Rewriting with Semantic Data Structures: Termination using Dependency Pairs

Stephan Falke and Deepak Kapur

Department of Computer Science
University of New Mexico
Albuquerque, NM, USA

October 2007

Introduction

- Standard TRSs on free data structures cannot model semantic data structures (sets, multisets, sorted lists, sorted arrays, balanced trees, ...)
- We define a class of rewrite systems that allows the use of semantic data structures
- Natural numbers are Built-in, particularly, equality, ordering, and divisibility constraints on natural numbers, and their boolean combinations, are supported to guard when a rewrite rule may be applied
- The rewrite mechanism combines normalized equational rewriting with validity checking of instantiated constraints on natural numbers

Introduction

- Standard TRSs on free data structures cannot model semantic data structures (sets, multisets, sorted lists, sorted arrays, balanced trees, ...)
- We define a class of rewrite systems that allows the use of semantic data structures
- Natural numbers are Built-in, particularly, equality, ordering, and divisibility constraints on natural numbers, and their boolean combinations, are supported to guard when a rewrite rule may be applied
- The rewrite mechanism combines normalized equational rewriting with validity checking of instantiated constraints on natural numbers

Introduction

- Standard TRSs on free data structures cannot model semantic data structures (sets, multisets, sorted lists, sorted arrays, balanced trees, ...)
- We define a class of rewrite systems that allows the use of semantic data structures
- Natural numbers are Built-in, particularly, equality, ordering, and divisibility constraints on natural numbers, and their boolean combinations, are supported to guard when a rewrite rule may be applied
- The rewrite mechanism combines normalized equational rewriting with validity checking of instantiated constraints on natural numbers

Introduction

- Standard TRSs on free data structures cannot model semantic data structures (sets, multisets, sorted lists, sorted arrays, balanced trees, . . .)
- We define a class of rewrite systems that allows the use of semantic data structures
- Natural numbers are Built-in, particularly, equality, ordering, and divisibility constraints on natural numbers, and their boolean combinations, are supported to guard when a rewrite rule may be applied
- The rewrite mechanism combines normalized equational rewriting with validity checking of instantiated constraints on natural numbers

Introduction

- Functional programs can be translated into this class of rewrite systems
- Imperative programs operating on natural numbers can be translated into this class of rewrite systems
- Expressive and natural framework for writing algorithms
- Termination of program = termination of rewrite system
- The dependency pair framework is extended to be applicable to this class of rewrite systems

Introduction

- Functional programs can be translated into this class of rewrite systems
- Imperative programs operating on natural numbers can be translated into this class of rewrite systems
- Expressive and natural framework for writing algorithms
- Termination of program = termination of rewrite system
- The dependency pair framework is extended to be applicable to this class of rewrite systems

Introduction

- Functional programs can be translated into this class of rewrite systems
- Imperative programs operating on natural numbers can be translated into this class of rewrite systems
- Expressive and natural framework for writing algorithms
- Termination of program = termination of rewrite system
- The dependency pair framework is extended to be applicable to this class of rewrite systems

Introduction

- Functional programs can be translated into this class of rewrite systems
- Imperative programs operating on natural numbers can be translated into this class of rewrite systems
- Expressive and natural framework for writing algorithms
- Termination of program = termination of rewrite system
- The dependency pair framework is extended to be applicable to this class of rewrite systems

Introduction

- Functional programs can be translated into this class of rewrite systems
- Imperative programs operating on natural numbers can be translated into this class of rewrite systems
- Expressive and natural framework for writing algorithms
- Termination of program = termination of rewrite system
- The dependency pair framework is extended to be applicable to this class of rewrite systems

Example – Mergesort

Sets are modelled using $\emptyset$, $\langle \cdot \rangle$ (singleton set) and $\cup$. The following algorithm produces a sorted list.

$$
\begin{aligned}
\mathsf{merge}(\mathsf{nil}, y) &\to y \\
\mathsf{merge}(x, \mathsf{nil}) &\to x \\
\mathsf{merge}(\mathsf{cons}(x, xs), \mathsf{cons}(y, ys)) &\to \mathsf{cons}(y, \mathsf{merge}(\mathsf{cons}(x, xs), ys))\ [\![x > y]\!] \\
\mathsf{merge}(\mathsf{cons}(x, xs), \mathsf{cons}(y, ys)) &\to \mathsf{cons}(x, \mathsf{merge}(xs, \mathsf{cons}(y, ys)))\ [\![x \not> y]\!] \\
\mathsf{msort}(\emptyset) &\to \mathsf{nil} \\
\mathsf{msort}(\langle x \rangle) &\to \mathsf{cons}(x, \mathsf{nil}) \\
\mathsf{msort}(x \cup y) &\to \mathsf{merge}(\mathsf{msort}(x), \mathsf{msort}(y))
\end{aligned}
$$

Modelling Semantic Data Structures

- For sets, we expect the following properties:

$$
\begin{aligned}
u \cup v &\approx v \cup u \\
u \cup (v \cup w) &\approx (u \cup v) \cup w \\
u \cup u &\approx u \\
u \cup \emptyset &\approx \emptyset
\end{aligned}
$$

- The mergesort algorithm does not terminate if we use rewriting modulo these properties (or extended rewriting with these properties)
- Solution: Use normalized rewriting (Marché, 1996)
- The last two properties are oriented as rewrite rules and the redex is normalized with these rules before matching it with a rule from $\mathcal{R}$

Modelling Semantic Data Structures

- For sets, we expect the following properties:

$$\begin{aligned} u \cup v &\approx v \cup u \\ u \cup (v \cup w) &\approx (u \cup v) \cup w \\ u \cup u &\approx u \\ u \cup \emptyset &\approx \emptyset \end{aligned}$$

- The mergesort algorithm does not terminate if we use rewriting modulo these properties (or extended rewriting with these properties)
- Solution: Use normalized rewriting (Marché, 1996)
- The last two properties are oriented as rewrite rules and the redex is normalized with these rules before matching it with a rule from $\mathcal{R}$

Modelling Semantic Data Structures

- For sets, we expect the following properties:

$$\begin{aligned} u \cup v &\approx v \cup u \\ u \cup (v \cup w) &\approx (u \cup v) \cup w \\ u \cup u &\approx u \\ u \cup \emptyset &\approx \emptyset \end{aligned}$$

- The mergesort algorithm does not terminate if we use rewriting modulo these properties (or extended rewriting with these properties)
- Solution: Use normalized rewriting (Marché, 1996)
- The last two properties are oriented as rewrite rules and the redex is normalized with these rules before matching it with a rule from $\mathcal{R}$

Modelling Semantic Data Structures

- For sets, we expect the following properties:

$$\begin{aligned} u \cup v &\approx v \cup u \\ u \cup (v \cup w) &\approx (u \cup v) \cup w \\ u \cup u &\approx u \\ u \cup \emptyset &\approx \emptyset \end{aligned}$$

- The mergesort algorithm does not terminate if we use rewriting modulo these properties (or extended rewriting with these properties)
- Solution: Use normalized rewriting (Marché, 1996)
- The last two properties are oriented as rewrite rules and the redex is normalized with these rules before matching it with a rule from $\mathcal{R}$

Example – Sieve of Eratosthenes

The following algorithm simulates the sieve of Eratosthenes up to a given upper bound.

$$
\begin{array}{rcll}
\mathsf{primes}(x) & \to & \mathsf{sieve}(\mathsf{nats}(2, x)) & \\
\mathsf{nats}(x, y) & \to & \mathsf{nil} & [\![x > y]\!] \\
\mathsf{nats}(x, y) & \to & \mathsf{cons}(x, \mathsf{nats}(x + 1, y)) & [\![x \not> y]\!] \\
\mathsf{sieve}(\mathsf{nil}) & \to & \mathsf{nil} & \\
\mathsf{sieve}(\mathsf{cons}(x, ys)) & \to & \mathsf{cons}(x, \mathsf{sieve}(\mathsf{filter}(x, ys))) & \\
\mathsf{filter}(x, \mathsf{nil}) & \to & \mathsf{nil} & \\
\mathsf{filter}(x, \mathsf{cons}(y, zs)) & \to & \mathsf{cond}(\mathsf{isdiv}(x, y), x, y, zs) & \\
\mathsf{cond}(\mathsf{true}, x, y, zs) & \to & \mathsf{filter}(x, zs) & \\
\mathsf{cond}(\mathsf{false}, x, y, zs) & \to & \mathsf{cons}(y, \mathsf{filter}(x, zs)) & \\
\mathsf{isdiv}(x, 0) & \to & \mathsf{true} & [\![x > 0]\!] \\
\mathsf{isdiv}(x, y) & \to & \mathsf{false} & [\![x > y \wedge y > 0]\!] \\
\mathsf{isdiv}(x, x + y) & \to & \mathsf{isdiv}(x, y) & [\![x > 0]\!]
\end{array}
$$

Using $\mathcal{PA}$-constraints

- We use the following properties of natural numbers:

$$\begin{aligned} u + v &\approx v + u \\ u + (v + w) &\approx (u + v) + w \\ u + 0 &\approx u \end{aligned}$$

- $\mathcal{PA}$-constraints are Boolean combinations of atomic $\mathcal{PA}$-constraints of the form
  - $s \simeq t$,
  - $s > t$, and
  - $k \mid s$ (for some $k \in \mathbb{N} - \{0\}$)
- $s \geq t$, $s \not> t$, $s \not\simeq t$, ... can be defined as usual
- A rewrite rule can only be applied if the constraint becomes $\mathcal{PA}$-valid after being instantiated with the matching substitution

Using $\mathcal{PA}$-constraints

- We use the following properties of natural numbers:

$$\begin{array}{rcl} u + v & \approx & v + u \\ u + (v + w) & \approx & (u + v) + w \\ u + 0 & \approx & u \end{array}$$

- $\mathcal{PA}$-constraints are Boolean combinations of atomic $\mathcal{PA}$-constraints of the form
  - $s \simeq t$,
  - $s > t$, and
  - $k \mid s$ (for some $k \in \mathbb{N} - \{0\}$)
- $s \geq t$, $s \not> t$, $s \not\simeq t$, ... can be defined as usual
- A rewrite rule can only be applied if the constraint becomes $\mathcal{PA}$-valid after being instantiated with the matching substitution

Using $\mathcal{PA}$-constraints

- We use the following properties of natural numbers:

$$\begin{aligned} u + v &\approx v + u \\ u + (v + w) &\approx (u + v) + w \\ u + 0 &\approx u \end{aligned}$$

- $\mathcal{PA}$-constraints are Boolean combinations of atomic $\mathcal{PA}$-constraints of the form
  - $s \simeq t$,
  - $s > t$, and
  - $k \mid s$ (for some $k \in \mathbb{N} - \{0\}$)
- $s \geq t$, $s \not> t$, $s \not\simeq t$, ... can be defined as usual
- A rewrite rule can only be applied if the constraint becomes $\mathcal{PA}$-valid after being instantiated with the matching substitution

Using $\mathcal{PA}$-constraints

- We use the following properties of natural numbers:

$$\begin{array}{rcl} u + v & \approx & v + u \\ u + (v + w) & \approx & (u + v) + w \\ u + 0 & \approx & u \end{array}$$

