# Termination of Term Rewriting and Applications

Jürgen Giesl

LuFG Informatik 2
RWTH Aachen
Germany

# Termination Analysis for TRSs

$$\begin{array}{lrcl} \mathcal{R}: & \mathsf{plus}(x,0) & \to & x \\ & \mathsf{plus}(x,\mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{plus}(x,y)) \end{array}$$

$\mathcal{R}$ is *terminating* iff there is no infinite evaluation $t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} \ldots$

- easier / more general than for functional programs
- termination technique for TRSs can be adapted to strategies, types, higher-order functions, ...
- suitable for automation
- **But:** halting problem is undecidable!
  $\Rightarrow$ automated termination proofs do not always succeed

# Termination Analysis for TRSs

$$\begin{array}{lrcl} \mathcal{R}: & \mathsf{plus}(x,0) & \to & x \\ & \mathsf{plus}(x,\mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{plus}(x,y)) \end{array}$$

$\mathcal{R}$ is *terminating* iff there is no infinite evaluation $t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} \ldots$

**Computation of "2 + 1":**
$$\begin{array}{lll} \mathsf{plus}(\mathsf{s}(\mathsf{s}(0)),\mathsf{s}(0)) & \to_{\mathcal{R}} & \mathsf{s}(\mathsf{plus}(\mathsf{s}(\mathsf{s}(0)),0)) \\ & \to_{\mathcal{R}} & \mathsf{s}(\mathsf{s}(\mathsf{s}(0))) \end{array}$$

# Outline

**1. Classical Techniques**
- Lexicographic Path Order (with Status) *(Kamin & Lévy, 80)*
- Recursive Path Order (with Status) *(Dershowitz, 82)*
- Polynomial Order *(Lankford, 79)*

**2. Dependency Pairs** *(Arts & Giesl et al, 96 – today)*
- Proving Termination
- Proving Innermost Termination
- Disproving Termination
- Integrating Other Termination Techniques

**3. Other Recent Techniques**
- Semantic Labelling *(Zantema, 95)*
- Match-Bounds *(Geser, Hofbauer, Waldmann, Zantema, 03 – today)*

**4. Applications**
- Termination of Logic Programs *(Schneider-Kamp, Giesl, Serebrenik, Thiemann, 06)*
- Termination of Functional Programs *(Giesl, Swiderski, Schneider-K., Thiemann, 06)*

# 1. Classical Techniques

$$\mathcal{R}: \quad \begin{array}{rcl} \mathsf{plus}(x,0) & \to & x \\ \mathsf{plus}(x,\mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{plus}(x,y)) \end{array}$$

$\mathcal{R}$ is *terminating* iff there is no infinite evaluation $t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} \ldots$

**Goal:** Find order $\succ$ such that $l \succ r$ for all rules $l \to r \in \mathcal{R}$

- $\succ$ is well-founded: no infinite sequence $t_1 \succ t_2 \succ \ldots$
- $\succ$ is monotonic: if $t_1 \succ t_2$ then $f(\ldots t_1 \ldots) \succ f(\ldots t_2 \ldots)$
  if $\mathsf{plus}(x,0) \succ x$ then $\mathsf{s}(\mathsf{plus}(x,0)) \succ \mathsf{s}(x)$
- $\succ$ is stable: if $t_1 \succ t_2$ then $\sigma(t_1) \succ \sigma(t_2)$
  if $\mathsf{plus}(x,0) \succ x$ then $\mathsf{plus}(0,0) \succ 0$

# Lexicographic Path Order (LPO)

$$\mathcal{R}: \quad \begin{array}{rcl} \mathsf{plus}(x,0) & \to & x \\ \mathsf{plus}(x,\mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{plus}(x,y)) \end{array}$$

LPO is an order where $s \succ t$ iff

- $s = f(s_1 \ldots s_n)$, $s_i \succeq t$ for some $i$
  $\mathsf{plus}(x,0) \succ x$ since $x \succeq x$
- $s = f(\ldots)$, $t = g(t_1 \ldots t_n)$, $f \succ g$, and $s \succ t_i$ for all $i$ (e.g. $\mathsf{plus} \succ \mathsf{s}$)
  $\mathsf{plus}(x,\mathsf{s}(y)) \succ \mathsf{s}(\mathsf{plus}(x,y))$ if $\mathsf{plus}(x,\mathsf{s}(y)) \succ \mathsf{plus}(x,y)$
- $s = f(s_1 \ldots s_{j-1} s_j \ldots s_n)$, $t = f(s_1 \ldots s_{j-1} t_j \ldots t_n)$, $s_j \succ t_j$, $s \succ t_i$ for $i > j$
  $\mathsf{plus}(x,\mathsf{s}(y)) \succ \mathsf{plus}(x,y)$ since $\mathsf{s}(y) \succ y$
  $\mathsf{plus}(\mathsf{s}(x),y) \succ \mathsf{plus}(x,\mathsf{s}(y))$ since $\mathsf{s}(x) \succ x$, $\mathsf{plus}(\mathsf{s}(x),y) \succ \mathsf{s}(y)$

# LPO with Status (LPOS)

$$\mathcal{R}: \quad \begin{array}{rcl} \mathsf{plus}(x,0) & \to & x \\ \mathsf{plus}(x,\mathsf{s}(y)) & \to & \mathsf{plus}(\mathsf{s}(x),y) \end{array}$$

LPO does lexicographic comparison from left to right:

- $s = f(s_1 \ldots s_{j-1} s_j \ldots s_n)$, $t = f(s_1 \ldots s_{j-1} t_j \ldots t_n)$, $s_j \succ t_j$, $s \succ t_i$ for $i > j$
  $\mathsf{plus}(x,\mathsf{s}(y)) \not\succ \mathsf{plus}(\mathsf{s}(x),y)$

We need lexicographic comparison from right to left:

- $s = f(s_1 \ldots s_j s_{j+1} \ldots s_n)$, $t = f(t_1 \ldots t_j s_{j+1} \ldots s_n)$, $s_j \succ t_j$, $s \succ t_i$ for $i < j$
  $\mathsf{plus}(x,\mathsf{s}(y)) \succ \mathsf{plus}(\mathsf{s}(x),y)$ since $\mathsf{s}(y) \succ y$, $\mathsf{plus}(x,\mathsf{s}(y)) \succ \mathsf{s}(x)$

LPO with Status: status($f$) = permutation of 1, ..., arity($f$)
determines order in lexicographic comparison

# Recursive (Multiset) Path Order (RPO)

$$\mathcal{R}: \quad \begin{array}{rcl} \mathsf{plus}(x,0) & \to & x \\ \mathsf{plus}(x,\mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{plus}(y,x)) \end{array}$$

RPO is an order where $s \succ t$ iff

- $s = f(s_1 \ldots s_n)$, $s_i \succeq t$ for some $i$
  $\mathsf{plus}(x,0) \succ x$ since $x \succeq x$
- $s = f(\ldots)$, $t = g(t_1 \ldots t_n)$, $f \succ g$, and $s \succ t_i$ for all $i$ (e.g. $\mathsf{plus} \succ \mathsf{s}$)
  $\mathsf{plus}(x,\mathsf{s}(y)) \succ \mathsf{s}(\mathsf{plus}(y,x))$ if $\mathsf{plus}(x,\mathsf{s}(y)) \succ \mathsf{plus}(y,x)$
- $s = f(s_1 \ldots s_n)$, $t = f(t_1 \ldots t_n)$, $\{s_1, \ldots, s_n\} \succ_{mul} \{t_1, \ldots, t_n\}$
  $\mathsf{plus}(x,\mathsf{s}(y)) \succ \mathsf{plus}(y,x)$ since $\{x, \mathsf{s}(y)\} \succ_{mul} \{y, x\}$

# RPO with Status (RPOS)

$$\begin{aligned} \mathsf{plus}_1(0, y) &\to y \\ \mathsf{plus}_1(\mathsf{s}(x), y) &\to \mathsf{plus}_1(x, \mathsf{s}(y)) \end{aligned}$$

$$\begin{aligned} \mathsf{plus}_2(x, 0) &\to x \\ \mathsf{plus}_2(x, \mathsf{s}(y)) &\to \mathsf{plus}_2(\mathsf{s}(x), y) \end{aligned}$$

$$\begin{aligned} \mathsf{plus}_3(x, 0) &\to x \\ \mathsf{plus}_3(x, \mathsf{s}(y)) &\to \mathsf{s}(\mathsf{plus}_3(y, x)) \end{aligned}$$

RPO with Status: status($f$) = permutation of 1, ..., arity($f$) or "*mul*"

status($\mathsf{plus}_1$) = (1,2)
status($\mathsf{plus}_2$) = (2,1)
status($\mathsf{plus}_3$) = *mul*

# Polynomial Order

$$\begin{aligned} \mathcal{P}ol(\,\mathsf{plus}(0, y)\,) &> \mathcal{P}ol(\,y\,) \\ \mathcal{P}ol(\,\mathsf{plus}(\mathsf{s}(x), y)\,) &> \mathcal{P}ol(\,\mathsf{s}(\mathsf{plus}(x, y))\,) \end{aligned}$$

$\mathcal{R}$ is *terminating* iff there is no infinite evaluation $t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} \ldots$

**Goal:** Find order $\succ$ such that $l \succ r$ for all rules $l \to r \in \mathcal{R}$

**Polynomial Order:** $l \succ r$ iff $\mathcal{P}ol(l) > \mathcal{P}ol(r)$

$$\begin{aligned} \mathcal{P}ol(0) &= 1 \\ \mathcal{P}ol(\mathsf{s}(t)) &= 1 + \mathcal{P}ol(t) \\ \mathcal{P}ol(\mathsf{plus}(t_1, t_2)) &= 2\,\mathcal{P}ol(t_1) + \mathcal{P}ol(t_2) \end{aligned}$$

# Polynomial Order

$$\mathcal{R}: \quad \begin{aligned} \mathsf{plus}(0, y) &\to y \\ \mathsf{plus}(\mathsf{s}(x), y) &\to \mathsf{s}(\mathsf{plus}(x, y)) \end{aligned}$$

$\mathcal{R}$ is *terminating* iff there is no infinite evaluation $t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} \ldots$

**Goal:** Find order $\succ$ such that $l \succ r$ for all rules $l \to r \in \mathcal{R}$

**Polynomial Order:** $l \succ r$ iff $\mathcal{P}ol(l) > \mathcal{P}ol(r)$

# Polynomial Order

$$\begin{aligned} 2 + \mathcal{P}ol(y) &> \mathcal{P}ol(y) \\ 2\,(1 + \mathcal{P}ol(x)) + \mathcal{P}ol(y) &> 1 + 2\,\mathcal{P}ol(x) + \mathcal{P}ol(y) \end{aligned}$$

$\mathcal{R}$ is *terminating* iff there is no infinite evaluation $t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} \ldots$

**Goal:** Find order $\succ$ such that $l \succ r$ for all rules $l \to r \in \mathcal{R}$

**Polynomial Order:** $l \succ r$ iff $\mathcal{P}ol(l) > \mathcal{P}ol(r)$

$$\begin{aligned} \mathcal{P}ol(0) &= 1 \\ \mathcal{P}ol(\mathsf{s}(t)) &= 1 + \mathcal{P}ol(t) \\ \mathcal{P}ol(\mathsf{plus}(t_1, t_2)) &= 2\,\mathcal{P}ol(t_1) + \mathcal{P}ol(t_2) \end{aligned}$$

Recent Improvement: **Matrix Order** (map to *vectors*, not *numbers*)

# 1. Classical Techniques

- **Goal:** Find order $\succ$ such that $l \succ r$ for all rules $l \to r \in \mathcal{R}$
  - Lexicographic Path Order (Kamin & Levy 1980)
  - Recursive Path Order (Dershowitz 1982)
  - Recursive Path Order with Status
  - Polynomial Orders (Lankford 1979)
- all these orders are *simplification orders*: $f(\ldots t \ldots) \succ t$
- too restrictive for many important TRSs
- Dependency Pairs
  - Original method: Arts & Giesl (1996 - 2002)
  - New refinements: Giesl, Thiemann, Schneider-Kamp (since 2003)
    Middeldorp, Hirokawa (since 2001)

# Dependency Pairs for Termination

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

- Standard Approach:
  Compare left- and right-hand sides of rules
- Problem:
  Automated techniques use *simplification orders*
  $\Rightarrow$ Failure!
- *Dependency Pair Approach:*
  Examine only those subterms which are responsible
  for starting new reductions

# Dependency Pairs for Termination

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

- Standard Approach:
  Compare left- and right-hand sides of rules
- Problem:
  Automated techniques use *simplification orders*
  $\mathsf{div}(\mathsf{s}(x),\mathsf{s}(\mathsf{s}(x))) \succ \mathsf{s}(\mathsf{div}(\mathsf{minus}(x,\mathsf{s}(x)),\mathsf{s}(\mathsf{s}(x))))$

# