\mathsf{plus}(0, y) & \to & y & \\
\mathsf{s}(\mathsf{p}(x)) & \to & x & \\
\mathsf{p}(\mathsf{s}(x)) & \to & x &
\end{array}
$$

Rewrites e.g. $\mathsf{sum}(\mathsf{s}(\mathsf{s}(0)))$ to $\mathsf{s}(\mathsf{s}(\mathsf{s}(0)))$.

(Nagoya) Constrained TRSs: Downsides

(Nagoya) Constrained TRSs: Downsides

- encoded symbols and rules not very pleasant to use

(Nagoya) Constrained TRSs: Downsides

- encoded symbols and rules not very pleasant to use (no constructor TRS!)

(Nagoya) Constrained TRSs: Downsides

- encoded symbols and rules not very pleasant to use (no constructor TRS!)
- only applicable to underlying sets which can be (finitely) encoded

(Nagoya) Constrained TRSs: Downsides

- encoded symbols and rules not very pleasant to use (no constructor TRS!)
- only applicable to underlying sets which can be (finitely) encoded
  $\Longrightarrow$ reals are going to give problems

(Nagoya) Constrained TRSs: Downsides

- encoded symbols and rules not very pleasant to use (no constructor TRS!)
- only applicable to underlying sets which can be (finitely) encoded
  $\Longrightarrow$ reals are going to give problems
- "basic" operations not basic

(Nagoya) Constrained TRSs: Downsides

- encoded symbols and rules not very pleasant to use (no constructor TRS!)
- only applicable to underlying sets which can be (finitely) encoded
  $\Longrightarrow$ reals are going to give problems
- "basic" operations not basic
  $\Longrightarrow$ impractical notion of complexity

Observations

Observations

- almost all defined only for integers
- incompatible formalisms; results do not transfer

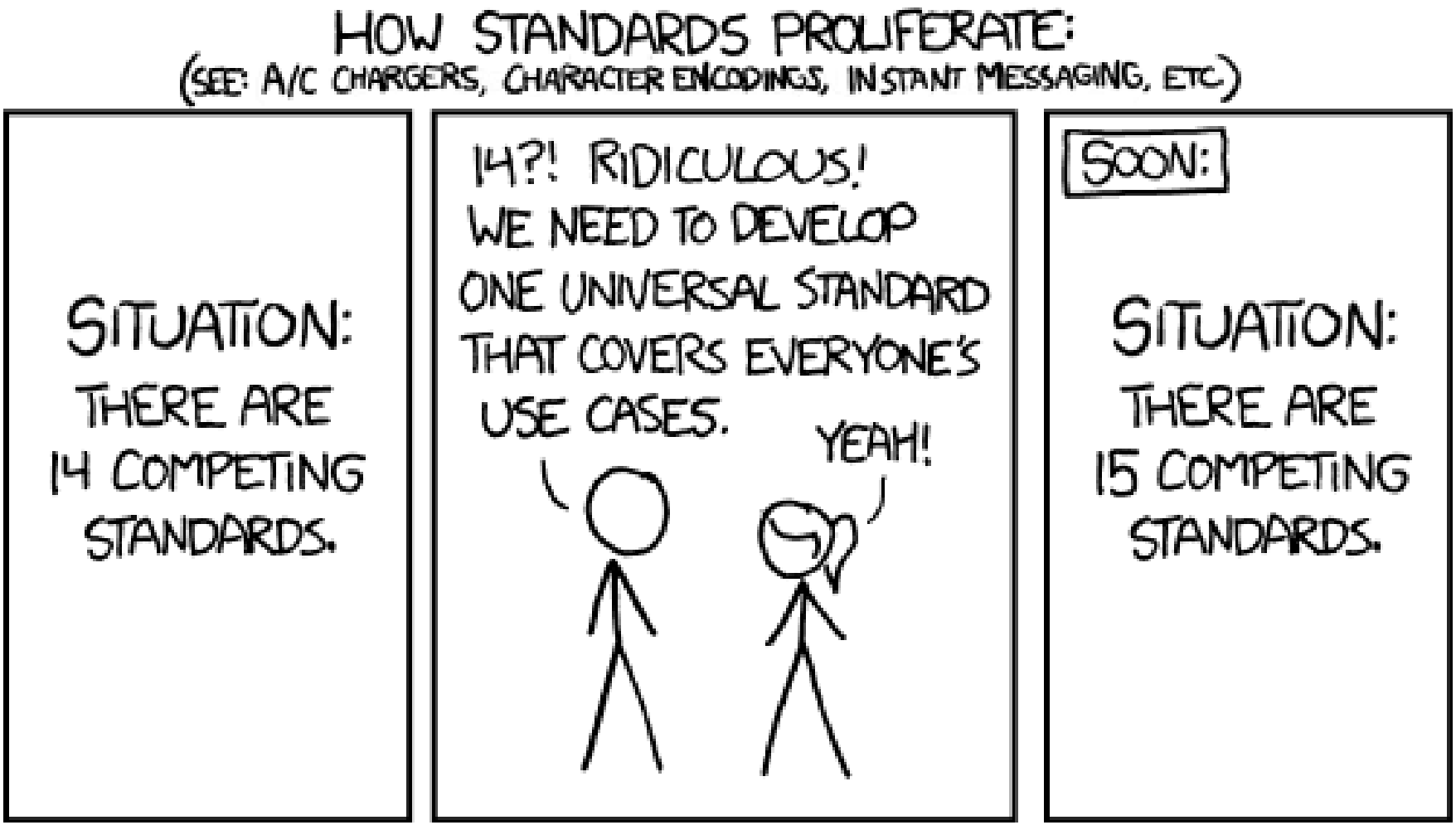
Define constrained rewriting in a natural way,

Define constrained rewriting in a natural way,which is general enough for integers and other models

Define constrained rewriting in a natural way,which is general enough for integers and other models (e.g. real numbers, bit vectors),

Define constrained rewriting in a natural way,which is general enough for integers and other models (e.g. real numbers, bit vectors), powerful enough to do at least all we can do with the existing formalisms

Define constrained rewriting in a natural way,which is general enough for integers and other models (e.g. real numbers, bit vectors), powerful enough to do at least all we can do with the existing formalisms and sufficiently flexible that we can still obtain useful results.



Source: http://www.xkcd.com

Symbols and terms

Symbols and terms

- function symbols with sort declaration

Symbols and terms

- function symbols with sort declaration
  e.g. $0, 1, \ldots : \text{int},\ +\ : [\text{int} \times \text{int}] \Rightarrow \text{int},\ \text{sum} : [\text{int}] \Rightarrow \text{int},$
  $>: [\text{int} \times \text{int}] \Rightarrow \text{bool}$

Symbols and terms

- function symbols with sort declaration
  e.g. $0, 1, \ldots : \text{int},\ +\ : [\text{int} \times \text{int}] \Rightarrow \text{int},\ \text{sum} : [\text{int}] \Rightarrow \text{int},$
  $>: [\text{int} \times \text{int}] \Rightarrow \text{bool}$
- theory symbols and sorts have a meaning in some fixed model

Symbols and terms

- function symbols with sort declaration
  e.g. $0, 1, \ldots : \text{int},\ +: [\text{int} \times \text{int}] \Rightarrow \text{int},\ \text{sum} : [\text{int}] \Rightarrow \text{int},$
  $>: [\text{int} \times \text{int}] \Rightarrow \text{bool}$
- theory symbols and sorts have a meaning in some fixed model
  e.g. int corresponds to $\mathbb{Z}$, $0, 1, \ldots, +$ are interpreted as usual

Symbols and terms

- function symbols with sort declaration
  e.g. $0, 1, \ldots : \text{int},\ +: [\text{int} \times \text{int}] \Rightarrow \text{int},\ \text{sum} : [\text{int}] \Rightarrow \text{int},$
  $>: [\text{int} \times \text{int}] \Rightarrow \text{bool}$
- theory symbols and sorts have a meaning in some fixed model
  e.g. int corresponds to $\mathbb{Z}$, $0, 1, \ldots, +$ are interpreted as usual
  (sum defines a function, and is not a theory symbol)

Symbols and terms

- function symbols with sort declaration
  e.g. $0, 1, \ldots : \text{int},\ +\ : [\text{int} \times \text{int}] \Rightarrow \text{int},\ \text{sum} : [\text{int}] \Rightarrow \text{int},$
  $>: [\text{int} \times \text{int}] \Rightarrow \text{bool}$
- theory symbols and sorts have a meaning in some fixed model
  e.g. int corresponds to $\mathbb{Z}$, $0, 1, \ldots, +$ are interpreted as usual
  (sum defines a function, and is not a theory symbol)
- values represent the elements of the underlying sets

Symbols and terms

- function symbols with sort declaration
  e.g. $0, 1, \ldots : \text{int},\ +\ : [\text{int} \times \text{int}] \Rightarrow \text{int},\ \text{sum} : [\text{int}] \Rightarrow \text{int},$
  $>: [\text{int} \times \text{int}] \Rightarrow \text{bool}$
- theory symbols and sorts have a meaning in some fixed model
  e.g. int corresponds to $\mathbb{Z}$, $0, 1, \ldots, +$ are interpreted as usual
  (sum defines a function, and is not a theory symbol)
- values represent the elements of the underlying sets
  e.g. $0, 1, 2, \ldots, \text{true}, \text{false}$

Symbols and terms

- function symbols with sort declaration
  e.g. $0, 1, \ldots : \text{int},\ +\ : [\text{int} \times \text{int}] \Rightarrow \text{int},\ \text{sum} : [\text{int}] \Rightarrow \text{int},$
  $>: [\text{int} \times \text{int}] \Rightarrow \text{bool}$
- theory symbols and sorts have a meaning in some fixed model
  e.g. int corresponds to $\mathbb{Z}$, $0, 1, \ldots, +$ are interpreted as usual
  (sum defines a function, and is not a theory symbol)
- values represent the elements of the underlying sets
  e.g. $0, 1, 2, \ldots,$ true, false
- terms are formed from function symbols and sorted variables
  in a sort-respecting way

Symbols and terms

- function symbols with sort declaration
  e.g. $0, 1, \ldots : \text{int},\ +\ : [\text{int} \times \text{int}] \Rightarrow \text{int},\ \text{sum} : [\text{int}] \Rightarrow \text{int},$
  $>: [\text{int} \times \text{int}] \Rightarrow \text{bool}$
- theory symbols and sorts have a meaning in some fixed model
  e.g. int corresponds to $\mathbb{Z}$, $0, 1, \ldots, +$ are interpreted as usual
  (sum defines a function, and is not a theory symbol)
- values represent the elements of the underlying sets
  e.g. $0, 1, 2, \ldots,$ true, false
- terms are formed from function symbols and sorted variables
  in a sort-respecting way
  e.g. $\text{sum}(0, 1)$ and $\text{concat}(\text{"}\textit{hello}\text{"}, \text{"}\textit{world}\text{"})$, but not
  $\text{concat}(\text{"}\textit{hello}\text{"}, 3)$

Rules and rewriting

- constraints are terms built of theory symbols, with sort bool

Rules and rewriting

- constraints are terms built of theory symbols, with sort bool
  e.g. $3 > x$, $\mathit{isprime}(17 + 8)$

Rules and rewriting

- constraints are terms built of theory symbols, with sort bool
  e.g. $3 > x$, $\mathit{isprime}(17 + 8)$
- rules are triples $l \to r\ [\varphi]$ with l, r terms and φ a constraint

Rules and rewriting

- constraints are terms built of theory symbols, with sort bool
  e.g. $3 > x$, $\mathit{isprime}(17 + 8)$
- rules are triples $l \to r\ [\varphi]$ with l, r terms and φ a constraint
- to rewrite, instantiate variables in the rules by terms

Rules and rewriting

- constraints are terms built of theory symbols, with sort bool
  e.g. $3 > x$, $\mathit{isprime}(17 + 8)$
- rules are triples $l \to r\ [\varphi]$ with l, r terms and φ a constraint
- to rewrite, instantiate variables in the rules by terms
  variables in the constraint must be instantiated by values

Rules and rewriting

- constraints are terms built of theory symbols, with sort bool
  e.g. $3 > x$, $\mathit{isprime}(17 + 8)$
- rules are triples $l \to r\ [\varphi]$ with l, r terms and φ a constraint
- to rewrite, instantiate variables in the rules by terms
  variables in the constraint must be instantiated by values
  instantiated constraints must be valid

Rules and rewriting

- constraints are terms built of theory symbols, with sort bool
  e.g. $3 > x$, $\mathit{isprime}(17 + 8)$
- rules are triples $l \to r\ [\varphi]$ with l, r terms and φ a constraint
- to rewrite, instantiate variables in the rules by terms
  variables in the constraint must be instantiated by values
  instantiated constraints must be valid
- theory symbols applied on values can be reduced to their value

Rules and rewriting

- constraints are terms built of theory symbols, with sort bool
  e.g. $3 > x$, $\mathit{isprime}(17 + 8)$
- rules are triples $l \to r\ [\varphi]$ with l, r terms and φ a constraint
- to rewrite, instantiate variables in the rules by terms
  variables in the constraint must be instantiated by values
  instantiated constraints must be valid
- theory symbols applied on values can be reduced to their value
  e.g. $3 + 5 \to 8$ and $\mathit{isprime}(17 + 8) \to \mathit{isprime}(25) \to$ false

Reducing sum

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & x + \mathsf{sum}(x-1) & [x \geq 0] \end{array}$$

Reducing sum

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & x + \mathsf{sum}(x-1) & [x \geq 0] \end{array}$$

$$\mathsf{sum}(2)$$

Reducing sum

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & x + \mathsf{sum}(x-1) & [x \geq 0] \end{array}$$

$$\begin{array}{rl} & \mathsf{sum}(2) \\ \to & 2 + \mathsf{sum}(2-1) \end{array}$$

Reducing sum

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & x + \mathsf{sum}(x-1) & [x \geq 0] \end{array}$$

$$\begin{array}{lll} & \mathsf{sum}(2) & \\ \to & 2 + \mathsf{sum}(2-1) & X \end{array}$$

Reducing sum

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & x + \mathsf{sum}(x-1) & [x \geq 0] \end{array}$$

$$\begin{array}{rl} & \mathsf{sum}(2) \\ \to & 2 + \mathsf{sum}(2-1) \end{array}$$

Reducing sum

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & x + \mathsf{sum}(x-1) & [x \geq 0] \end{array}$$

$$\begin{array}{rl} & \mathsf{sum}(2) \\ \to & 2 + \mathsf{sum}(2-1) \\ \to & 2 + \mathsf{sum}(1) \end{array}$$

Reducing sum

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & x + \mathsf{sum}(x-1) & [x \geq 0] \end{array}$$

$$\begin{array}{ll} & \mathsf{sum}(2) \\ \to & 2 + \mathsf{sum}(2-1) \\ \to & 2 + \mathsf{sum}(1) \\ \to & 2 + (1 + \mathsf{sum}(1-1)) \end{array}$$

Reducing sum

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & x + \mathsf{sum}(x-1) & [x \geq 0] \end{array}$$

$$\begin{array}{ll} & \mathsf{sum}(2) \\ \to & 2 + \mathsf{sum}(2-1) \\ \to & 2 + \mathsf{sum}(1) \\ \to & 2 + (1 + \mathsf{sum}(1-1)) \\ \to & 2 + (1 + \mathsf{sum}(0)) \end{array}$$

Reducing sum

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & x + \mathsf{sum}(x-1) & [x \geq 0] \end{array}$$

$$\begin{array}{ll} & \mathsf{sum}(2) \\ \to & 2 + \mathsf{sum}(2-1) \\ \to & 2 + \mathsf{sum}(1) \\ \to & 2 + (1 + \mathsf{sum}(1-1)) \\ \to & 2 + (1 + \mathsf{sum}(0)) \\ \to & 2 + (1 + 0) \end{array}$$

Reducing sum

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & x + \mathsf{sum}(x-1) & [x \geq 0] \end{array}$$

$$\begin{array}{ll} & \mathsf{sum}(2) \\ \to & 2 + \mathsf{sum}(2-1) \\ \to & 2 + \mathsf{sum}(1) \\ \to & 2 + (1 + \mathsf{sum}(1-1)) \\ \to & 2 + (1 + \mathsf{sum}(0)) \\ \to & 2 + (1 + 0) \\ \to & 2 + 1 \end{array}$$

Reducing sum

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & x + \mathsf{sum}(x-1) & [x \geq 0] \end{array}$$

$$\begin{array}{ll} & \mathsf{sum}(2) \\ \to & 2 + \mathsf{sum}(2-1) \\ \to & 2 + \mathsf{sum}(1) \\ \to & 2 + (1 + \mathsf{sum}(1-1)) \\ \to & 2 + (1 + \mathsf{sum}(0)) \\ \to & 2 + (1 + 0) \\ \to & 2 + 1 \\ \to & 3 \end{array}$$

(Naive) Ackerman Function

$$\begin{array}{rcll} \mathsf{ack}(m,n) & \to & \mathsf{ack}(m-1,\mathsf{ack}(m,n-1)) & [m \neq 0 \wedge n \neq 0] \\ \mathsf{ack}(m,0) & \to & \mathsf{ack}(m-1,1) & [m \neq 0] \\ \mathsf{ack}(0,n) & \to & n+1 & \end{array}$$

(Naive) Ackerman Function

$$\begin{array}{rcll} \mathsf{ack}(m,n) & \to & \mathsf{ack}(m-1,\mathsf{ack}(m,n-1)) & [m \neq 0 \wedge n \neq 0] \\ \mathsf{ack}(m,0) & \to & \mathsf{ack}(m-1,1) & [m \neq 0] \\ \mathsf{ack}(0,n) & \to & n+1 & \end{array}$$

With $1, -, +$ interpreted over 16-bit bitvectors.

Bubblesort

```
swap(arr, i, j) {
  int tmp = arr[i];
  arr[i] = arr[j];
  arr[j] = tmp;

}

bubblesort(arr, n) {
  for (int i = 0; i < n; i++)
    for (int j = i+1; j < n; j++)
      if (arr[i] > arr[j])  swap(arr,i,j);

}
```

Bubblesort

```
swap(arr, i, j) {
  int tmp = arr[i];
  arr = arr[i ↦ arr[j]]
  arr = arr[j ↦ tmp];
  return arr;
}

bubblesort(arr, n) {
  for (int i = 0; i < n; i++)
    for (int j = i+1; j < n; j++)
      if (arr[i] > arr[j]) arr = swap(arr,i,j);
  return arr;
}
```

Bubblesort

Sorts: IntArray and Int

$$
\begin{array}{rcll}
\mathsf{bubblesort}(arr, n) & \to & \mathsf{loop1}(arr, n, 0) & \\
\mathsf{loop1}(arr, n, i) & \to & \mathsf{loop2}(arr, n, i, 0) & [i < n] \\
\mathsf{loop1}(arr, n, i) & \to & arr & [i \geq n] \\
\mathsf{loop2}(arr, n, i, j) & \to & \mathsf{loop2}(arr, n, i, j + 1) & \\
 & & [j < n \wedge get(arr, i) \leq get(arr, j)] & \\
\mathsf{loop2}(arr, n, i, j) & \to & \mathsf{loop2}(\mathsf{swap}(arr, i, j), n, i, j + 1) & \\
 & & [j < n \wedge get(arr, i) > get(arr, j)] & \\
\mathsf{loop2}(arr, n, i, j) & \to & \mathsf{loop1}(arr, n, i + 1) & [j \geq n] \\
\mathsf{swap}(arr, i, j) & \to & set(set(arr, i, get(arr, j)), j, get(arr, i)) &
\end{array}
$$

Bubblesort

Sorts: IntArray and Int

$$
\begin{array}{rcll}
\mathsf{bubblesort}(arr, n) & \to & \mathsf{loop1}(arr, n, 0) & \\
\mathsf{loop1}(arr, n, i) & \to & \mathsf{loop2}(arr, n, i, 0) & [i < n] \\
\mathsf{loop1}(arr, n, i) & \to & arr & [i \geq n] \\
\mathsf{loop2}(arr, n, i, j) & \to & \mathsf{loop2}(arr, n, i, j + 1) & \\
 & & [j < n \wedge \mathit{get}(arr, i) \leq \mathit{get}(arr, j)] & \\
\mathsf{loop2}(arr, n, i, j) & \to & \mathsf{loop2}(\mathsf{swap}(arr, i, j), n, i, j + 1) & \\
 & & [j < n \wedge \mathit{get}(arr, i) > \mathit{get}(arr, j)] & \\
\mathsf{loop2}(arr, n, i, j) & \to & \mathsf{loop1}(arr, n, i + 1) & [j \geq n] \\
\mathsf{swap}(arr, i, j) & \to & \mathit{set}(\mathit{set}(arr, i, \mathit{get}(arr, j)), j, \mathit{get}(arr, i)) &
\end{array}
$$

Terms With Constraints

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & x + \mathsf{sum}(x-1) & [x \geq 0] \end{array}$$

Terms With Constraints

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & x + \mathsf{sum}(x-1) & [x \geq 0] \end{array}$$

$\mathsf{sum}(x)\ [x \geq 2]$

Terms With Constraints

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & x + \mathsf{sum}(x-1) & [x \geq 0] \end{array}$$

$$\begin{array}{lll} & \mathsf{sum}(x)\ [x \geq 2] \\ \to & x + \mathsf{sum}(x-1)\ [x \geq 2] \end{array}$$

Terms With Constraints

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & x + \mathsf{sum}(x-1) & [x \geq 0] \end{array}$$

$$\begin{array}{lll} & \mathsf{sum}(x)\ [x \geq 2] & \\ \to & x + \mathsf{sum}(x-1)\ [x \geq 2] & X \end{array}$$

Terms With Constraints

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & x + \mathsf{sum}(x-1) & [x \geq 0] \end{array}$$

$$\begin{array}{lll} & \mathsf{sum}(x)\ [x \geq 2] \\ \to & x + \mathsf{sum}(x-1)\ [x \geq 2] \end{array}$$

Terms With Constraints

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & x + \mathsf{sum}(x-1) & [x \geq 0] \end{array}$$

$$\begin{array}{ll} & \mathsf{sum}(x)\ [x \geq 2] \\ \to & x + \mathsf{sum}(x-1)\ [x \geq 2] \\ \to & x + \mathsf{sum}(y)\ [x \geq 2 \wedge y = x - 1] \end{array}$$

Terms With Constraints

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & x + \mathsf{sum}(x-1) & [x \geq 0] \end{array}$$

$$\begin{array}{ll} & \mathsf{sum}(x)\ [x \geq 2] \\ \to & x + \mathsf{sum}(x-1)\ [x \geq 2] \\ \to & x + \mathsf{sum}(y)\ [x \geq 2 \wedge y = x-1] \\ \to & x + (y + \mathsf{sum}(y-1))\ [x \geq 2 \wedge y = x-1] \end{array}$$

Terms With Constraints

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & x + \mathsf{sum}(x-1) & [x \geq 0] \end{array}$$

$$\begin{array}{ll} & \mathsf{sum}(x)\ [x \geq 2] \\ \to & x + \mathsf{sum}(x-1)\ [x \geq 2] \\ \to & x + \mathsf{sum}(y)\ [x \geq 2 \wedge y = x-1] \\ \to & x + (y + \mathsf{sum}(y-1))\ [x \geq 2 \wedge y = x-1] \\ \to & x + (y + \mathsf{sum}(z))\ [x \geq 2 \wedge y = x-1 \wedge z = y-1] \end{array}$$

Terms With Constraints

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & x + \mathsf{sum}(x-1) & [x \geq 0] \end{array}$$

$$\begin{array}{lll} & \mathsf{sum}(x)\ [x \geq 2] & \\ \to & x + \mathsf{sum}(x-1)\ [x \geq 2] & \\ \to & x + \mathsf{sum}(y)\ [x \geq 2 \wedge y = x-1] & \\ \to & x + (y + \mathsf{sum}(y-1))\ [x \geq 2 \wedge y = x-1] & \\ \to & x + (y + \mathsf{sum}(z))\ [x \geq 2 \wedge y = x-1 \wedge z = y-1] & \\ \approx & x + (y + \mathsf{sum}(z))\ [x = z+2 \wedge y = z+1 \wedge z \geq 0] & \end{array}$$

Terms With Constraints

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & x + \mathsf{sum}(x-1) & [x \geq 0] \end{array}$$

$$\begin{array}{ll} & \mathsf{sum}(x)\ [x \geq 2] \\ \to & x + \mathsf{sum}(x-1)\ [x \geq 2] \\ \to & x + \mathsf{sum}(y)\ [x \geq 2 \wedge y = x-1] \\ \to & x + (y + \mathsf{sum}(y-1))\ [x \geq 2 \wedge y = x-1] \\ \to & x + (y + \mathsf{sum}(z))\ [x \geq 2 \wedge y = x-1 \wedge z = y-1] \\ \approx & x + (y + \mathsf{sum}(z))\ [x = z+2 \wedge y = z+1 \wedge z \geq 0] \end{array}$$

$$\mathsf{sum}(x-1)\ [\mathsf{true}]$$

Terms With Constraints

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & x + \mathsf{sum}(x-1) & [x \geq 0] \end{array}$$

$$\begin{array}{ll} & \mathsf{sum}(x)\ [x \geq 2] \\ \to & x + \mathsf{sum}(x-1)\ [x \geq 2] \\ \to & x + \mathsf{sum}(y)\ [x \geq 2 \wedge y = x-1] \\ \to & x + (y + \mathsf{sum}(y-1))\ [x \geq 2 \wedge y = x-1] \\ \to & x + (y + \mathsf{sum}(z))\ [x \geq 2 \wedge y = x-1 \wedge z = y-1] \\ \approx & x + (y + \mathsf{sum}(z))\ [x = z+2 \wedge y = z+1 \wedge z \geq 0] \end{array}$$

$$\mathsf{sum}(x-1)\ [\mathsf{true}]$$

Terms With Constraints

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & x + \mathsf{sum}(x-1) & [x \geq 0] \end{array}$$

$$\begin{array}{ll} & \mathsf{sum}(x)\ [x \geq 2] \\ \to & x + \mathsf{sum}(x-1)\ [x \geq 2] \\ \to & x + \mathsf{sum}(y)\ [x \geq 2 \wedge y = x-1] \\ \to & x + (y + \mathsf{sum}(y-1))\ [x \geq 2 \wedge y = x-1] \\ \to & x + (y + \mathsf{sum}(z))\ [x \geq 2 \wedge y = x-1 \wedge z = y-1] \\ \approx & x + (y + \mathsf{sum}(z))\ [x = z+2 \wedge y = z+1 \wedge z \geq 0] \end{array}$$

$\mathsf{sum}(x-1)\ [\mathsf{true}]$ ✗

Terms With Constraints

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & x + \mathsf{sum}(x-1) & [x \geq 0] \end{array}$$

$$\begin{array}{lll} & \mathsf{sum}(x) & [x \geq 2] \\ \to & x + \mathsf{sum}(x-1) & [x \geq 2] \\ \to & x + \mathsf{sum}(y) & [x \geq 2 \wedge y = x-1] \\ \to & x + (y + \mathsf{sum}(y-1)) & [x \geq 2 \wedge y = x-1] \\ \to & x + (y + \mathsf{sum}(z)) & [x \geq 2 \wedge y = x-1 \wedge z = y-1] \\ \approx & x + (y + \mathsf{sum}(z)) & [x = z+2 \wedge y = z+1 \wedge z \geq 0] \end{array}$$

$$\mathsf{sum}(\mathsf{sum}(3) - 1) \; [\mathsf{true}]$$

Goals

Natural:

Goals

Natural:

- sorts with natural meanings
- no encodings: we include values of underlying set

Goals

Natural:

- sorts with natural meanings
- no encodings: we include values of underlying set

General:

Goals

Natural:

- sorts with natural meanings
- no encodings: we include values of underlying set

General:

- we can use any underlying theory (or theories)
- results work for whichever predicates and functions we use

Goals

Natural:

- sorts with natural meanings
- no encodings: we include values of underlying set

General:

- we can use any underlying theory (or theories)
- results work for whichever predicates and functions we use

Powerful:

Goals

Natural:

- sorts with natural meanings
- no encodings: we include values of underlying set

General:

- we can use any underlying theory (or theories)
- results work for whichever predicates and functions we use

Powerful:

- can (mostly) encode or simulate existing formalisms
- all handle typical existing applications (often more easily)

Goals

Natural:

- sorts with natural meanings
- no encodings: we include values of underlying set

General:

- we can use any underlying theory (or theories)
- results work for whichever predicates and functions we use

Powerful:

- can (mostly) encode or simulate existing formalisms
- all handle typical existing applications (often more easily)

Flexible:

Goals

Natural:

- sorts with natural meanings
- no encodings: we include values of underlying set

General:

- we can use any underlying theory (or theories)
- results work for whichever predicates and functions we use

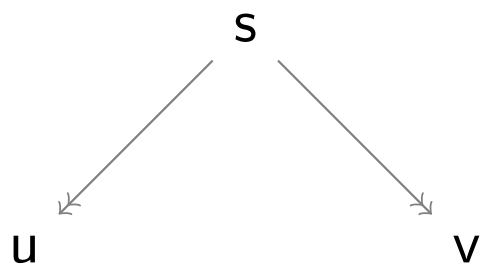
Powerful:

- can (mostly) encode or simulate existing formalisms
- all handle typical existing applications (often more easily)

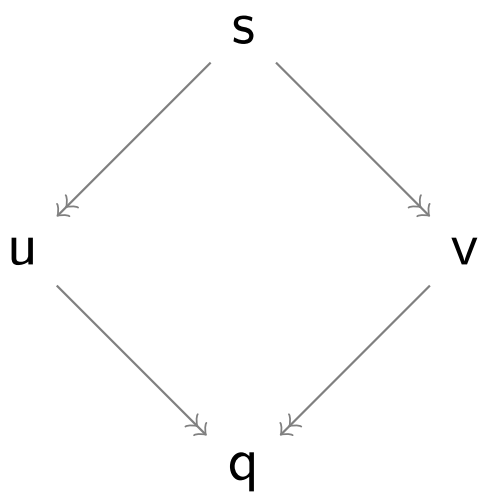
Flexible:

- getting there!

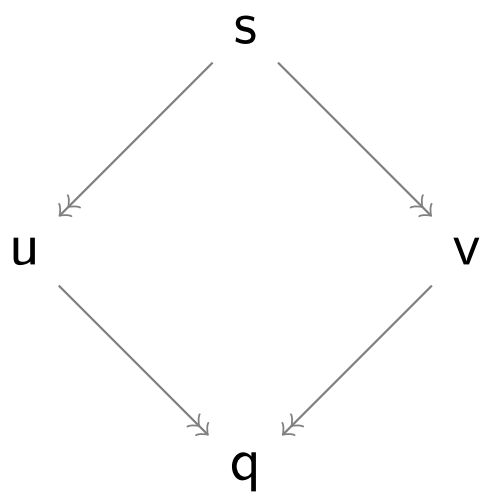
Confluence



Confluence

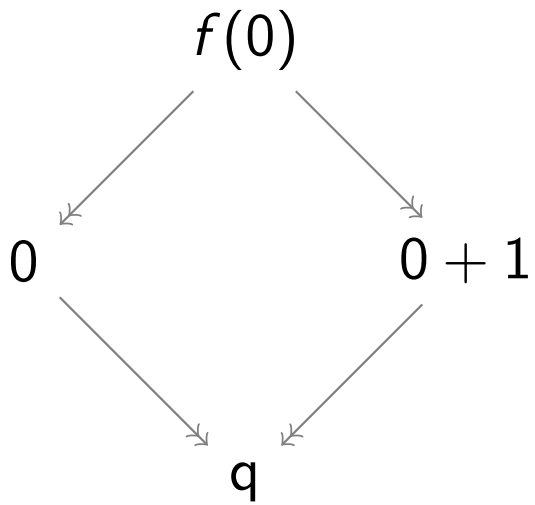


Confluence



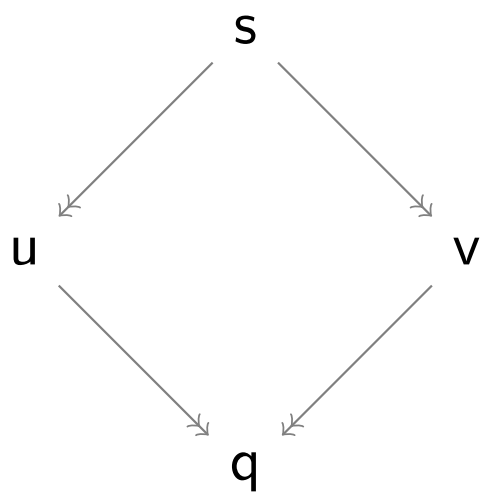
- $f(x) \to 0\ [x \leq 0],\ f(x) \to x + 1\ [x \geq 0]$ is not confluent

Confluence



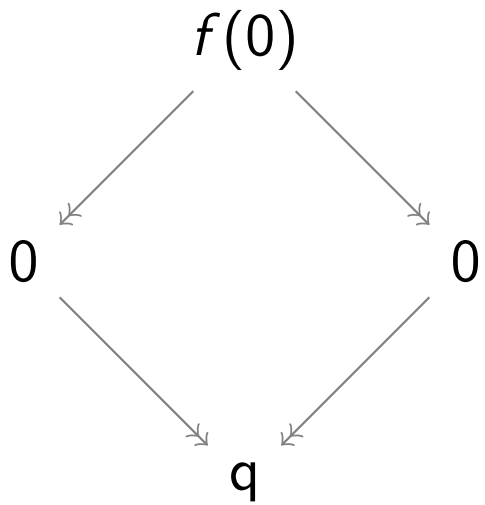
- $f(x) \to 0\ [x \leq 0],\ f(x) \to x+1\ [x \geq 0]$ is not confluent

Confluence



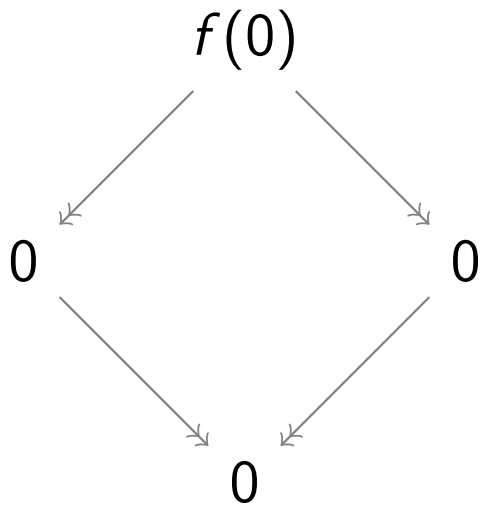
- $f(x) \to 0\ [x \leq 0],\ f(x) \to x + 1\ [x \geq 0]$ is not confluent
- $f(x) \to 0\ [x \leq 0],\ f(x) \to x\ [x \geq 0]$ is

Confluence



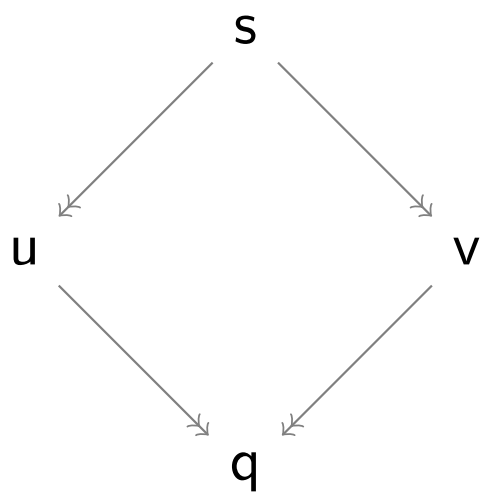
- $f(x) \to 0\ [x \leq 0],\ f(x) \to x + 1\ [x \geq 0]$ is not confluent
- $f(x) \to 0\ [x \leq 0],\ f(x) \to x\ [x \geq 0]$ is confluent

Confluence



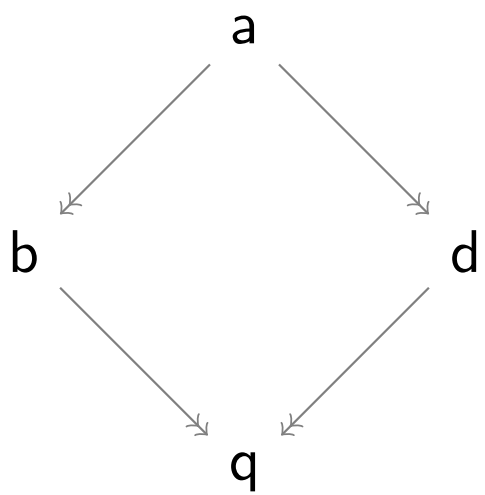
- $f(x) \to 0\ [x \leq 0],\ f(x) \to x + 1\ [x \geq 0]$ is not confluent
- $f(x) \to 0\ [x \leq 0],\ f(x) \to x\ [x \geq 0]$ is confluent

Confluence



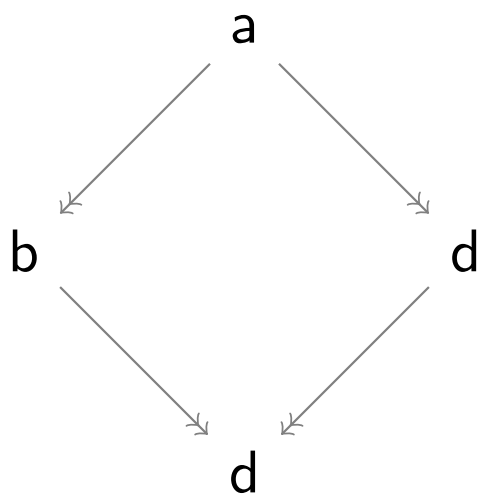
- $f(x) \to 0\ [x \leq 0],\ f(x) \to x + 1\ [x \geq 0]$ is not confluent
- $f(x) \to 0\ [x \leq 0],\ f(x) \to x\ [x \geq 0]$ is confluent
- $a \to b,\ b \to a,\ a \to c,\ c \to d$ is

Confluence



- $f(x) \to 0\ [x \leq 0],\ f(x) \to x + 1\ [x \geq 0]$ is not confluent
- $f(x) \to 0\ [x \leq 0],\ f(x) \to x\ [x \geq 0]$ is confluent
- $a \to b,\ b \to a,\ a \to c,\ c \to d$ is

Confluence



- $f(x) \to 0\ [x \leq 0],\ f(x) \to x + 1\ [x \geq 0]$ is not confluent
- $f(x) \to 0\ [x \leq 0],\ f(x) \to x\ [x \geq 0]$ is confluent
- $a \to b,\ b \to a,\ a \to c,\ c \to d$ is confluent

Orthogonality

Sufficient Conditions for Confluence:

Orthogonality

Sufficient Conditions for Confluence:

- left-linearity

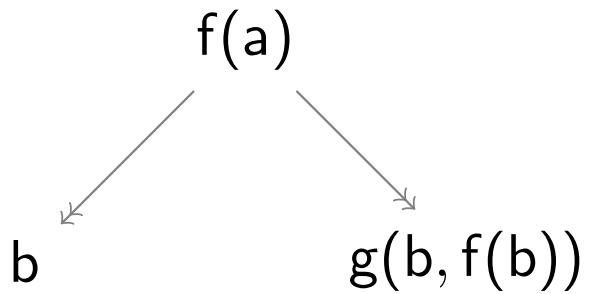
Orthogonality

Sufficient Conditions for Confluence:

- left-linearity

[Klop80]

$$\begin{aligned} a &\to f(a) \\ f(x) &\to g(x, f(x)) \\ g(x, x) &\to b \end{aligned}$$



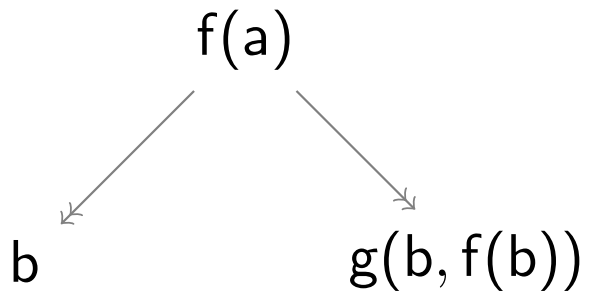
Orthogonality

Sufficient Conditions for Confluence:

- left-linearity

[Klop80]

$$\begin{aligned} a &\to f(a) \\ f(x) &\to g(x, f(x)) \\ g(x, x) &\to b \end{aligned}$$



Orthogonality

Sufficient Conditions for Confluence:

- left-linearity
- non-overlapping rules

Orthogonality

Sufficient Conditions for Confluence:

- left-linearity
- non-overlapping rules

$$\begin{array}{lcll} \mathsf{f}(x) & \to & 0 & [x \leq 0] \\ \mathsf{f}(x) & \to & x+1 & [x \geq 0] \end{array}$$

Orthogonality

Sufficient Conditions for Confluence:

- left-linearity
- non-overlapping rules

$$\begin{array}{lcll} \mathsf{f}(x) & \to & r_1 & [x \leq 0] \\ \mathsf{f}(x) & \to & r_2 & [x \geq 0] \end{array}$$

Orthogonality

Sufficient Conditions for Confluence:

- left-linearity
- non-overlapping rules

$$\begin{array}{lcll} f(x) & \to & r_1 & [x \leq 0] \\ f(x) & \to & r_2 & [x \geq 0] \end{array}$$

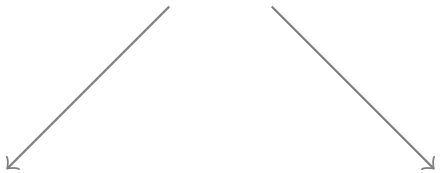
Orthogonality

Sufficient Conditions for Confluence:

- left-linearity
- non-overlapping rules

$$\begin{array}{lcll} \mathsf{f}(x) & \to & r_1 & [x \leq 0] \\ \mathsf{f}(x) & \to & r_2 & [x \geq 0] \end{array}$$

$$\mathsf{f}(x)\ [x \leq 0 \wedge x \geq 0]$$



$$r_1\ [x \leq 0 \wedge x \geq 0] \qquad r_2\ [x \leq 0 \wedge x \geq 0]$$

Orthogonality

Sufficient Conditions for Confluence:

- left-linearity
- non-overlapping rules

$$\mathsf{f}(\mathsf{f}(x)) \quad \to \quad r \quad [\text{true}]$$

Orthogonality

Sufficient Conditions for Confluence:

- left-linearity
- non-overlapping rules

$$\mathsf{f}(\mathsf{f}(x)) \quad \to \quad r \quad [\text{true}]$$

$$\mathsf{f}(\mathsf{f}(\mathsf{f}(y)))$$

Orthogonality

Sufficient Conditions for Confluence:

- left-linearity
- non-overlapping rules

$$\mathsf{f}(\mathsf{f}(x)) \quad \to \quad r \quad [\text{true}]$$

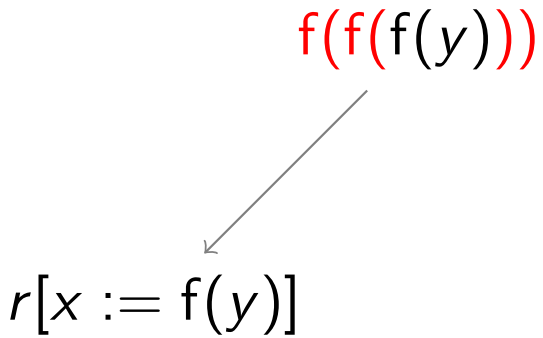
$$\mathsf{f}(\mathsf{f}(\mathsf{f}(y)))$$

Orthogonality

Sufficient Conditions for Confluence:

- left-linearity
- non-overlapping rules

$$\mathsf{f}(\mathsf{f}(x)) \quad \to \quad r \quad [\text{true}]$$

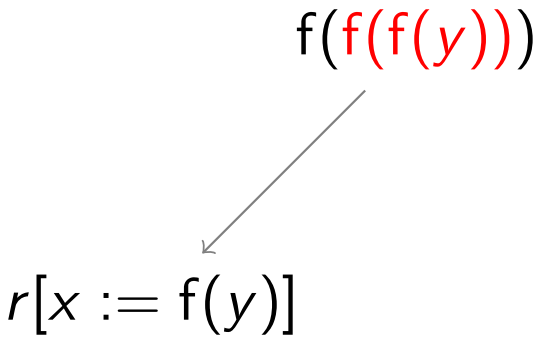


Orthogonality

Sufficient Conditions for Confluence:

- left-linearity
- non-overlapping rules

$$\mathsf{f}(\mathsf{f}(x)) \quad \to \quad r \quad [\text{true}]$$

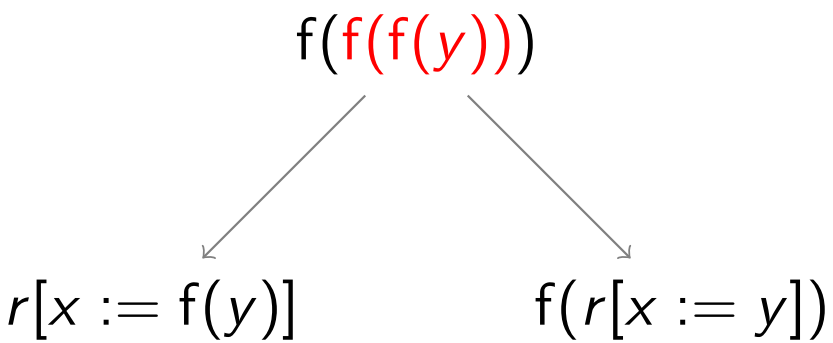


Orthogonality

Sufficient Conditions for Confluence:

- left-linearity
- non-overlapping rules

$$\mathsf{f}(\mathsf{f}(x)) \quad \to \quad r \quad [\text{true}]$$



Orthogonality

Sufficient Conditions for Confluence:

- left-linearity
- non-overlapping rules

$$\mathsf{f}(\mathsf{f}(x)) \quad \to \quad r \quad [x = x]$$

Orthogonality

Sufficient Conditions for Confluence:

- left-linearity
- non-overlapping rules

$$\mathsf{f}(\mathsf{f}(x)) \quad \to \quad r \quad [x = x]$$

$$\mathsf{f}(\mathsf{f}(\mathsf{f}(y)))$$

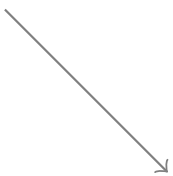
Orthogonality

Sufficient Conditions for Confluence:

- left-linearity
- non-overlapping rules

$$\mathsf{f}(\mathsf{f}(x)) \quad \to \quad r \quad [x = x]$$

$$\mathsf{f}(\mathsf{f}(\mathsf{f}(y)))$$



$$\mathsf{f}(r[x := y])\ [y = y]$$

Orthogonality

Sufficient Conditions for Confluence:

- left-linearity
- non-overlapping rules

$$\mathsf{f}(\mathsf{f}(x)) \quad \to \quad r \quad [x = x]$$

$$\mathsf{f}(\mathsf{f}(\mathsf{f}(y)))$$

$$\mathsf{f}(r[x := y])\ [y = y]$$

Orthogonality

Sufficient Conditions for Confluence:

- left-linearity
- non-overlapping rules

$$\mathsf{f}(\mathsf{f}(x)) \quad \to \quad r \quad [x = x]$$

$$\mathsf{f}(\mathsf{f}(\mathsf{f}(y)))$$

$$\mathsf{f}(r[x := y])\ [y = y]$$

Orthogonality Checking

Orthogonality Checking

- check for overlaps (standard)

Orthogonality Checking

- check for overlaps (standard)
- in overlapping rules with constraints φ, ψ, test satisfiability of $\varphi \wedge \psi$

Orthogonality Checking

- check for overlaps (standard)
- in overlapping rules with constraints φ, ψ, test satisfiability of $\varphi \wedge \psi$

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & x + \mathsf{sum}(x-1) & [x > 0] \end{array}$$

Orthogonality Checking

- check for overlaps (standard)
- in overlapping rules with constraints φ, ψ, test satisfiability of $\varphi \wedge \psi$

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & x + \mathsf{sum}(x-1) & [x > 0] \end{array}$$

$x \leq 0 \wedge x > 0$ is not satisfiable ✓

Orthogonality Checking

- check for overlaps (standard)
- in overlapping rules with constraints φ, ψ, test satisfiability of $\varphi \wedge \psi$

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & x + \mathsf{sum}(x-1) & [x \geq 0] \end{array}$$

$x \leq 0 \wedge x > 0$ is not satisfiable ✓

Orthogonality Checking

- check for overlaps (standard)
- in overlapping rules with constraints φ, ψ, test satisfiability of $\varphi \wedge \psi$

$$\begin{array}{lcll} \mathsf{sum}(x) & \to & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \to & x + \mathsf{sum}(x-1) & [x \geq 0] \end{array}$$

$x \leq 0 \wedge x > 0$ is not satisfiable ✓

$x \leq 0 \wedge x \geq 0$ is satisfiable X

Termination

Termination

$$s_1 \to s_2 \to s_3 \to \dots$$

Termination

$$s_1 \to s_2 \to s_3 \to \dots$$

Recursive Path Ordering (Unconstrained TRSs)

Given: well-founded ordering $\triangleright$ on function symbols

Recursive Path Ordering (Unconstrained TRSs)

Given: well-founded ordering $\triangleright$ on function symbols

Define:

- $s \succeq t$ if:
  - $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i$ *or*
  - s and t are the same variable *or*
  - $s \succ t$
- $f(s_1, \ldots, s_n) \succ t$ if:
  - $s_i \succeq t$ for some i *or*
  - $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i$ and some $s_i \succ t_i$ *or*
  - $t = g(t_1, \ldots, t_m)$ and $f \triangleright g$ and $s \succ t_i$ for all i

Recursive Path Ordering (Constrained TRSs)

Given: well-founded ordering $\triangleright$ on function symbols

Define:

- $s \succeq t$ if:
  - $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i$ *or*
  - s and t are the same variable *or*
  - $s \succ t$
- $f(s_1, \ldots, s_n) \succ t$ if:
  - $s_i \succeq t$ for some i *or*
  - $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i$ and some $s_i \succ t_i$ *or*
  - $t = g(t_1, \ldots, t_m)$ and $f \triangleright g$ and $s \succ t_i$ for all i

Recursive Path Ordering (Constrained TRSs)

Given: well-founded ordering $\triangleright$ on function symbols, $\Sigma_{\mathtt{th}}$ minimal well-founded ordering $\sqsupset$ on values

Define:

- $s \succeq t$ if:
  - $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i$ *or*
  - s and t are the same variable *or*
  - $s \succ t$
- $f(s_1, \ldots, s_n) \succ t$ if:
  - $s_i \succeq t$ for some i *or*
  - $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i$ and some $s_i \succ t_i$ *or*
  - $t = g(t_1, \ldots, t_m)$ and $f \triangleright g$ and $s \succ t_i$ for all i

Recursive Path Ordering (Constrained TRSs)

Given: well-founded ordering $\triangleright$ on function symbols, $\Sigma_{\mathtt{th}}$ minimal well-founded ordering $\sqsupset$ on values

Define:

- $s \succeq t \ [\varphi]$ if:
  - $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i$ *or*
  - s and t are the same variable *or*
  - $s \succ t \ [\varphi]$
- $f(s_1, \ldots, s_n) \succ t \ [\varphi]$ if:
  - $s_i \succeq t \ [\varphi]$ for some i *or*
  - $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i \ [\varphi]$ and some $s_i \succ t_i \ [\varphi]$ *or*
  - $t = g(t_1, \ldots, t_m)$ and $f \triangleright g$ and $s \succ t_i \ [\varphi]$ for all i

Recursive Path Ordering (Constrained TRSs)

Given: well-founded ordering $\triangleright$ on function symbols, $\Sigma_{\mathtt{th}}$ minimal well-founded ordering $\sqsupset$ on values

Define:

- $s \succeq t\ [\varphi]$ if:
  - $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i$ *or*
  - s and t are the same variable *or* $s \succ t\ [\varphi]$ *or*
  - $\varphi \Rightarrow s \sqsupseteq t$ is valid and $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$
- $f(s_1, \ldots, s_n) \succ t\ [\varphi]$ if f is not a theory symbol and:
  - $s_i \succeq t\ [\varphi]$ for some i *or*
  - $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i\ [\varphi]$ and some $s_i \succ t_i\ [\varphi]$ *or*
  - $t = g(t_1, \ldots, t_m)$ and $f \triangleright g$ and $s \succ t_i\ [\varphi]$ for all i *or*
  - $t \in \mathit{FV}(\varphi)$
- $s \succ t\ [\varphi]$ if $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$ and $\varphi \Rightarrow s \sqsupset t$ valid

Recursive Path Ordering: Example

1. $s \succeq t\ [\varphi]$ if:
   1. $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i$ *or*
   2. s and t are the same variable *or* $s \succ t\ [\varphi]$ *or*
   3. $\varphi \Rightarrow s \sqsupseteq t$ is valid and $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$
2. $f(s_1, \ldots, s_n) \succ t\ [\varphi]$ if f is not a theory symbol and:
   1. $s_i \succeq t\ [\varphi]$ for some i *or*
   2. $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i\ [\varphi]$ and some $s_i \succ t_i\ [\varphi]$ *or*
   3. $t = g(t_1, \ldots, t_m)$ and $f \rhd g$ and $s \succ t_i\ [\varphi]$ for all i *or*
   4. $t \in \mathit{FV}(\varphi)$
3. $s \succ t\ [\varphi]$ if $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$ and $\varphi \Rightarrow s \sqsupset t$ valid

Recursive Path Ordering: Example

1. $s \succeq t\ [\varphi]$ if:
   1. $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i$ *or*
   2. s and t are the same variable *or* $s \succ t\ [\varphi]$ *or*
   3. $\varphi \Rightarrow s \sqsupseteq t$ is valid and $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$
2. $f(s_1, \ldots, s_n) \succ t\ [\varphi]$ if f is not a theory symbol and:
   1. $s_i \succeq t\ [\varphi]$ for some i *or*
   2. $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i\ [\varphi]$ and some $s_i \succ t_i\ [\varphi]$ *or*
   3. $t = g(t_1, \ldots, t_m)$ and $f \rhd g$ and $s \succ t_i\ [\varphi]$ for all i *or*
   4. $t \in \mathit{FV}(\varphi)$
3. $s \succ t\ [\varphi]$ if $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$ and $\varphi \Rightarrow s \sqsupset t$ valid

$$\begin{array}{lcll} \mathrm{sum}(x) & \to & 0 & [x \leq 0] \\ \mathrm{sum}(x) & \to & x + \mathrm{sum}(x-1) & [x \geq 0] \end{array}$$

Recursive Path Ordering: Example

1. $s \succeq t\ [\varphi]$ if:
   1. $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i$ *or*
   2. s and t are the same variable *or* $s \succ t\ [\varphi]$ *or*
   3. $\varphi \Rightarrow s \sqsupseteq t$ is valid and $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$
2. $f(s_1, \ldots, s_n) \succ t\ [\varphi]$ if f is not a theory symbol and:
   1. $s_i \succeq t\ [\varphi]$ for some i *or*
   2. $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i\ [\varphi]$ and some $s_i \succ t_i\ [\varphi]$ *or*
   3. $t = g(t_1, \ldots, t_m)$ and $f \rhd g$ and $s \succ t_i\ [\varphi]$ for all i *or*
   4. $t \in \mathit{FV}(\varphi)$
3. $s \succ t\ [\varphi]$ if $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$ and $\varphi \Rightarrow s \sqsupset t$ valid

$$\begin{array}{lcll} \mathsf{sum}(x) & \succ & 0 & [x \leq 0] \\ \mathsf{sum}(x) & \succ & x + \mathsf{sum}(x-1) & [x \geq 0] \end{array}$$

Recursive Path Ordering: Example

1. $s \succeq t\ [\varphi]$ if:
   1. $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i$ *or*
   2. s and t are the same variable *or* $s \succ t\ [\varphi]$ *or*
   3. $\varphi \Rightarrow s \sqsupseteq t$ is valid and $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$
2. $f(s_1, \ldots, s_n) \succ t\ [\varphi]$ if f is not a theory symbol and:
   1. $s_i \succeq t\ [\varphi]$ for some i *or*
   2. $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i\ [\varphi]$ and some $s_i \succ t_i\ [\varphi]$ *or*
   3. $t = g(t_1, \ldots, t_m)$ and $f \rhd g$ and $s \succ t_i\ [\varphi]$ for all i *or*
   4. $t \in \mathit{FV}(\varphi)$
3. $s \succ t\ [\varphi]$ if $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$ and $\varphi \Rightarrow s \sqsupset t$ valid

$$\begin{array}{lcll} \mathrm{sum}(x) & \succ & 0 & [x \leq 0] \\ \mathrm{sum}(x) & \succ & x + \mathrm{sum}(x-1) & [x \geq 0] \end{array}$$

Recursive Path Ordering: Example

1. $s \succeq t\ [\varphi]$ if:
   1. $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i$ *or*
   2. s and t are the same variable *or* $s \succ t\ [\varphi]$ *or*
   3. $\varphi \Rightarrow s \sqsupseteq t$ is valid and $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$
2. $f(s_1, \ldots, s_n) \succ t\ [\varphi]$ if f is not a theory symbol and:
   1. $s_i \succeq t\ [\varphi]$ for some i *or*
   2. $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i\ [\varphi]$ and some $s_i \succ t_i\ [\varphi]$ *or*
   3. $t = g(t_1, \ldots, t_m)$ and $f \rhd g$ and $s \succ t_i\ [\varphi]$ for all i *or*
   4. $t \in \mathit{FV}(\varphi)$
3. $s \succ t\ [\varphi]$ if $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$ and $\varphi \Rightarrow s \sqsupset t$ valid

$$\mathrm{sum}(x) \quad \succ \quad x + \mathrm{sum}(x - 1) \quad [x \geq 0]$$

Recursive Path Ordering: Example

1. $s \succeq t\ [\varphi]$ if:
   1. $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i$ *or*
   2. s and t are the same variable *or* $s \succ t\ [\varphi]$ *or*
   3. $\varphi \Rightarrow s \sqsupseteq t$ is valid and $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$
2. $f(s_1, \ldots, s_n) \succ t\ [\varphi]$ if f is not a theory symbol and:
   1. $s_i \succeq t\ [\varphi]$ for some i *or*
   2. $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i\ [\varphi]$ and some $s_i \succ t_i\ [\varphi]$ *or*
   3. $t = g(t_1, \ldots, t_m)$ and $f \rhd g$ and $s \succ t_i\ [\varphi]$ for all i *or*
   4. $t \in \mathit{FV}(\varphi)$
3. $s \succ t\ [\varphi]$ if $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$ and $\varphi \Rightarrow s \sqsupset t$ valid

$$\mathrm{sum}(x) \quad \succ \quad x + \mathrm{sum}(x-1) \quad [x \geq 0]$$

Recursive Path Ordering: Example

1. $s \succeq t\ [\varphi]$ if:
   1. $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i$ *or*
   2. s and t are the same variable *or* $s \succ t\ [\varphi]$ *or*
   3. $\varphi \Rightarrow s \sqsupseteq t$ is valid and $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$
2. $f(s_1, \ldots, s_n) \succ t\ [\varphi]$ if f is not a theory symbol and:
   1. $s_i \succeq t\ [\varphi]$ for some i *or*
   2. $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i\ [\varphi]$ and some $s_i \succ t_i\ [\varphi]$ *or*
   3. $t = g(t_1, \ldots, t_m)$ and $f \rhd g$ and $s \succ t_i\ [\varphi]$ for all i *or*
   4. $t \in \mathit{FV}(\varphi)$
3. $s \succ t\ [\varphi]$ if $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$ and $\varphi \Rightarrow s \sqsupset t$ valid

$$\mathrm{sum}(x) \quad \succ \quad x + \mathrm{sum}(x - 1) \quad [x \geq 0]$$

Recursive Path Ordering: Example

1. $s \succeq t\ [\varphi]$ if:
   1. $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i$ *or*
   2. s and t are the same variable *or* $s \succ t\ [\varphi]$ *or*
   3. $\varphi \Rightarrow s \sqsupseteq t$ is valid and $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$
2. $f(s_1, \ldots, s_n) \succ t\ [\varphi]$ if f is not a theory symbol and:
   1. $s_i \succeq t\ [\varphi]$ for some i *or*
   2. $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i\ [\varphi]$ and some $s_i \succ t_i\ [\varphi]$ *or*
   3. $t = g(t_1, \ldots, t_m)$ and $f \rhd g$ and $s \succ t_i\ [\varphi]$ for all i *or*
   4. $t \in \mathit{FV}(\varphi)$
3. $s \succ t\ [\varphi]$ if $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$ and $\varphi \Rightarrow s \sqsupset t$ valid

$$\begin{array}{lcll} \mathsf{sum}(x) & \succ & x & [x \geq 0] \\ \mathsf{sum}(x) & \succ & \mathsf{sum}(x-1) & [x \geq 0] \end{array}$$

Recursive Path Ordering: Example

1. $s \succeq t\ [\varphi]$ if:
   1. $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i$ *or*
   2. s and t are the same variable *or* $s \succ t\ [\varphi]$ *or*
   3. $\varphi \Rightarrow s \sqsupseteq t$ is valid and $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$
2. $f(s_1, \ldots, s_n) \succ t\ [\varphi]$ if f is not a theory symbol and:
   1. $s_i \succeq t\ [\varphi]$ for some i *or*
   2. $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i\ [\varphi]$ and some $s_i \succ t_i\ [\varphi]$ *or*
   3. $t = g(t_1, \ldots, t_m)$ and $f \rhd g$ and $s \succ t_i\ [\varphi]$ for all i *or*
   4. $t \in \mathit{FV}(\varphi)$
3. $s \succ t\ [\varphi]$ if $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$ and $\varphi \Rightarrow s \sqsupset t$ valid

$$\begin{array}{lcll} \mathsf{sum}(x) & \succ & x & [x \geq 0] \\ \mathsf{sum}(x) & \succ & \mathsf{sum}(x-1) & [x \geq 0] \end{array}$$

Recursive Path Ordering: Example

1. $s \succeq t\ [\varphi]$ if:
   1. $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i$ *or*
   2. s and t are the same variable *or* $s \succ t\ [\varphi]$ *or*
   3. $\varphi \Rightarrow s \sqsupseteq t$ is valid and $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$
2. $f(s_1, \ldots, s_n) \succ t\ [\varphi]$ if f is not a theory symbol and:
   1. $s_i \succeq t\ [\varphi]$ for some i *or*
   2. $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i\ [\varphi]$ and some $s_i \succ t_i\ [\varphi]$ *or*
   3. $t = g(t_1, \ldots, t_m)$ and $f \rhd g$ and $s \succ t_i\ [\varphi]$ for all i *or*
   4. $t \in \mathit{FV}(\varphi)$
3. $s \succ t\ [\varphi]$ if $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$ and $\varphi \Rightarrow s \sqsupset t$ valid

$$\begin{array}{rcll} x & \succeq & x & [x \geq 0] \\ \mathrm{sum}(x) & \succ & \mathrm{sum}(x-1) & [x \geq 0] \end{array}$$

Recursive Path Ordering: Example

1. $s \succeq t\ [\varphi]$ if:
   1. $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i$ *or*
   2. s and t are the same variable *or* $s \succ t\ [\varphi]$ *or*
   3. $\varphi \Rightarrow s \sqsupseteq t$ is valid and $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$
2. $f(s_1, \ldots, s_n) \succ t\ [\varphi]$ if f is not a theory symbol and:
   1. $s_i \succeq t\ [\varphi]$ for some i *or*
   2. $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i\ [\varphi]$ and some $s_i \succ t_i\ [\varphi]$ *or*
   3. $t = g(t_1, \ldots, t_m)$ and $f \rhd g$ and $s \succ t_i\ [\varphi]$ for all i *or*
   4. $t \in \mathit{FV}(\varphi)$
3. $s \succ t\ [\varphi]$ if $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$ and $\varphi \Rightarrow s \sqsupset t$ valid

$$\begin{array}{rcll} x & \succeq & x & [x \geq 0] \\ \mathrm{sum}(x) & \succ & \mathrm{sum}(x-1) & [x \geq 0] \end{array}$$

Recursive Path Ordering: Example

1. $s \succeq t\ [\varphi]$ if:
   1. $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i$ *or*
   2. s and t are the same variable *or* $s \succ t\ [\varphi]$ *or*
   3. $\varphi \Rightarrow s \sqsupseteq t$ is valid and $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$
2. $f(s_1, \ldots, s_n) \succ t\ [\varphi]$ if f is not a theory symbol and:
   1. $s_i \succeq t\ [\varphi]$ for some i *or*
   2. $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i\ [\varphi]$ and some $s_i \succ t_i\ [\varphi]$ *or*
   3. $t = g(t_1, \ldots, t_m)$ and $f \rhd g$ and $s \succ t_i\ [\varphi]$ for all i *or*
   4. $t \in \mathit{FV}(\varphi)$
3. $s \succ t\ [\varphi]$ if $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$ and $\varphi \Rightarrow s \sqsupset t$ valid

$$\begin{array}{rcll} x & \succeq & x & [x \geq 0] \\ \mathrm{sum}(x) & \succ & \mathrm{sum}(x-1) & [x \geq 0] \end{array}$$

Recursive Path Ordering: Example

1. $s \succeq t\ [\varphi]$ if:
   1. $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i$ *or*
   2. s and t are the same variable *or* $s \succ t\ [\varphi]$ *or*
   3. $\varphi \Rightarrow s \sqsupseteq t$ is valid and $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$
2. $f(s_1, \ldots, s_n) \succ t\ [\varphi]$ if f is not a theory symbol and:
   1. $s_i \succeq t\ [\varphi]$ for some i *or*
   2. $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i\ [\varphi]$ and some $s_i \succ t_i\ [\varphi]$ *or*
   3. $t = g(t_1, \ldots, t_m)$ and $f \rhd g$ and $s \succ t_i\ [\varphi]$ for all i *or*
   4. $t \in \mathit{FV}(\varphi)$
3. $s \succ t\ [\varphi]$ if $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$ and $\varphi \Rightarrow s \sqsupset t$ valid

$$\mathrm{sum}(x) \quad \succ \quad \mathrm{sum}(x-1) \quad [x \geq 0]$$

Recursive Path Ordering: Example

1. $s \succeq t\ [\varphi]$ if:
   1. $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i$ *or*
   2. s and t are the same variable *or* $s \succ t\ [\varphi]$ *or*
   3. $\varphi \Rightarrow s \sqsupseteq t$ is valid and $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$
2. $f(s_1, \ldots, s_n) \succ t\ [\varphi]$ if f is not a theory symbol and:
   1. $s_i \succeq t\ [\varphi]$ for some i *or*
   2. $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i\ [\varphi]$ and some $s_i \succ t_i\ [\varphi]$ *or*
   3. $t = g(t_1, \ldots, t_m)$ and $f \rhd g$ and $s \succ t_i\ [\varphi]$ for all i *or*
   4. $t \in \mathit{FV}(\varphi)$
3. $s \succ t\ [\varphi]$ if $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$ and $\varphi \Rightarrow s \sqsupset t$ valid

$$\mathrm{sum}(x) \quad \succ \quad \mathrm{sum}(x-1) \quad [x \geq 0]$$

Recursive Path Ordering: Example

1. $s \succeq t\ [\varphi]$ if:
   1. $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i$ *or*
   2. s and t are the same variable *or* $s \succ t\ [\varphi]$ *or*
   3. $\varphi \Rightarrow s \sqsupseteq t$ is valid and $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$
2. $f(s_1, \ldots, s_n) \succ t\ [\varphi]$ if f is not a theory symbol and:
   1. $s_i \succeq t\ [\varphi]$ for some i *or*
   2. $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i\ [\varphi]$ and some $s_i \succ t_i\ [\varphi]$ *or*
   3. $t = g(t_1, \ldots, t_m)$ and $f \rhd g$ and $s \succ t_i\ [\varphi]$ for all i *or*
   4. $t \in \mathit{FV}(\varphi)$
3. $s \succ t\ [\varphi]$ if $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$ and $\varphi \Rightarrow s \sqsupset t$ valid

$$\mathrm{sum}(x) \quad \succ \quad \mathrm{sum}(x-1) \quad [x \geq 0]$$

Recursive Path Ordering: Example

1. $s \succeq t\ [\varphi]$ if:
   1. $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i$ *or*
   2. s and t are the same variable *or* $s \succ t\ [\varphi]$ *or*
   3. $\varphi \Rightarrow s \sqsupseteq t$ is valid and $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$
2. $f(s_1, \ldots, s_n) \succ t\ [\varphi]$ if f is not a theory symbol and:
   1. $s_i \succeq t\ [\varphi]$ for some i *or*
   2. $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i\ [\varphi]$ and some $s_i \succ t_i\ [\varphi]$ *or*
   3. $t = g(t_1, \ldots, t_m)$ and $f \rhd g$ and $s \succ t_i\ [\varphi]$ for all i *or*
   4. $t \in \mathit{FV}(\varphi)$
3. $s \succ t\ [\varphi]$ if $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$ and $\varphi \Rightarrow s \sqsupset t$ valid

$$x \quad \succ \quad x - 1 \quad [x \geq 0]$$

Recursive Path Ordering: Example

1. $s \succeq t\ [\varphi]$ if:
   1. $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i$ *or*
   2. s and t are the same variable *or* $s \succ t\ [\varphi]$ *or*
   3. $\varphi \Rightarrow s \sqsupseteq t$ is valid and $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$
2. $f(s_1, \ldots, s_n) \succ t\ [\varphi]$ if f is not a theory symbol and:
   1. $s_i \succeq t\ [\varphi]$ for some i *or*
   2. $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i\ [\varphi]$ and some $s_i \succ t_i\ [\varphi]$ *or*
   3. $t = g(t_1, \ldots, t_m)$ and $f \rhd g$ and $s \succ t_i\ [\varphi]$ for all i *or*
   4. $t \in \mathit{FV}(\varphi)$
3. $s \succ t\ [\varphi]$ if $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$ and $\varphi \Rightarrow s \sqsupset t$ valid

$$x \quad \succ \quad x - 1 \quad [x \geq 0]$$

Recursive Path Ordering: Example

1. $s \succeq t\ [\varphi]$ if:
   1. $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i$ *or*
   2. s and t are the same variable *or* $s \succ t\ [\varphi]$ *or*
   3. $\varphi \Rightarrow s \sqsupseteq t$ is valid and $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$
2. $f(s_1, \ldots, s_n) \succ t\ [\varphi]$ if f is not a theory symbol and:
   1. $s_i \succeq t\ [\varphi]$ for some i *or*
   2. $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i\ [\varphi]$ and some $s_i \succ t_i\ [\varphi]$ *or*
   3. $t = g(t_1, \ldots, t_m)$ and $f \rhd g$ and $s \succ t_i\ [\varphi]$ for all i *or*
   4. $t \in \mathit{FV}(\varphi)$
3. $s \succ t\ [\varphi]$ if $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$ and $\varphi \Rightarrow s \sqsupset t$ valid

$$x \quad \succ \quad x - 1 \quad [x \geq 0]$$

Recursive Path Ordering: Example

1. $s \succeq t\ [\varphi]$ if:
   1. $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i$ *or*
   2. s and t are the same variable *or* $s \succ t\ [\varphi]$ *or*
   3. $\varphi \Rightarrow s \sqsupseteq t$ is valid and $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$
2. $f(s_1, \ldots, s_n) \succ t\ [\varphi]$ if f is not a theory symbol and:
   1. $s_i \succeq t\ [\varphi]$ for some i *or*
   2. $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i\ [\varphi]$ and some $s_i \succ t_i\ [\varphi]$ *or*
   3. $t = g(t_1, \ldots, t_m)$ and $f \rhd g$ and $s \succ t_i\ [\varphi]$ for all i *or*
   4. $t \in \mathit{FV}(\varphi)$
3. $s \succ t\ [\varphi]$ if $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$ and $\varphi \Rightarrow s \sqsupset t$ valid

$$x \geq 0 \Rightarrow x \sqsupset x - 1$$

Recursive Path Ordering: Example

1. $s \succeq t\ [\varphi]$ if:
   1. $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i$ *or*
   2. s and t are the same variable *or* $s \succ t\ [\varphi]$ *or*
   3. $\varphi \Rightarrow s \sqsupseteq t$ is valid and $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$
2. $f(s_1, \ldots, s_n) \succ t\ [\varphi]$ if f is not a theory symbol and:
   1. $s_i \succeq t\ [\varphi]$ for some i *or*
   2. $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i\ [\varphi]$ and some $s_i \succ t_i\ [\varphi]$ *or*
   3. $t = g(t_1, \ldots, t_m)$ and $f \rhd g$ and $s \succ t_i\ [\varphi]$ for all i *or*
   4. $t \in \mathit{FV}(\varphi)$
3. $s \succ t\ [\varphi]$ if $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$ and $\varphi \Rightarrow s \sqsupset t$ valid

let $x \sqsupset y$ if $x > y$ and $x \geq 0$

$x \geq 0 \Rightarrow x \sqsupset x - 1$

Recursive Path Ordering: Example

1. $s \succeq t\ [\varphi]$ if:
   1. $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i$ *or*
   2. s and t are the same variable *or* $s \succ t\ [\varphi]$ *or*
   3. $\varphi \Rightarrow s \sqsupseteq t$ is valid and $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$
2. $f(s_1, \ldots, s_n) \succ t\ [\varphi]$ if f is not a theory symbol and:
   1. $s_i \succeq t\ [\varphi]$ for some i *or*
   2. $t = f(t_1, \ldots, t_n)$ and each $s_i \succeq t_i\ [\varphi]$ and some $s_i \succ t_i\ [\varphi]$ *or*
   3. $t = g(t_1, \ldots, t_m)$ and $f \rhd g$ and $s \succ t_i\ [\varphi]$ for all i *or*
   4. $t \in \mathit{FV}(\varphi)$
3. $s \succ t\ [\varphi]$ if $s, t \in \mathit{Terms}(\Sigma_{\mathtt{th}}, \mathit{FV}(\varphi))$ and $\varphi \Rightarrow s \sqsupset t$ valid

let $x \sqsupset y$ if $x > y$ and $x \geq 0$

$x \geq 0 \Rightarrow (x > x - 1 \wedge x \geq 0)$

Observation

Observation

Confluence: test non-satisfiability of constraints $\varphi \wedge \psi$

Observation

Confluence: test non-satisfiability of constraints $\varphi \wedge \psi$

Termination: test validity of constraints $\varphi \Rightarrow s \sqsupset t$ or $\varphi \Rightarrow s \sqsupseteq t$

Observation

Confluence: test validity of constraints $\neg(\varphi \wedge \psi)$

Termination: test validity of constraints $\varphi \Rightarrow s \sqsupset t$ or $\varphi \Rightarrow s \sqsupseteq t$

Implementation

Idea: transform problems into sequences of logical constraints over the theory, and solve these

Implementation

Idea: transform problems into sequences of logical constraints over the theory, and solve these

Problem: leads to requirements like: find $x_1, \ldots, x_n$ such that $\forall y_1 \ldots y_m[\varphi]$

Implementation

Idea: transform problems into sequences of logical constraints over the theory, and solve these

Problem: leads to requirements like: find $x_1, \ldots, x_n$ such that $\forall y_1 \ldots y_m[\varphi]$

Solutions:

- dedicated analysis (probably theory-dependent)

Implementation

Idea: transform problems into sequences of logical constraints over the theory, and solve these

Problem: leads to requirements like: find $x_1, \ldots, x_n$ such that $\forall y_1 \ldots y_m[\varphi]$

Solutions:

- dedicated analysis (probably theory-dependent)
- use existing (or future) SMT-solvers!

Future Plans

Future Plans

- extend existing unconstrained techniques

Future Plans

- extend existing unconstrained techniques
- define notions of complexity

Future Plans

- extend existing unconstrained techniques
- define notions of complexity
- implement tool with basic analysis (confluence, termination, function equivalence)

Future Plans

- extend existing unconstrained techniques
- define notions of complexity
- implement tool with basic analysis (confluence, termination, function equivalence)
- implement and improve transformation from imperative programs

Improving Modelling

```
int square(x) { return x * x; }
int squaresum(int x) {
  int i = 0, z = 0;
  if (x <= 0) return 0;
  for (i = 0; i != x; i++)
    z += square(i);
  return z;
}
```

Improving Modelling

```
int square(x) { return x * x; }
int squaresum(int x) {
  int i = 0, z = 0;
  if (x <= 0) return 0;
  for (i = 0; i != x; i++)
    z += square(i);
  return z;
}
```

$$
\begin{array}{rcll}
\mathsf{sum}(x) & \to & \mathsf{u}(x,0,0) & \\
\mathsf{u}(x,i,z) & \to & 0 & [x \leq 0] \\
\mathsf{u}(x,i,z) & \to & \mathsf{v}(x,i,z) & [\neg(x \leq 0)] \\
\mathsf{v}(x,i,z) & \to & \mathsf{v}(x,i+1,z+\mathsf{square}(i)) & [i \neq x] \\
\mathsf{v}(x,i,z) & \to & z & [i = x] \\
\mathsf{square}(x) & \to & x * x &
\end{array}
$$

Improving Modelling

```
int square(x) { return x * x; }
int squaresum(int x) {
  int i = 0, z = 0;
  if (x <= 0) return 0;
  for (i = 0; i != x; i++)
    z += square(i);
  return z;
}
```

$$
\begin{array}{rcll}
\mathsf{sum}(x) & \to & \mathsf{u}(x,0,0) & \\
\mathsf{u}(x,i,z) & \to & 0 & [x \leq 0] \\
\mathsf{u}(x,i,z) & \to & \mathsf{v}(x,i,z) & [\neg(x \leq 0)] \\
\mathsf{v}(x,i,z) & \to & \mathsf{v}(x,i+1,z+\mathsf{square}(i)) & [i \neq x \wedge x > 0] \\
\mathsf{v}(x,i,z) & \to & z & [i = x] \\
\mathsf{square}(x) & \to & x * x &
\end{array}
$$

Conclusions

Conclusions

LCTRSs are:

- natural: we obtain very intuitive systems
- general: not limited to a fixed logic, infinity only in signatures
- powerful: existing formalisms are typically simulated
- flexible: normal analysis techniques extend in a natural way

Conclusions

LCTRSs are:

- natural: we obtain very intuitive systems
- general: not limited to a fixed logic, infinity only in signatures
- powerful: existing formalisms are typically simulated
- flexible: normal analysis techniques extend in a natural way

Questions?