- $\mathcal{PA}$-constraints are Boolean combinations of atomic $\mathcal{PA}$-constraints of the form
  - $s \simeq t$,
  - $s > t$, and
  - $k \mid s$ (for some $k \in \mathbb{N} - \{0\}$)
- $s \geq t$, $s \not> t$, $s \not\simeq t$, ... can be defined as usual
- A rewrite rule can only be applied if the constraint becomes $\mathcal{PA}$-valid after being instantiated with the matching substitution

Example – Imperative Program 1

- Consider the imperative program fragment (Cook *et. al.*, 2005)

```
while (x > 0) {
    x++;
    y := 1;
    while (x > y) {
        y++
    }
    x := x - 2
}
```

- It can be translated into

$$
\begin{array}{rcll}
\mathsf{eval}_0(x, y) & \to & \mathsf{eval}_1(x+1, 1) & [\![x > 0]\!] \\
\mathsf{eval}_1(x, y) & \to & \mathsf{eval}_1(x, y+1) & [\![x > y]\!] \\
\mathsf{eval}_1(x+2, y) & \to & \mathsf{eval}_0(x, y) & [\![x+2 \not> y]\!]
\end{array}
$$

Example – Imperative Program 1

- Consider the imperative program fragment (Cook *et. al.*, 2005)

```
while (x > 0) {
   x++;
   y := 1;
   while (x > y) {
      y++
   }
   x := x - 2
}
```

- It can be translated into

$$
\begin{array}{rcll}
\mathsf{eval}_0(x, y) & \to & \mathsf{eval}_1(x + 1, 1) & [\![x > 0]\!] \\
\mathsf{eval}_1(x, y) & \to & \mathsf{eval}_1(x, y + 1) & [\![x > y]\!] \\
\mathsf{eval}_1(x + 2, y) & \to & \mathsf{eval}_0(x, y) & [\![x + 2 \not> y]\!]
\end{array}
$$

Example – Imperative Program 2

- Consider the imperative program fragment

```
while (x > 0 || y > 0) {
    if (x > 0) {
        x--
    } else if (y > 0) {
        y--
    } else {
        continue
    }
}
```

- It can be translated into

$$\begin{aligned} \mathsf{eval}(x+1,y) &\to \mathsf{eval}(x,y) \ [\![(x+1>0 \lor y>0) \land x+1>0]\!] \\ \mathsf{eval}(x,y+1) &\to \mathsf{eval}(x,y) \ [\![(x>0 \lor y+1>0) \land x \not> 0 \land y+1>0]\!] \\ \mathsf{eval}(x,y) &\to \mathsf{eval}(x,y) \ [\![(x>0 \lor y>0) \land x \not> 0 \land y \not> 0]\!] \end{aligned}$$

Example – Imperative Program 2

- Consider the imperative program fragment

```
while (x > 0 || y > 0) {
    if (x > 0) {
        x--
    } else if (y > 0) {
        y--
    } else {
        continue
    }
}
```

- It can be translated into

$$\begin{aligned}
\mathsf{eval}(x+1, y) &\to \mathsf{eval}(x, y)\ [\![(x+1 > 0 \vee y > 0) \wedge x+1 > 0]\!] \\
\mathsf{eval}(x, y+1) &\to \mathsf{eval}(x, y)\ [\![(x > 0 \vee y+1 > 0) \wedge x \not> 0 \wedge y+1 > 0]\!] \\
\mathsf{eval}(x, y) &\to \mathsf{eval}(x, y)\ [\![(x > 0 \vee y > 0) \wedge x \not> 0 \wedge y \not> 0]\!]
\end{aligned}$$

Constrained Equational Systems

- A constrained equational system (CES) has the form $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ where
  - $\mathcal{R}$ is a finite set of rewrite rules with $\mathcal{PA}$-constraints,
  - $\mathcal{S}$ is a finite set of rewrite rules with $\mathcal{PA}$-constraints over the constructors of $\mathcal{R}$,
  - $\mathcal{E}$ is a finite set of equations over the constructors of $\mathcal{R}$
- $\mathcal{S}$ and $\mathcal{E}$ model the semantic data structures
- Assumptions are imposed on $\mathcal{S}$ and $\mathcal{E}$
- We assume that no function symbol has sort $\mathtt{nat}$
- Rewriting with a CES: $s \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} t$ iff
  - $s|_p \downarrow_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} \sim_{\mathcal{E}\cup\mathcal{PA}} l\sigma$,
  - $C\sigma$ is $\mathcal{PA}$-valid, and
  - $t = s[r\sigma]_p$

  for some position p, some substitution σ, and some rule $l \to r[\![C]\!]$ from $\mathcal{R}$ (Here, $\downarrow_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}}$ means normalization w.r.t. $\mathcal{S}$)

Constrained Equational Systems

- A constrained equational system (CES) has the form $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ where
  - $\mathcal{R}$ is a finite set of rewrite rules with $\mathcal{PA}$-constraints,
  - $\mathcal{S}$ is a finite set of rewrite rules with $\mathcal{PA}$-constraints over the constructors of $\mathcal{R}$,
  - $\mathcal{E}$ is a finite set of equations over the constructors of $\mathcal{R}$
- $\mathcal{S}$ and $\mathcal{E}$ model the semantic data structures
- Assumptions are imposed on $\mathcal{S}$ and $\mathcal{E}$
- We assume that no function symbol has sort nat
- Rewriting with a CES: $s \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} t$ iff
  - $s|_p{\downarrow}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} \sim_{\mathcal{E}\cup\mathcal{PA}} l\sigma$,
  - $C\sigma$ is $\mathcal{PA}$-valid, and
  - $t = s[r\sigma]_p$

  for some position p, some substitution σ, and some rule $l \to r[\![C]\!]$ from $\mathcal{R}$ (Here, ${\downarrow}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}}$ means normalization w.r.t. $\mathcal{S}$)

Constrained Equational Systems

- A constrained equational system (CES) has the form $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ where
  - $\mathcal{R}$ is a finite set of rewrite rules with $\mathcal{PA}$-constraints,
  - $\mathcal{S}$ is a finite set of rewrite rules with $\mathcal{PA}$-constraints over the constructors of $\mathcal{R}$,
  - $\mathcal{E}$ is a finite set of equations over the constructors of $\mathcal{R}$
- $\mathcal{S}$ and $\mathcal{E}$ model the semantic data structures
- Assumptions are imposed on $\mathcal{S}$ and $\mathcal{E}$
- We assume that no function symbol has sort nat
- Rewriting with a CES: $s \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} t$ iff
  - $s|_p{\downarrow}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} \sim_{\mathcal{E}\cup\mathcal{PA}} l\sigma$,
  - $C\sigma$ is $\mathcal{PA}$-valid, and
  - $t = s[r\sigma]_p$

  for some position p, some substitution σ, and some rule $l \to r[\![C]\!]$ from $\mathcal{R}$ (Here, ${\downarrow}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}}$ means normalization w.r.t. $\mathcal{S}$)

Constrained Equational Systems

- A constrained equational system (CES) has the form $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ where
  - $\mathcal{R}$ is a finite set of rewrite rules with $\mathcal{PA}$-constraints,
  - $\mathcal{S}$ is a finite set of rewrite rules with $\mathcal{PA}$-constraints over the constructors of $\mathcal{R}$,
  - $\mathcal{E}$ is a finite set of equations over the constructors of $\mathcal{R}$
- $\mathcal{S}$ and $\mathcal{E}$ model the semantic data structures
- Assumptions are imposed on $\mathcal{S}$ and $\mathcal{E}$
- We assume that no function symbol has sort `nat`
- Rewriting with a CES: $s \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} t$ iff
  - $s|_p{\downarrow_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}}} \sim_{\mathcal{E}\cup\mathcal{PA}} l\sigma$,
  - $C\sigma$ is $\mathcal{PA}$-valid, and
  - $t = s[r\sigma]_p$

  for some position p, some substitution σ, and some rule $l \to r[\![C]\!]$ from $\mathcal{R}$ (Here, $\downarrow_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}}$ means normalization w.r.t. $\mathcal{S}$)

Constrained Equational Systems

- A constrained equational system (CES) has the form $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ where
  - $\mathcal{R}$ is a finite set of rewrite rules with $\mathcal{PA}$-constraints,
  - $\mathcal{S}$ is a finite set of rewrite rules with $\mathcal{PA}$-constraints over the constructors of $\mathcal{R}$,
  - $\mathcal{E}$ is a finite set of equations over the constructors of $\mathcal{R}$
- $\mathcal{S}$ and $\mathcal{E}$ model the semantic data structures
- Assumptions are imposed on $\mathcal{S}$ and $\mathcal{E}$
- We assume that no function symbol has sort `nat`
- Rewriting with a CES: $s \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} t$ iff
  - $s|_p{\downarrow}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} \sim_{\mathcal{E}\cup\mathcal{PA}} l\sigma$,
  - $C\sigma$ is $\mathcal{PA}$-valid, and
  - $t = s[r\sigma]_p$

  for some position p, some substitution σ, and some rule $l \to r[\![C]\!]$ from $\mathcal{R}$ (Here, ${\downarrow}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}}$ means normalization w.r.t. $\mathcal{S}$)

Rewrite Relation Illustrated

$$\mathcal{E}: \quad u \cup v \approx v \cup u \qquad u \cup (v \cup w) \approx (u \cup v) \cup w$$

$$\mathcal{S}: \quad u \cup u \to u \qquad u \cup \emptyset \to u$$

- Reduce $t = \mathsf{msort}(\langle 1 \rangle \cup (\langle 3 \rangle \cup \langle 1 \rangle))$ using $\xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$
- Use $\mathsf{msort}(x \cup y) \to \mathsf{merge}(\mathsf{msort}(x), \mathsf{msort}(y))$ with $\sigma = \{x \mapsto \langle 1 \rangle, y \mapsto \langle 3 \rangle\}$
- $t \xrightarrow{!}{}^{<\varepsilon}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} \mathsf{msort}(\langle 1 \rangle \cup \langle 3 \rangle) \sim^{<\varepsilon}_{\mathcal{E}\cup\mathcal{PA}} \mathsf{msort}(x \cup y)\sigma$
- Thus $t \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} \mathsf{merge}(\mathsf{msort}(\langle 1 \rangle), \mathsf{msort}(\langle 3 \rangle)) = t'$
- $t' \xrightarrow{\mathcal{S}}{}^{+}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} \mathsf{merge}(\mathsf{cons}(1, \mathsf{nil}), \mathsf{cons}(3, \mathsf{nil}))$
- $\mathsf{merge}(\mathsf{cons}(x, xs), \mathsf{cons}(y, ys)) \to \mathsf{cons}(x, \mathsf{merge}(xs, \mathsf{cons}(y, ys)))[\![x \not> y]\!]$ with $\sigma = \{x \mapsto 1, xs \mapsto \mathsf{nil}, y \mapsto 3, ys \mapsto \mathsf{nil}\}$
- $\mathsf{merge}(\mathsf{cons}(1, \mathsf{nil}), \mathsf{cons}(3, \mathsf{nil})) \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} \mathsf{cons}(1, \mathsf{merge}(\mathsf{nil}, \mathsf{cons}(3, \mathsf{nil})))$ because $(x \not> y)\sigma = 1 \not> 3$ is $\mathcal{PA}$-valid
- One further $\xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$ reduction yields $\mathsf{cons}(1, \mathsf{cons}(3, \mathsf{nil}))$

Rewrite Relation Illustrated

$$\mathcal{E}: \quad u \cup v \approx v \cup u \qquad u \cup (v \cup w) \approx (u \cup v) \cup w$$

$$\mathcal{S}: \quad u \cup u \to u \qquad u \cup \emptyset \to u$$

- Reduce $t = \mathsf{msort}(\langle 1 \rangle \cup (\langle 3 \rangle \cup \langle 1 \rangle))$ using $\xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$
- Use $\mathsf{msort}(x \cup y) \to \mathsf{merge}(\mathsf{msort}(x), \mathsf{msort}(y))$ with $\sigma = \{x \mapsto \langle 1 \rangle, y \mapsto \langle 3 \rangle\}$
- $t \xrightarrow{!}{}^{<\varepsilon}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} \mathsf{msort}(\langle 1 \rangle \cup \langle 3 \rangle) \sim^{<\varepsilon}_{\mathcal{E}\cup\mathcal{PA}} \mathsf{msort}(x \cup y)\sigma$
- Thus $t \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} \mathsf{merge}(\mathsf{msort}(\langle 1 \rangle), \mathsf{msort}(\langle 3 \rangle)) = t'$
- $t' \xrightarrow{\mathcal{S}}{}^{+}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} \mathsf{merge}(\mathsf{cons}(1, \mathsf{nil}), \mathsf{cons}(3, \mathsf{nil}))$
- $\mathsf{merge}(\mathsf{cons}(x, xs), \mathsf{cons}(y, ys)) \to \mathsf{cons}(x, \mathsf{merge}(xs, \mathsf{cons}(y, ys)))[\![x \not> y]\!]$ with $\sigma = \{x \mapsto 1, xs \mapsto \mathsf{nil}, y \mapsto 3, ys \mapsto \mathsf{nil}\}$
- $\mathsf{merge}(\mathsf{cons}(1, \mathsf{nil}), \mathsf{cons}(3, \mathsf{nil})) \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$ $\mathsf{cons}(1, \mathsf{merge}(\mathsf{nil}, \mathsf{cons}(3, \mathsf{nil})))$ because $(x \not> y)\sigma = 1 \not> 3$ is $\mathcal{PA}$-valid
- One further $\xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$ reduction yields $\mathsf{cons}(1, \mathsf{cons}(3, \mathsf{nil}))$

Rewrite Relation Illustrated

$$\mathcal{E}: \quad u \cup v \approx v \cup u \qquad\qquad \mathcal{S}: \quad u \cup u \to u$$
$$u \cup (v \cup w) \approx (u \cup v) \cup w \qquad\qquad u \cup \emptyset \to u$$

- Reduce $t = \mathsf{msort}(\langle 1 \rangle \cup (\langle 3 \rangle \cup \langle 1 \rangle))$ using $\xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$
- Use $\mathsf{msort}(x \cup y) \to \mathsf{merge}(\mathsf{msort}(x), \mathsf{msort}(y))$ with $\sigma = \{x \mapsto \langle 1 \rangle, y \mapsto \langle 3 \rangle\}$
- $t \xrightarrow{!}{}^{<\varepsilon}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} \mathsf{msort}(\langle 1 \rangle \cup \langle 3 \rangle) \sim^{<\varepsilon}_{\mathcal{E}\cup\mathcal{PA}} \mathsf{msort}(x \cup y)\sigma$
- Thus $t \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} \mathsf{merge}(\mathsf{msort}(\langle 1 \rangle), \mathsf{msort}(\langle 3 \rangle)) = t'$
- $t' \xrightarrow{\mathcal{S}}{}^{+}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} \mathsf{merge}(\mathsf{cons}(1, \mathsf{nil}), \mathsf{cons}(3, \mathsf{nil}))$
- $\mathsf{merge}(\mathsf{cons}(x, xs), \mathsf{cons}(y, ys)) \to \mathsf{cons}(x, \mathsf{merge}(xs, \mathsf{cons}(y, ys)))[\![x \not> y]\!]$ with $\sigma = \{x \mapsto 1, xs \mapsto \mathsf{nil}, y \mapsto 3, ys \mapsto \mathsf{nil}\}$
- $\mathsf{merge}(\mathsf{cons}(1, \mathsf{nil}), \mathsf{cons}(3, \mathsf{nil})) \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$ $\mathsf{cons}(1, \mathsf{merge}(\mathsf{nil}, \mathsf{cons}(3, \mathsf{nil})))$ because $(x \not> y)\sigma = 1 \not> 3$ is $\mathcal{PA}$-valid
- One further $\xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$ reduction yields $\mathsf{cons}(1, \mathsf{cons}(3, \mathsf{nil}))$

Rewrite Relation Illustrated

$\mathcal{E}$: $u \cup v \approx v \cup u$
$u \cup (v \cup w) \approx (u \cup v) \cup w$

$\mathcal{S}$: $u \cup u \to u$
$u \cup \emptyset \to u$

- Reduce $t = \mathsf{msort}(\langle 1 \rangle \cup (\langle 3 \rangle \cup \langle 1 \rangle))$ using $\xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$
- Use $\mathsf{msort}(x \cup y) \to \mathsf{merge}(\mathsf{msort}(x), \mathsf{msort}(y))$ with $\sigma = \{x \mapsto \langle 1 \rangle, y \mapsto \langle 3 \rangle\}$
- $t \xrightarrow{!}{}^{<\varepsilon}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} \mathsf{msort}(\langle 1 \rangle \cup \langle 3 \rangle) \sim^{<\varepsilon}_{\mathcal{E}\cup\mathcal{PA}} \mathsf{msort}(x \cup y)\sigma$
- Thus $t \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} \mathsf{merge}(\mathsf{msort}(\langle 1 \rangle), \mathsf{msort}(\langle 3 \rangle)) = t'$
- $t' \xrightarrow{\mathcal{S}}{}^{+}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} \mathsf{merge}(\mathsf{cons}(1, \mathsf{nil}), \mathsf{cons}(3, \mathsf{nil}))$
- $\mathsf{merge}(\mathsf{cons}(x, xs), \mathsf{cons}(y, ys)) \to \mathsf{cons}(x, \mathsf{merge}(xs, \mathsf{cons}(y, ys)))[\![x \not> y]\!]$ with $\sigma = \{x \mapsto 1, xs \mapsto \mathsf{nil}, y \mapsto 3, ys \mapsto \mathsf{nil}\}$
- $\mathsf{merge}(\mathsf{cons}(1, \mathsf{nil}), \mathsf{cons}(3, \mathsf{nil})) \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$ $\mathsf{cons}(1, \mathsf{merge}(\mathsf{nil}, \mathsf{cons}(3, \mathsf{nil})))$ because $(x \not> y)\sigma = 1 \not> 3$ is $\mathcal{PA}$-valid
- One further $\xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$ reduction yields $\mathsf{cons}(1, \mathsf{cons}(3, \mathsf{nil}))$

Rewrite Relation Illustrated

$\mathcal{E}$: $u \cup v \approx v \cup u$
$u \cup (v \cup w) \approx (u \cup v) \cup w$

$\mathcal{S}$: $u \cup u \to u$
$u \cup \emptyset \to u$

- Reduce $t = \mathsf{msort}(\langle 1 \rangle \cup (\langle 3 \rangle \cup \langle 1 \rangle))$ using $\xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$
- Use $\mathsf{msort}(x \cup y) \to \mathsf{merge}(\mathsf{msort}(x), \mathsf{msort}(y))$ with $\sigma = \{x \mapsto \langle 1 \rangle, y \mapsto \langle 3 \rangle\}$
- $t \xrightarrow{!}{}^{<\varepsilon}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} \mathsf{msort}(\langle 1 \rangle \cup \langle 3 \rangle) \sim^{<\varepsilon}_{\mathcal{E}\cup\mathcal{PA}} \mathsf{msort}(x \cup y)\sigma$
- Thus $t \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} \mathsf{merge}(\mathsf{msort}(\langle 1 \rangle), \mathsf{msort}(\langle 3 \rangle)) = t'$
- $t' \xrightarrow{\mathcal{S}}{}^{+}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} \mathsf{merge}(\mathsf{cons}(1, \mathsf{nil}), \mathsf{cons}(3, \mathsf{nil}))$
- $\mathsf{merge}(\mathsf{cons}(x, xs), \mathsf{cons}(y, ys)) \to \mathsf{cons}(x, \mathsf{merge}(xs, \mathsf{cons}(y, ys)))[\![x \not> y]\!]$ with $\sigma = \{x \mapsto 1, xs \mapsto \mathsf{nil}, y \mapsto 3, ys \mapsto \mathsf{nil}\}$
- $\mathsf{merge}(\mathsf{cons}(1, \mathsf{nil}), \mathsf{cons}(3, \mathsf{nil})) \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$ $\mathsf{cons}(1, \mathsf{merge}(\mathsf{nil}, \mathsf{cons}(3, \mathsf{nil})))$ because $(x \not> y)\sigma = 1 \not> 3$ is $\mathcal{PA}$-valid
- One further $\xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$ reduction yields $\mathsf{cons}(1, \mathsf{cons}(3, \mathsf{nil}))$

Rewrite Relation Illustrated

$\mathcal{E}$: $u \cup v \approx v \cup u$ $\quad$ $\mathcal{S}$: $u \cup u \to u$

$u \cup (v \cup w) \approx (u \cup v) \cup w$ $\quad$ $u \cup \emptyset \to u$

- Reduce $t = \mathsf{msort}(\langle 1 \rangle \cup (\langle 3 \rangle \cup \langle 1 \rangle))$ using $\xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$
- Use $\mathsf{msort}(x \cup y) \to \mathsf{merge}(\mathsf{msort}(x), \mathsf{msort}(y))$ with $\sigma = \{x \mapsto \langle 1 \rangle, y \mapsto \langle 3 \rangle\}$
- $t \xrightarrow{!}{}^{<\varepsilon}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} \mathsf{msort}(\langle 1 \rangle \cup \langle 3 \rangle) \sim^{<\varepsilon}_{\mathcal{E}\cup\mathcal{PA}} \mathsf{msort}(x \cup y)\sigma$
- Thus $t \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} \mathsf{merge}(\mathsf{msort}(\langle 1 \rangle), \mathsf{msort}(\langle 3 \rangle)) = t'$
- $t' \xrightarrow{\mathcal{S}}{}^{+}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} \mathsf{merge}(\mathsf{cons}(1, \mathsf{nil}), \mathsf{cons}(3, \mathsf{nil}))$
- $\mathsf{merge}(\mathsf{cons}(x, xs), \mathsf{cons}(y, ys)) \to \mathsf{cons}(x, \mathsf{merge}(xs, \mathsf{cons}(y, ys)))[\![x \not> y]\!]$ with $\sigma = \{x \mapsto 1, xs \mapsto \mathsf{nil}, y \mapsto 3, ys \mapsto \mathsf{nil}\}$
- $\mathsf{merge}(\mathsf{cons}(1, \mathsf{nil}), \mathsf{cons}(3, \mathsf{nil})) \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$ $\mathsf{cons}(1, \mathsf{merge}(\mathsf{nil}, \mathsf{cons}(3, \mathsf{nil})))$ because $(x \not> y)\sigma = 1 \not> 3$ is $\mathcal{PA}$-valid
- One further $\xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$ reduction yields $\mathsf{cons}(1, \mathsf{cons}(3, \mathsf{nil}))$

Rewrite Relation Illustrated

$\mathcal{E}$: $u \cup v \approx v \cup u$, $u \cup (v \cup w) \approx (u \cup v) \cup w$

$\mathcal{S}$: $u \cup u \to u$, $u \cup \emptyset \to u$

- Reduce $t = \text{msort}(\langle 1 \rangle \cup (\langle 3 \rangle \cup \langle 1 \rangle))$ using $\xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$
- Use $\text{msort}(x \cup y) \to \text{merge}(\text{msort}(x), \text{msort}(y))$ with $\sigma = \{x \mapsto \langle 1 \rangle, y \mapsto \langle 3 \rangle\}$
- $t \xrightarrow{!}{}^{<\varepsilon}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} \text{msort}(\langle 1 \rangle \cup \langle 3 \rangle) \sim^{<\varepsilon}_{\mathcal{E}\cup\mathcal{PA}} \text{msort}(x \cup y)\sigma$
- Thus $t \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} \text{merge}(\text{msort}(\langle 1 \rangle), \text{msort}(\langle 3 \rangle)) = t'$
- $t' \xrightarrow{\mathcal{S}}{}^{+}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} \text{merge}(\text{cons}(1, \text{nil}), \text{cons}(3, \text{nil}))$
- $\text{merge}(\text{cons}(x, xs), \text{cons}(y, ys)) \to \text{cons}(x, \text{merge}(xs, \text{cons}(y, ys)))[\![x \not> y]\!]$ with $\sigma = \{x \mapsto 1, xs \mapsto \text{nil}, y \mapsto 3, ys \mapsto \text{nil}\}$
- $\text{merge}(\text{cons}(1, \text{nil}), \text{cons}(3, \text{nil})) \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$ $\text{cons}(1, \text{merge}(\text{nil}, \text{cons}(3, \text{nil})))$ because $(x \not> y)\sigma = 1 \not> 3$ is $\mathcal{PA}$-valid
- One further $\xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$ reduction yields $\text{cons}(1, \text{cons}(3, \text{nil}))$

Rewrite Relation Illustrated

$$\mathcal{E}: \quad u \cup v \approx v \cup u \qquad u \cup (v \cup w) \approx (u \cup v) \cup w$$

$$\mathcal{S}: \quad u \cup u \to u \qquad u \cup \emptyset \to u$$

- Reduce $t = \mathsf{msort}(\langle 1\rangle \cup (\langle 3\rangle \cup \langle 1\rangle))$ using $\xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$
- Use $\mathsf{msort}(x \cup y) \to \mathsf{merge}(\mathsf{msort}(x), \mathsf{msort}(y))$ with $\sigma = \{x \mapsto \langle 1\rangle, y \mapsto \langle 3\rangle\}$
- $t \xrightarrow{!}{}^{<\varepsilon}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} \mathsf{msort}(\langle 1\rangle \cup \langle 3\rangle) \sim^{<\varepsilon}_{\mathcal{E}\cup\mathcal{PA}} \mathsf{msort}(x \cup y)\sigma$
- Thus $t \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} \mathsf{merge}(\mathsf{msort}(\langle 1\rangle), \mathsf{msort}(\langle 3\rangle)) = t'$
- $t' \xrightarrow{\mathcal{S}}{}^{+}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} \mathsf{merge}(\mathsf{cons}(1, \mathsf{nil}), \mathsf{cons}(3, \mathsf{nil}))$
- $\mathsf{merge}(\mathsf{cons}(x, xs), \mathsf{cons}(y, ys)) \to \mathsf{cons}(x, \mathsf{merge}(xs, \mathsf{cons}(y, ys)))[\![x \not> y]\!]$ with $\sigma = \{x \mapsto 1, xs \mapsto \mathsf{nil}, y \mapsto 3, ys \mapsto \mathsf{nil}\}$
- $\mathsf{merge}(\mathsf{cons}(1, \mathsf{nil}), \mathsf{cons}(3, \mathsf{nil})) \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$ $\mathsf{cons}(1, \mathsf{merge}(\mathsf{nil}, \mathsf{cons}(3, \mathsf{nil})))$ because $(x \not> y)\sigma = 1 \not> 3$ is $\mathcal{PA}$-valid
- One further $\xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$ reduction yields $\mathsf{cons}(1, \mathsf{cons}(3, \mathsf{nil}))$

Dependency Pairs

- Powerful method for automated termination proofs
- Virtually all automated termination tools for rewriting use dependency pairs
- Initially only for ordinary rewriting (Arts & Giesl, 1997)
- Extended to AC-rewriting (Marché & Urbain, 1998; Kusakari & Toyama, 1998)
- Extended to equational rewriting where $\mathcal{E}$ is regular, linear, and collapse-free (but may contain defined symbols) (Giesl & Kapur, 2001)
- Extended to rewriting with semantic data structures, but without built-in natural numbers (CADE, 2007)

Dependency Pairs

- Powerful method for automated termination proofs
- Virtually all automated termination tools for rewriting use dependency pairs
- Initially only for ordinary rewriting (Arts & Giesl, 1997)
- Extended to AC-rewriting (Marché & Urbain, 1998; Kusakari & Toyama, 1998)
- Extended to equational rewriting where $\mathcal{E}$ is regular, linear, and collapse-free (but may contain defined symbols) (Giesl & Kapur, 2001)
- Extended to rewriting with semantic data structures, but without built-in natural numbers (CADE, 2007)

Dependency Pairs

- Powerful method for automated termination proofs
- Virtually all automated termination tools for rewriting use dependency pairs
- Initially only for ordinary rewriting (Arts & Giesl, 1997)
- Extended to AC-rewriting (Marché & Urbain, 1998; Kusakari & Toyama, 1998)
- Extended to equational rewriting where $\mathcal{E}$ is regular, linear, and collapse-free (but may contain defined symbols) (Giesl & Kapur, 2001)
- Extended to rewriting with semantic data structures, but without built-in natural numbers (CADE, 2007)

Dependency Pairs

- Powerful method for automated termination proofs
- Virtually all automated termination tools for rewriting use dependency pairs
- Initially only for ordinary rewriting (Arts & Giesl, 1997)
- Extended to AC-rewriting (Marché & Urbain, 1998; Kusakari & Toyama, 1998)
- Extended to equational rewriting where $\mathcal{E}$ is regular, linear, and collapse-free (but may contain defined symbols) (Giesl & Kapur, 2001)
- Extended to rewriting with semantic data structures, but without built-in natural numbers (CADE, 2007)

Dependency Pairs

- Powerful method for automated termination proofs
- Virtually all automated termination tools for rewriting use dependency pairs
- Initially only for ordinary rewriting (Arts & Giesl, 1997)
- Extended to AC-rewriting (Marché & Urbain, 1998; Kusakari & Toyama, 1998)
- Extended to equational rewriting where $\mathcal{E}$ is regular, linear, and collapse-free (but may contain defined symbols) (Giesl & Kapur, 2001)
- Extended to rewriting with semantic data structures, but without built-in natural numbers (CADE, 2007)

Dependency Pairs

- Powerful method for automated termination proofs
- Virtually all automated termination tools for rewriting use dependency pairs
- Initially only for ordinary rewriting (Arts & Giesl, 1997)
- Extended to AC-rewriting (Marché & Urbain, 1998; Kusakari & Toyama, 1998)
- Extended to equational rewriting where $\mathcal{E}$ is regular, linear, and collapse-free (but may contain defined symbols) (Giesl & Kapur, 2001)
- Extended to rewriting with semantic data structures, but without built-in natural numbers (CADE, 2007)

Our Contributions

- Extension to rewriting with semantic data structures and built-in natural numbers, i.e., constraints on natural numbers are allowed
- Properties on semantic data structures may be collapsing
- Practical: $\mathcal{E}$, $\mathcal{S}$, and $\mathcal{R}$ are taken "as is"
  No special pre-processing needed

Our Contributions

- Extension to rewriting with semantic data structures and built-in natural numbers, i.e., constraints on natural numbers are allowed
- Properties on semantic data structures may be collapsing
- Practical: $\mathcal{E}$, $\mathcal{S}$, and $\mathcal{R}$ are taken "as is"
  No special pre-processing needed

Our Contributions

- Extension to rewriting with semantic data structures and built-in natural numbers, i.e., constraints on natural numbers are allowed
- Properties on semantic data structures may be collapsing
- Practical: $\mathcal{E}$, $\mathcal{S}$, and $\mathcal{R}$ are taken “as is”
  No special pre-processing needed

Key Result

- As usual, a dependency pair corresponds to a call to a defined function:
  $\mathrm{DP}(\mathcal{R}) = \{ l^\sharp \to t^\sharp [\![C]\!] \mid l \to r [\![C]\!] \in \mathcal{R},\ t$ is a subterm of r with defined root symbol $\}$
- Note that the dependency pairs inherit the constraints
- A sequence $s_1 \to t_1 [\![C_1]\!], s_2 \to t_2 [\![C_2]\!], \ldots$ of dependency pairs from a set $\mathcal{P}$ is a $(\mathcal{P}, \mathcal{R})$-chain iff there exists a σ such that

$$t_i \sigma \xrightarrow{\mathcal{S}}{}^{*}_{\mathcal{PA} \| \mathcal{E} \backslash \mathcal{R}} \circ \to^{!}_{\mathcal{PA} \| \mathcal{E} \backslash \mathcal{S}} \circ \sim_{\mathcal{E} \cup \mathcal{PA}} s_{i+1} \sigma$$

  and $C_i \sigma$ is $\mathcal{PA}$-valid for all $i \geq 0$
- Key result: $\xrightarrow{\mathcal{S}}_{\mathcal{PA} \| \mathcal{E} \backslash \mathcal{R}}$ is terminating if there are no infinite $(\mathrm{DP}(\mathcal{R}), \mathcal{R})$-chains

Key Result

- As usual, a dependency pair corresponds to a call to a defined function:

  $$\mathrm{DP}(\mathcal{R}) = \{ l^\sharp \to t^\sharp[\![C]\!] \mid l \to r[\![C]\!] \in \mathcal{R}, t \text{ is a subterm of } r \text{ with defined root symbol } \}$$

- Note that the dependency pairs inherit the constraints
- A sequence $s_1 \to t_1[\![C_1]\!], s_2 \to t_2[\![C_2]\!], \ldots$ of dependency pairs from a set $\mathcal{P}$ is a $(\mathcal{P}, \mathcal{R})$-chain iff there exists a σ such that

  $$t_i\sigma \xrightarrow{\mathcal{S}}{}^*_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} \circ \to^!_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} \circ \sim_{\mathcal{E}\cup\mathcal{PA}} s_{i+1}\sigma$$

  and $C_i\sigma$ is $\mathcal{PA}$-valid for all $i \geq 0$
- Key result: $\xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$ is terminating if there are no infinite $(\mathrm{DP}(\mathcal{R}), \mathcal{R})$-chains

Key Result

- As usual, a dependency pair corresponds to a call to a defined function:

  $\mathrm{DP}(\mathcal{R}) = \{ l^\sharp \to t^\sharp[\![C]\!] \mid l \to r[\![C]\!] \in \mathcal{R},\ t$ is a subterm of r with defined root symbol $\}$

- Note that the dependency pairs inherit the constraints
- A sequence $s_1 \to t_1[\![C_1]\!], s_2 \to t_2[\![C_2]\!], \ldots$ of dependency pairs from a set $\mathcal{P}$ is a $(\mathcal{P}, \mathcal{R})$-chain iff there exists a σ such that

  $$t_i\sigma \xrightarrow{\mathcal{S}}{}^{*}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} \circ \to^{!}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} \circ \sim_{\mathcal{E}\cup\mathcal{PA}} s_{i+1}\sigma$$

  and $C_i\sigma$ is $\mathcal{PA}$-valid for all $i \geq 0$

- Key result: $\xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$ is terminating if there are no infinite $(\mathrm{DP}(\mathcal{R}), \mathcal{R})$-chains

Key Result

- As usual, a dependency pair corresponds to a call to a defined function:

  $\mathrm{DP}(\mathcal{R}) = \{ l^\sharp \to t^\sharp[\![C]\!] \mid l \to r[\![C]\!] \in \mathcal{R}, t$ is a subterm of r with defined root symbol $\}$

- Note that the dependency pairs inherit the constraints
- A sequence $s_1 \to t_1[\![C_1]\!], s_2 \to t_2[\![C_2]\!], \ldots$ of dependency pairs from a set $\mathcal{P}$ is a $(\mathcal{P}, \mathcal{R})$-chain iff there exists a σ such that

  $$t_i\sigma \xrightarrow{\mathcal{S}}{}^{*}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} \circ \to^{!}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} \circ \sim_{\mathcal{E}\cup\mathcal{PA}} s_{i+1}\sigma$$

  and $C_i\sigma$ is $\mathcal{PA}$-valid for all $i \geq 0$

- Key result: $\xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$ is terminating if there are no infinite $(\mathrm{DP}(\mathcal{R}), \mathcal{R})$-chains

Example

- Sets are modelled using $\emptyset$ and ins
- $\mathsf{nats}(x, y)$ computes $\{z \mid x \leq z \leq y\}$
- $$\begin{array}{llcll} \mathcal{E}: & \mathsf{ins}(x, \mathsf{ins}(y, zs)) & \approx & \mathsf{ins}(y, \mathsf{ins}(x, zs)) & \\ \mathcal{S}: & \mathsf{ins}(x, \mathsf{ins}(y, zs)) & \to & \mathsf{ins}(x, ys) & [\![x \simeq y]\!] \\ \mathcal{R}: & \mathsf{inc}(\emptyset) & \to & \emptyset & \\ & \mathsf{inc}(\mathsf{ins}(x, ys)) & \to & \mathsf{ins}(x+1, \mathsf{inc}(ys)) & \\ & \mathsf{nats}(0, 0) & \to & \mathsf{ins}(0, \emptyset) & \\ & \mathsf{nats}(0, y+1) & \to & \mathsf{ins}(0, \mathsf{nats}(1, y+1)) & \\ & \mathsf{nats}(x+1, 0) & \to & \emptyset & \\ & \mathsf{nats}(x+1, y+1) & \to & \mathsf{inc}(\mathsf{nats}(x, y)) & \end{array}$$
- $$\mathsf{DP}(\mathcal{R}) = \{\ \mathsf{inc}^\sharp(\mathsf{ins}(x, ys)) \to \mathsf{inc}^\sharp(ys) \\ \mathsf{nats}^\sharp(0, y+1) \to \mathsf{nats}^\sharp(1, y+1) \\ \mathsf{nats}^\sharp(x+1, y+1) \to \mathsf{inc}^\sharp(\mathsf{nats}(x, y)) \\ \mathsf{nats}^\sharp(x+1, y+1) \to \mathsf{nats}^\sharp(x, y)\ \}$$

Example

- Sets are modelled using $\emptyset$ and ins
- nats(x, y) computes $\{z \mid x \leq z \leq y\}$
- $$\begin{array}{lrcll} \mathcal{E}: & \mathsf{ins}(x, \mathsf{ins}(y, zs)) & \approx & \mathsf{ins}(y, \mathsf{ins}(x, zs)) & \\ \mathcal{S}: & \mathsf{ins}(x, \mathsf{ins}(y, zs)) & \to & \mathsf{ins}(x, ys) & [\![x \simeq y]\!] \\ \mathcal{R}: & \mathsf{inc}(\emptyset) & \to & \emptyset & \\ & \mathsf{inc}(\mathsf{ins}(x, ys)) & \to & \mathsf{ins}(x + 1, \mathsf{inc}(ys)) & \\ & \mathsf{nats}(0, 0) & \to & \mathsf{ins}(0, \emptyset) & \\ & \mathsf{nats}(0, y + 1) & \to & \mathsf{ins}(0, \mathsf{nats}(1, y + 1)) & \\ & \mathsf{nats}(x + 1, 0) & \to & \emptyset & \\ & \mathsf{nats}(x + 1, y + 1) & \to & \mathsf{inc}(\mathsf{nats}(x, y)) & \end{array}$$
- $$\begin{array}{ll} \mathsf{DP}(\mathcal{R}) = \{ & \mathsf{inc}^\sharp(\mathsf{ins}(x, ys)) \to \mathsf{inc}^\sharp(ys) \\ & \mathsf{nats}^\sharp(0, y + 1) \to \mathsf{nats}^\sharp(1, y + 1) \\ & \mathsf{nats}^\sharp(x + 1, y + 1) \to \mathsf{inc}^\sharp(\mathsf{nats}(x, y)) \\ & \mathsf{nats}^\sharp(x + 1, y + 1) \to \mathsf{nats}^\sharp(x, y) \quad \} \end{array}$$

Dependency Pair Framework

- Flexible framework for showing absence of infinite chains
- Originally developed for ordinary rewriting (Giesl *et. al.*, 2005)
- DP problem $(\mathcal{P}, \mathcal{R})$:
  - a set $\mathcal{P}$ of dependency pairs
  - a finite set $\mathcal{R}$ of rewrite rules with constraints
- $(\mathcal{P}, \mathcal{R})$ is finite iff there are no infinite $(\mathcal{P}, \mathcal{R})$-chains
- A DP processor is a function mapping a DP problem to a finite set of DP problems such that the input is finite if all outputs are finite
- Method:
  - start with the DP problem $(\mathrm{DP}(\mathcal{R}), \mathcal{R})$
  - apply DP processors until all DP problems obtained are trivial $(\mathcal{P} = \emptyset)$

Dependency Pair Framework

- Flexible framework for showing absence of infinite chains
- Originally developed for ordinary rewriting (Giesl *et. al.*, 2005)
- DP problem $(\mathcal{P}, \mathcal{R})$:
  - a set $\mathcal{P}$ of dependency pairs
  - a finite set $\mathcal{R}$ of rewrite rules with constraints
- $(\mathcal{P}, \mathcal{R})$ is finite iff there are no infinite $(\mathcal{P}, \mathcal{R})$-chains
- A DP processor is a function mapping a DP problem to a finite set of DP problems such that the input is finite if all outputs are finite
- Method:
  - start with the DP problem $(\mathrm{DP}(\mathcal{R}), \mathcal{R})$
  - apply DP processors until all DP problems obtained are trivial $(\mathcal{P} = \emptyset)$

Dependency Pair Framework

- Flexible framework for showing absence of infinite chains
- Originally developed for ordinary rewriting (Giesl *et. al.*, 2005)
- DP problem $(\mathcal{P}, \mathcal{R})$:
  - a set $\mathcal{P}$ of dependency pairs
  - a finite set $\mathcal{R}$ of rewrite rules with constraints
- $(\mathcal{P}, \mathcal{R})$ is finite iff there are no infinite $(\mathcal{P}, \mathcal{R})$-chains
- A DP processor is a function mapping a DP problem to a finite set of DP problems such that the input is finite if all outputs are finite
- Method:
  - start with the DP problem $(\mathrm{DP}(\mathcal{R}), \mathcal{R})$
  - apply DP processors until all DP problems obtained are trivial $(\mathcal{P} = \emptyset)$

Dependency Pair Framework

- Flexible framework for showing absence of infinite chains
- Originally developed for ordinary rewriting (Giesl *et. al.*, 2005)
- DP problem $(\mathcal{P}, \mathcal{R})$:
  - a set $\mathcal{P}$ of dependency pairs
  - a finite set $\mathcal{R}$ of rewrite rules with constraints
- $(\mathcal{P}, \mathcal{R})$ is finite iff there are no infinite $(\mathcal{P}, \mathcal{R})$-chains
- A DP processor is a function mapping a DP problem to a finite set of DP problems such that the input is finite if all outputs are finite
- Method:
  - start with the DP problem $(\mathsf{DP}(\mathcal{R}), \mathcal{R})$
  - apply DP processors until all DP problems obtained are trivial $(\mathcal{P} = \emptyset)$

Dependency Pair Framework

- Flexible framework for showing absence of infinite chains
- Originally developed for ordinary rewriting (Giesl *et. al.*, 2005)
- DP problem $(\mathcal{P}, \mathcal{R})$:
  - a set $\mathcal{P}$ of dependency pairs
  - a finite set $\mathcal{R}$ of rewrite rules with constraints
- $(\mathcal{P}, \mathcal{R})$ is finite iff there are no infinite $(\mathcal{P}, \mathcal{R})$-chains
- A DP processor is a function mapping a DP problem to a finite set of DP problems such that the input is finite if all outputs are finite
- Method:
  - start with the DP problem $(\mathrm{DP}(\mathcal{R}), \mathcal{R})$
  - apply DP processors until all DP problems obtained are trivial $(\mathcal{P} = \emptyset)$

Dependency Pair Framework

- Flexible framework for showing absence of infinite chains
- Originally developed for ordinary rewriting (Giesl *et. al.*, 2005)
- DP problem $(\mathcal{P}, \mathcal{R})$:
  - a set $\mathcal{P}$ of dependency pairs
  - a finite set $\mathcal{R}$ of rewrite rules with constraints
- $(\mathcal{P}, \mathcal{R})$ is finite iff there are no infinite $(\mathcal{P}, \mathcal{R})$-chains
- A DP processor is a function mapping a DP problem to a finite set of DP problems such that the input is finite if all outputs are finite
- Method:
  - start with the DP problem $(\mathrm{DP}(\mathcal{R}), \mathcal{R})$
  - apply DP processors until all DP problems obtained are trivial $(\mathcal{P} = \emptyset)$

A Collection of Sound DP Processors

We show soundness of several DP processors for our class of rewrite systems:

- A DP processor based on (approximated) dependency graphs
- A DP processor based on the subterm criterion
- A DP processor based on well-founded relations

A Collection of Sound DP Processors

We show soundness of several DP processors for our class of rewrite systems:

- A DP processor based on (approximated) dependency graphs
- A DP processor based on the subterm criterion
- A DP processor based on well-founded relations

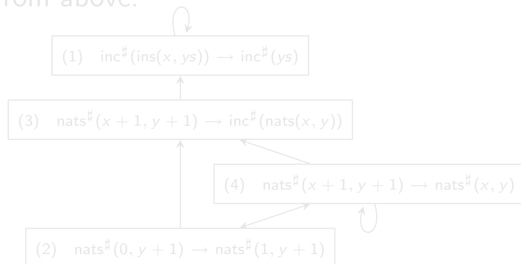
A Collection of Sound DP Processors

We show soundness of several DP processors for our class of rewrite systems:

- A DP processor based on (approximated) dependency graphs
- A DP processor based on the subterm criterion
- A DP processor based on well-founded relations

Dependency Graphs

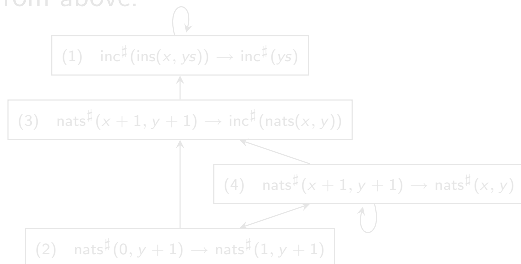
- $(\mathcal{P}, \mathcal{R})$-dependency graph:
  - nodes are the dependency pairs from $\mathcal{P}$
  - edge from $s \to t⟦C⟧$ to $u \to v⟦D⟧$ iff $s \to t⟦D⟧,\ u \to v⟦D⟧$ is a chain
- Can only be approximated in general
- For $(\mathrm{DP}(\mathcal{R}), \mathcal{R})$ from above:



- Every infinite chain has an infinite tail that stays in some SCC
- $\mathrm{Proc}(\mathrm{DP}(\mathcal{R}), \mathcal{R}) = \{(\{(1)\}, \mathcal{R}), (\{(2), (4)\}, \mathcal{R})\}$

Dependency Graphs

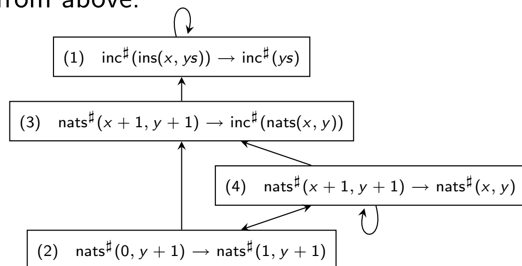
- $(\mathcal{P}, \mathcal{R})$-dependency graph:
  - nodes are the dependency pairs from $\mathcal{P}$
  - edge from $s \to t[\![C]\!]$ to $u \to v[\![D]\!]$ iff $s \to t[\![D]\!],\ u \to v[\![D]\!]$ is a chain
- Can only be approximated in general
- For $(\mathrm{DP}(\mathcal{R}), \mathcal{R})$ from above:



- Every infinite chain has an infinite tail that stays in some SCC
- $\mathrm{Proc}(\mathrm{DP}(\mathcal{R}), \mathcal{R}) = \{(\{(1)\}, \mathcal{R}), (\{(2), (4)\}, \mathcal{R})\}$

Dependency Graphs

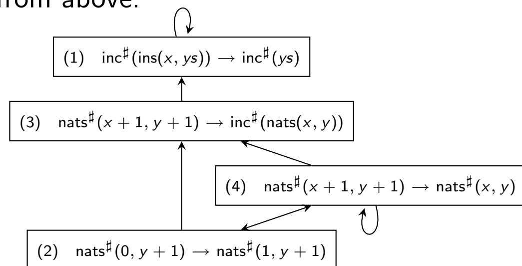
- $(\mathcal{P}, \mathcal{R})$-dependency graph:
  - nodes are the dependency pairs from $\mathcal{P}$
  - edge from $s \to t[\![C]\!]$ to $u \to v[\![D]\!]$ iff $s \to t[\![D]\!],\ u \to v[\![D]\!]$ is a chain
- Can only be approximated in general
- For $(\mathsf{DP}(\mathcal{R}), \mathcal{R})$ from above:



- Every infinite chain has an infinite tail that stays in some SCC
- $\mathrm{Proc}(\mathsf{DP}(\mathcal{R}), \mathcal{R}) = \{(\{(1)\}, \mathcal{R}), (\{(2), (4)\}, \mathcal{R})\}$

Dependency Graphs

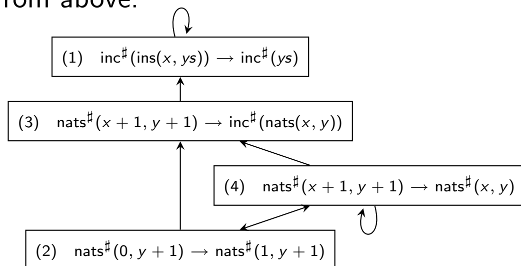
- $(\mathcal{P}, \mathcal{R})$-dependency graph:
  - nodes are the dependency pairs from $\mathcal{P}$
  - edge from $s \to t[\![C]\!]$ to $u \to v[\![D]\!]$ iff $s \to t[\![D]\!],\ u \to v[\![D]\!]$ is a chain
- Can only be approximated in general
- For $(\mathsf{DP}(\mathcal{R}), \mathcal{R})$ from above:



- Every infinite chain has an infinite tail that stays in some SCC
- $\mathsf{Proc}(\mathsf{DP}(\mathcal{R}), \mathcal{R}) = \{(\{(1)\}, \mathcal{R}), (\{(2), (4)\}, \mathcal{R})\}$

Dependency Graphs

- $(\mathcal{P}, \mathcal{R})$-dependency graph:
  - nodes are the dependency pairs from $\mathcal{P}$
  - edge from $s \to t[\![C]\!]$ to $u \to v[\![D]\!]$ iff $s \to t[\![D]\!],\ u \to v[\![D]\!]$ is a chain
- Can only be approximated in general
- For $(\mathrm{DP}(\mathcal{R}), \mathcal{R})$ from above:



- Every infinite chain has an infinite tail that stays in some SCC
- $\mathrm{Proc}(\mathrm{DP}(\mathcal{R}), \mathcal{R}) = \{(\{(1)\}, \mathcal{R}), (\{(2), (4)\}, \mathcal{R})\}$

Subterm Criterion

- Simple yet powerful DP processor (for ordinary rewriting due to Hirokawa & Middeldorp, 2004)
- We use the following strict subterm relations on (constrained) terms:
  - for s, t of sort $\mathtt{nat}$ we let $s[\![C]\!] \rhd_{\mathtt{nat}} t[\![C]\!]$ iff $C \Rightarrow s > t$ is $\mathcal{PA}$-valid
  - for s, t not of sort $\mathtt{nat}$ we let $s \rhd_{\mathtt{univ}} t$ iff $s \sim_{\mathcal{E} \cup \mathcal{PA}} \circ \rhd \circ \sim_{\mathcal{E} \cup \mathcal{PA}} t$
- Simple projection π: Assign to each $f^\sharp$ one of its arguments
- $\mathrm{Proc}(\mathcal{P}, \mathcal{R}) = \{(\mathcal{P} - \mathcal{P}', \mathcal{R})\}$, if $\rhd_{\mathtt{univ}}$ is well-founded and $\mathcal{P}' \subseteq \mathcal{P}$ such that
  - for all $s \to t[\![C]\!] \in \mathcal{P} - \mathcal{P}'$, either $\pi(s) \unrhd_{\mathtt{univ}} \pi(t)$ or $\pi(s)[\![C]\!] \unrhd_{\mathtt{nat}} \pi(t)[\![C]\!]$, and
  - for all $s \to t[\![C]\!] \in \mathcal{P}'$, either $\pi(s) \rhd_{\mathtt{univ}} \pi(t)$ or $\pi(s)[\![C]\!] \rhd_{\mathtt{nat}} \pi(t)[\![C]\!]$,
- Note that $\mathcal{R}$, $\mathcal{S}$, and $\mathcal{E}$ do not need to be considered

Subterm Criterion

- Simple yet powerful DP processor (for ordinary rewriting due to Hirokawa & Middeldorp, 2004)
- We use the following strict subterm relations on (constrained) terms:
    - for s, t of sort $\mathtt{nat}$ we let $s[\![C]\!] \rhd_{\mathtt{nat}} t[\![C]\!]$ iff $C \Rightarrow s > t$ is $\mathcal{PA}$-valid
    - for s, t not of sort $\mathtt{nat}$ we let $s \rhd_{\mathtt{univ}} t$ iff $s \sim_{\mathcal{E}\cup\mathcal{PA}} \circ \rhd \circ \sim_{\mathcal{E}\cup\mathcal{PA}} t$
- Simple projection π: Assign to each $f^\sharp$ one of its arguments
- $\mathrm{Proc}(\mathcal{P}, \mathcal{R}) = \{(\mathcal{P} - \mathcal{P}', \mathcal{R})\}$, if $\rhd_{\mathtt{univ}}$ is well-founded and $\mathcal{P}' \subseteq \mathcal{P}$ such that
    - for all $s \to t[\![C]\!] \in \mathcal{P} - \mathcal{P}'$, either $\pi(s) \unrhd_{\mathtt{univ}} \pi(t)$ or $\pi(s)[\![C]\!] \unrhd_{\mathtt{nat}} \pi(t)[\![C]\!]$, and
    - for all $s \to t[\![C]\!] \in \mathcal{P}'$, either $\pi(s) \rhd_{\mathtt{univ}} \pi(t)$ or $\pi(s)[\![C]\!] \rhd_{\mathtt{nat}} \pi(t)[\![C]\!]$,
- Note that $\mathcal{R}$, $\mathcal{S}$, and $\mathcal{E}$ do not need to be considered

Subterm Criterion

- Simple yet powerful DP processor (for ordinary rewriting due to Hirokawa & Middeldorp, 2004)
- We use the following strict subterm relations on (constrained) terms:
  - for s, t of sort $\mathtt{nat}$ we let $s[\![C]\!] \rhd_{\mathtt{nat}} t[\![C]\!]$ iff $C \Rightarrow s > t$ is $\mathcal{PA}$-valid
  - for s, t not of sort $\mathtt{nat}$ we let $s \rhd_{\mathtt{univ}} t$ iff $s \sim_{\mathcal{E} \cup \mathcal{PA}} \circ \rhd \circ \sim_{\mathcal{E} \cup \mathcal{PA}} t$
- Simple projection π: Assign to each $f^\sharp$ one of its arguments
- $\mathrm{Proc}(\mathcal{P}, \mathcal{R}) = \{(\mathcal{P} - \mathcal{P}', \mathcal{R})\}$, if $\rhd_{\mathtt{univ}}$ is well-founded and $\mathcal{P}' \subseteq \mathcal{P}$ such that
  - for all $s \to t[\![C]\!] \in \mathcal{P} - \mathcal{P}'$, either $\pi(s) \trianglerighteq_{\mathtt{univ}} \pi(t)$ or $\pi(s)[\![C]\!] \trianglerighteq_{\mathtt{nat}} \pi(t)[\![C]\!]$, and
  - for all $s \to t[\![C]\!] \in \mathcal{P}'$, either $\pi(s) \rhd_{\mathtt{univ}} \pi(t)$ or $\pi(s)[\![C]\!] \rhd_{\mathtt{nat}} \pi(t)[\![C]\!]$,
- Note that $\mathcal{R}$, $\mathcal{S}$, and $\mathcal{E}$ do not need to be considered

Subterm Criterion

- Simple yet powerful DP processor (for ordinary rewriting due to Hirokawa & Middeldorp, 2004)
- We use the following strict subterm relations on (constrained) terms:
  - for s, t of sort $\mathtt{nat}$ we let $s[\![C]\!] \rhd_{\mathtt{nat}} t[\![C]\!]$ iff $C \Rightarrow s > t$ is $\mathcal{PA}$-valid
  - for s, t not of sort $\mathtt{nat}$ we let $s \rhd_{\mathtt{univ}} t$ iff $s \sim_{\mathcal{E}\cup\mathcal{PA}} \circ \rhd \circ \sim_{\mathcal{E}\cup\mathcal{PA}} t$
- Simple projection π: Assign to each $f^\sharp$ one of its arguments
- $\mathrm{Proc}(\mathcal{P}, \mathcal{R}) = \{(\mathcal{P} - \mathcal{P}', \mathcal{R})\}$, if $\rhd_{\mathtt{univ}}$ is well-founded and $\mathcal{P}' \subseteq \mathcal{P}$ such that
  - for all $s \to t[\![C]\!] \in \mathcal{P} - \mathcal{P}'$, either $\pi(s) \unrhd_{\mathtt{univ}} \pi(t)$ or $\pi(s)[\![C]\!] \unrhd_{\mathtt{nat}} \pi(t)[\![C]\!]$, and
  - for all $s \to t[\![C]\!] \in \mathcal{P}'$, either $\pi(s) \rhd_{\mathtt{univ}} \pi(t)$ or $\pi(s)[\![C]\!] \rhd_{\mathtt{nat}} \pi(t)[\![C]\!]$,
- Note that $\mathcal{R}$, $\mathcal{S}$, and $\mathcal{E}$ do not need to be considered

Subterm Criterion

- Simple yet powerful DP processor (for ordinary rewriting due to Hirokawa & Middeldorp, 2004)
- We use the following strict subterm relations on (constrained) terms:
  - for s, t of sort $\mathtt{nat}$ we let $s[\![C]\!] \rhd_{\mathtt{nat}} t[\![C]\!]$ iff $C \Rightarrow s > t$ is $\mathcal{PA}$-valid
  - for s, t not of sort $\mathtt{nat}$ we let $s \rhd_{\mathtt{univ}} t$ iff $s \sim_{\mathcal{E}\cup\mathcal{PA}} \circ \rhd \circ \sim_{\mathcal{E}\cup\mathcal{PA}} t$
- Simple projection π: Assign to each $f^\sharp$ one of its arguments
- $\mathrm{Proc}(\mathcal{P}, \mathcal{R}) = \{(\mathcal{P} - \mathcal{P}', \mathcal{R})\}$, if $\rhd_{\mathtt{univ}}$ is well-founded and $\mathcal{P}' \subseteq \mathcal{P}$ such that
  - for all $s \to t[\![C]\!] \in \mathcal{P} - \mathcal{P}'$, either $\pi(s) \unrhd_{\mathtt{univ}} \pi(t)$ or $\pi(s)[\![C]\!] \unrhd_{\mathtt{nat}} \pi(t)[\![C]\!]$, and
  - for all $s \to t[\![C]\!] \in \mathcal{P}'$, either $\pi(s) \rhd_{\mathtt{univ}} \pi(t)$ or $\pi(s)[\![C]\!] \rhd_{\mathtt{nat}} \pi(t)[\![C]\!]$,
- Note that $\mathcal{R}$, $\mathcal{S}$, and $\mathcal{E}$ do not need to be considered

Example

- For the DP problem

$$(\{\mathsf{inc}^\sharp(\mathsf{ins}(x, ys)) \to \mathsf{inc}^\sharp(ys)\}, \mathcal{R})$$

we use the simple projection $\pi(\mathsf{inc}^\sharp) = 1$. Then this DP problem is transformed into $(\emptyset, \mathcal{R})$ since

$$\pi(\mathsf{inc}^\sharp(\mathsf{ins}(x, ys))) = \mathsf{ins}(x, ys) \rhd_{\mathtt{univ}} ys = \pi(\mathsf{inc}^\sharp(ys))$$

Example

- For the DP problem

$$(\{\mathsf{inc}^\sharp(\mathsf{ins}(x, ys)) \to \mathsf{inc}^\sharp(ys)\}, \mathcal{R})$$

  we use the simple projection $\pi(\mathsf{inc}^\sharp) = 1$. Then this DP problem is transformed into $(\emptyset, \mathcal{R})$

- For the DP problem

$$\begin{array}{lrcll}(\{ & \mathsf{nats}^\sharp(0, y+1) & \to & \mathsf{nats}^\sharp(1, y+1), & \\ & \mathsf{nats}^\sharp(x+1, y+1) & \to & \mathsf{nats}^\sharp(x, y) & \}, \mathcal{R})\end{array}$$

  we use the simple projection $\pi(\mathsf{nats}^\sharp) = 2$. Then this DP problem is transformed into $(\{\mathsf{nats}^\sharp(0, y+1) \to \mathsf{nats}^\sharp(1, y+1)\}, \mathcal{R})$ since

$$\pi(\mathsf{nats}^\sharp(0, y+1) = y + 1 \trianglerighteq_{\mathtt{nat}} 1 = \pi(\mathsf{nats}^\sharp(1, y+1))$$
$$\pi(\mathsf{nats}^\sharp(x+1, y+1)) = y + 1 \vartriangleright_{\mathtt{nat}} y = \pi(\mathsf{nats}^\sharp(x, y))$$

The resulting DP problem is handled by the dependency graph

$\mathcal{PA}$-polynomial Interpretations

- A $\mathcal{PA}$-polynomial interpretation assigns polynomials to function symbols such that
  - $\mathcal{P}ol(0) = 0$, $\mathcal{P}ol(1) = 1$, and $\mathcal{P}ol(x_1 + x_2) = x_1 + x_2$
  - $\mathcal{P}ol(f) \in \mathbb{N}[x_1, \ldots, x_n]$ if f has n arguments
  - $\mathcal{P}ol(f^\sharp) \in \mathbb{Z}[x_1, \ldots, x_n]$ if $f^\sharp$ has n arguments, where $\mathcal{P}ol(f^\sharp)$ is weakly increasing in all x_i that correspond to arguments that do not have sort `nat`
- Similar to the class of polynomial interpretations used in (Giesl *et. al.*, 2007) for innermost termination analysis of ordinary rewriting
- Gives rise to relations on constrained terms:
  - $s⟦C⟧ \succ_{\mathcal{P}ol} t⟦C⟧$ iff $C \Rightarrow [s]_{\mathcal{P}ol} \geq 0$ and $C \Rightarrow [s]_{\mathcal{P}ol} > [t]_{\mathcal{P}ol}$
  - $s⟦C⟧ \gtrsim_{\mathcal{P}ol} t⟦C⟧$ iff $C \Rightarrow [s]_{\mathcal{P}ol} \geq [t]_{\mathcal{P}ol}$

  In both cases we consider all instantiations over $\mathbb{N}$

$\mathcal{PA}$-polynomial Interpretations

- A $\mathcal{PA}$-polynomial interpretation assigns polynomials to function symbols such that
  - $\mathcal{P}ol(0) = 0$, $\mathcal{P}ol(1) = 1$, and $\mathcal{P}ol(x_1 + x_2) = x_1 + x_2$
  - $\mathcal{P}ol(f) \in \mathbb{N}[x_1, \ldots, x_n]$ if f has n arguments
  - $\mathcal{P}ol(f^\sharp) \in \mathbb{Z}[x_1, \ldots, x_n]$ if $f^\sharp$ has n arguments, where $\mathcal{P}ol(f^\sharp)$ is weakly increasing in all x_i that correspond to arguments that do not have sort `nat`
- Similar to the class of polynomial interpretations used in (Giesl *et. al.*, 2007) for innermost termination analysis of ordinary rewriting
- Gives rise to relations on constrained terms:
  - $s[\![C]\!] \succ_{\mathcal{P}ol} t[\![C]\!]$ iff $C \Rightarrow [s]_{\mathcal{P}ol} \geq 0$ and $C \Rightarrow [s]_{\mathcal{P}ol} > [t]_{\mathcal{P}ol}$
  - $s[\![C]\!] \succsim_{\mathcal{P}ol} t[\![C]\!]$ iff $C \Rightarrow [s]_{\mathcal{P}ol} \geq [t]_{\mathcal{P}ol}$

  In both cases we consider all instantiations over $\mathbb{N}$

$\mathcal{PA}$-polynomial Interpretations

- A $\mathcal{PA}$-polynomial interpretation assigns polynomials to function symbols such that
  - $\mathcal{P}ol(0) = 0$, $\mathcal{P}ol(1) = 1$, and $\mathcal{P}ol(x_1 + x_2) = x_1 + x_2$
  - $\mathcal{P}ol(f) \in \mathbb{N}[x_1, \ldots, x_n]$ if f has n arguments
  - $\mathcal{P}ol(f^\sharp) \in \mathbb{Z}[x_1, \ldots, x_n]$ if $f^\sharp$ has n arguments, where $\mathcal{P}ol(f^\sharp)$ is weakly increasing in all x_i that correspond to arguments that do not have sort `nat`
- Similar to the class of polynomial interpretations used in (Giesl *et. al.*, 2007) for innermost termination analysis of ordinary rewriting
- Gives rise to relations on constrained terms:
  - $s[\![C]\!] \succ_{\mathcal{P}ol} t[\![C]\!]$ iff $C \Rightarrow [s]_{\mathcal{P}ol} \geq 0$ and $C \Rightarrow [s]_{\mathcal{P}ol} > [t]_{\mathcal{P}ol}$
  - $s[\![C]\!] \succsim_{\mathcal{P}ol} t[\![C]\!]$ iff $C \Rightarrow [s]_{\mathcal{P}ol} \geq [t]_{\mathcal{P}ol}$

  In both cases we consider all instantiations over $\mathbb{N}$

DP Processor Based on $\mathcal{PA}$-polynomial Interpretations

- Dependency pairs that are decreasing w.r.t. $\succ_{\mathcal{P}ol}$ cannot be used infinitely often in an infinite chain
- $\text{Proc}(\mathcal{P}, \mathcal{R}) = \{(\mathcal{P} - \mathcal{P}', \mathcal{R})\}$, if $\mathcal{P}' \subseteq \mathcal{P}$ such that
  - $s[\![C]\!] \gtrsim_{\mathcal{P}ol} t[\![C]\!]$ for all $s \to t[\![C]\!] \in \mathcal{P} - \mathcal{P}'$,
  - $s[\![C]\!] \succ_{\mathcal{P}ol} t[\![C]\!]$ for all $s \to t[\![C]\!] \in \mathcal{P}'$,
  - $l[\![C]\!] \gtrsim_{\mathcal{P}ol} r[\![C]\!]$ for all $l \to r[\![C]\!] \in \mathcal{R}$,
  - $l[\![C]\!] \gtrsim_{\mathcal{P}ol} r[\![C]\!]$ for all $l \to r[\![C]\!] \in \mathcal{S}$, and
  - $u \gtrsim_{\mathcal{P}ol} \cap \gtrsim_{\mathcal{P}ol}^{-1} v$ for all $u \approx v \in \mathcal{E}$

DP Processor Based on $\mathcal{PA}$-polynomial Interpretations

- Dependency pairs that are decreasing w.r.t. $\succ_{\mathcal{P}ol}$ cannot be used infinitely often in an infinite chain
- $\text{Proc}(\mathcal{P}, \mathcal{R}) = \{(\mathcal{P} - \mathcal{P}', \mathcal{R})\}$, if $\mathcal{P}' \subseteq \mathcal{P}$ such that
  - $s[\![C]\!] \gtrsim_{\mathcal{P}ol} t[\![C]\!]$ for all $s \to t[\![C]\!] \in \mathcal{P} - \mathcal{P}'$,
  - $s[\![C]\!] \succ_{\mathcal{P}ol} t[\![C]\!]$ for all $s \to t[\![C]\!] \in \mathcal{P}'$,
  - $l[\![C]\!] \gtrsim_{\mathcal{P}ol} r[\![C]\!]$ for all $l \to r[\![C]\!] \in \mathcal{R}$,
  - $l[\![C]\!] \gtrsim_{\mathcal{P}ol} r[\![C]\!]$ for all $l \to r[\![C]\!] \in \mathcal{S}$, and
  - $u \gtrsim_{\mathcal{P}ol} \cap \gtrsim_{\mathcal{P}ol}^{-1} v$ for all $u \approx v \in \mathcal{E}$

Example

- For the first imperative program we obtain the DP problem

$$
\begin{array}{rlll}
(\{ & \mathsf{eval}_0^\sharp(x,y) & \to & \mathsf{eval}_1^\sharp(x+1,1) & [\![x > 0]\!] \\
& \mathsf{eval}_1^\sharp(x,y) & \to & \mathsf{eval}_1^\sharp(x,y+1) & [\![x > y]\!] \\
& \mathsf{eval}_1^\sharp(x+2,y) & \to & \mathsf{eval}_0^\sharp(x,y) & [\![x+2 \not> y]\!]\ \}, \mathcal{R})
\end{array}
$$

- Since terms of sort `nat` cannot be reduced it suffices to consider the dependency pairs
- First, we use the $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_0^\sharp) = x_1 + 1$ and $\mathcal{P}ol(\mathsf{eval}_1^\sharp) = x_1$ to remove the third dependency pair
- Using the dependency graph, the first dependency pair is removed
- Finally, we use the $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_1^\sharp) = x_1 - x_2$ to remove the second dependency pair since $x > y \Rightarrow x - y \geq 0$ and $x > y \Rightarrow x - y > x - (y+1)$ (Recall that we consider instantiations over $\mathbb{N}$ only)

Example

- For the first imperative program we obtain the DP problem

$$(\{ \begin{array}{rcll} \text{eval}_0^\sharp(x,y) & \to & \text{eval}_1^\sharp(x+1,1) & [\![x>0]\!] \\ \text{eval}_1^\sharp(x,y) & \to & \text{eval}_1^\sharp(x,y+1) & [\![x>y]\!] \\ \text{eval}_1^\sharp(x+2,y) & \to & \text{eval}_0^\sharp(x,y) & [\![x+2 \not> y]\!] \ \}, \mathcal{R}) \end{array}$$

- Since terms of sort `nat` cannot be reduced it suffices to consider the dependency pairs
- First, we use the $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\text{eval}_0^\sharp) = x_1 + 1$ and $\mathcal{P}ol(\text{eval}_1^\sharp) = x_1$ to remove the third dependency pair
- Using the dependency graph, the first dependency pair is removed
- Finally, we use the $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\text{eval}_1^\sharp) = x_1 - x_2$ to remove the second dependency pair since $x > y \Rightarrow x - y \geq 0$ and $x > y \Rightarrow x - y > x - (y+1)$ (Recall that we consider instantiations over $\mathbb{N}$ only)

Example

- For the first imperative program we obtain the DP problem

$$\left(\left\{ \begin{array}{rcll} \mathsf{eval}_0^\sharp(x,y) & \to & \mathsf{eval}_1^\sharp(x+1,1) & [\![x>0]\!] \\ \mathsf{eval}_1^\sharp(x,y) & \to & \mathsf{eval}_1^\sharp(x,y+1) & [\![x>y]\!] \\ \mathsf{eval}_1^\sharp(x+2,y) & \to & \mathsf{eval}_0^\sharp(x,y) & [\![x+2 \not> y]\!] \end{array} \right\}, \mathcal{R}\right)$$

- Since terms of sort `nat` cannot be reduced it suffices to consider the dependency pairs
- First, we use the $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_0^\sharp) = x_1 + 1$ and $\mathcal{P}ol(\mathsf{eval}_1^\sharp) = x_1$ to remove the third dependency pair
- Using the dependency graph, the first dependency pair is removed
- Finally, we use the $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_1^\sharp) = x_1 - x_2$ to remove the second dependency pair since $x > y \Rightarrow x - y \geq 0$ and $x > y \Rightarrow x - y > x - (y+1)$ (Recall that we consider instantiations over $\mathbb{N}$ only)

Example

- For the first imperative program we obtain the DP problem

$$(\{\ \mathsf{eval}_0^\sharp(x,y) \rightarrow \mathsf{eval}_1^\sharp(x+1,1)\ [\![x>0]\!] \\ \mathsf{eval}_1^\sharp(x,y) \rightarrow \mathsf{eval}_1^\sharp(x,y+1)\ [\![x>y]\!] \\ \mathsf{eval}_1^\sharp(x+2,y) \rightarrow \mathsf{eval}_0^\sharp(x,y)\ [\![x+2 \not> y]\!]\ \}, \mathcal{R})$$

- Since terms of sort `nat` cannot be reduced it suffices to consider the dependency pairs
- First, we use the $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_0^\sharp) = x_1 + 1$ and $\mathcal{P}ol(\mathsf{eval}_1^\sharp) = x_1$ to remove the third dependency pair
- Using the dependency graph, the first dependency pair is removed
- Finally, we use the $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_1^\sharp) = x_1 - x_2$ to remove the second dependency pair since $x > y \Rightarrow x - y \geq 0$ and $x > y \Rightarrow x - y > x - (y+1)$ (Recall that we consider instantiations over $\mathbb{N}$ only)

Example

- For the first imperative program we obtain the DP problem

$$(\{ \begin{array}{lcll} \mathsf{eval}_0^\sharp(x,y) & \to & \mathsf{eval}_1^\sharp(x+1,1) & [\![x>0]\!] \\ \mathsf{eval}_1^\sharp(x,y) & \to & \mathsf{eval}_1^\sharp(x,y+1) & [\![x>y]\!] \\ \mathsf{eval}_1^\sharp(x+2,y) & \to & \mathsf{eval}_0^\sharp(x,y) & [\![x+2 \not> y]\!] \end{array} \}, \mathcal{R})$$

- Since terms of sort `nat` cannot be reduced it suffices to consider the dependency pairs
- First, we use the $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_0^\sharp) = x_1 + 1$ and $\mathcal{P}ol(\mathsf{eval}_1^\sharp) = x_1$ to remove the third dependency pair
- Using the dependency graph, the first dependency pair is removed
- Finally, we use the $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_1^\sharp) = x_1 - x_2$ to remove the second dependency pair since $x > y \Rightarrow x - y \geq 0$ and $x > y \Rightarrow x - y > x - (y+1)$ (Recall that we consider instantiations over $\mathbb{N}$ only)

Conclusions and Future Work

- We defined an expressive class of rewrite systems with semantic data structures and built-in natural numbers
- Both functional and imperative programs can be translated into this class of rewrite systems
- We extended the dependency pair framework for termination proving to this class of rewrite systems
- We presented powerful DP processors for this framework
- Future work:
  - Conditional rewriting (done)
  - Further DP processors
  - Sufficient conditions for confluence, sufficient completeness, etc.
  - Inductive theorem proving
  - Beyond PA – other theories for specifying constraints

Conclusions and Future Work

- We defined an expressive class of rewrite systems with semantic data structures and built-in natural numbers
- Both functional and imperative programs can be translated into this class of rewrite systems
- We extended the dependency pair framework for termination proving to this class of rewrite systems
- We presented powerful DP processors for this framework
- Future work:
  - Conditional rewriting (done)
  - Further DP processors
  - Sufficient conditions for confluence, sufficient completeness, etc
  - Inductive theorem proving
  - Beyond PA – other theories for specifying constraints

Conclusions and Future Work

- We defined an expressive class of rewrite systems with semantic data structures and built-in natural numbers
- Both functional and imperative programs can be translated into this class of rewrite systems
- We extended the dependency pair framework for termination proving to this class of rewrite systems
- We presented powerful DP processors for this framework
- Future work:
  - Conditional rewriting (done)
  - Further DP processors
  - Sufficient conditions for confluence, sufficient completeness, etc.
  - Inductive theorem proving
  - Beyond PA – other theories for specifying constraints

Conclusions and Future Work

- We defined an expressive class of rewrite systems with semantic data structures and built-in natural numbers
- Both functional and imperative programs can be translated into this class of rewrite systems
- We extended the dependency pair framework for termination proving to this class of rewrite systems
- We presented powerful DP processors for this framework
- Future work:
  - Conditional rewriting (done)
  - Further DP processors
  - Sufficient conditions for confluence, sufficient completeness, etc.
  - Inductive theorem proving
  - Beyond PA – other theories for specifying constraints

Conclusions and Future Work

- We defined an expressive class of rewrite systems with semantic data structures and built-in natural numbers
- Both functional and imperative programs can be translated into this class of rewrite systems
- We extended the dependency pair framework for termination proving to this class of rewrite systems
- We presented powerful DP processors for this framework
- Future work:
  - Conditional rewriting (done)
  - Further DP processors
  - Sufficient conditions for confluence, sufficient completeness, etc.
  - Inductive theorem proving
  - Beyond PA – other theories for specifying constraints

Conclusions and Future Work

- We defined an expressive class of rewrite systems with semantic data structures and built-in natural numbers
- Both functional and imperative programs can be translated into this class of rewrite systems
- We extended the dependency pair framework for termination proving to this class of rewrite systems
- We presented powerful DP processors for this framework
- Future work:
    - Conditional rewriting (done)
    - Further DP processors
    - Sufficient conditions for confluence, sufficient completeness, etc.
    - Inductive theorem proving
    - Beyond PA – other theories for specifying constraints

Conclusions and Future Work

- We defined an expressive class of rewrite systems with semantic data structures and built-in natural numbers
- Both functional and imperative programs can be translated into this class of rewrite systems
- We extended the dependency pair framework for termination proving to this class of rewrite systems
- We presented powerful DP processors for this framework
- Future work:
  - Conditional rewriting (done)
  - Further DP processors
  - Sufficient conditions for confluence, sufficient completeness, etc.
  - Inductive theorem proving
  - Beyond PA – other theories for specifying constraints

Conclusions and Future Work

- We defined an expressive class of rewrite systems with semantic data structures and built-in natural numbers
- Both functional and imperative programs can be translated into this class of rewrite systems
- We extended the dependency pair framework for termination proving to this class of rewrite systems
- We presented powerful DP processors for this framework
- Future work:
  - Conditional rewriting (done)
  - Further DP processors
  - Sufficient conditions for confluence, sufficient completeness, etc.
  - Inductive theorem proving
  - Beyond PA – other theories for specifying constraints

Conclusions and Future Work

- We defined an expressive class of rewrite systems with semantic data structures and built-in natural numbers
- Both functional and imperative programs can be translated into this class of rewrite systems
- We extended the dependency pair framework for termination proving to this class of rewrite systems
- We presented powerful DP processors for this framework
- Future work:
  - Conditional rewriting (done)
  - Further DP processors
  - Sufficient conditions for confluence, sufficient completeness, etc.
  - Inductive theorem proving
  - Beyond PA – other theories for specifying constraints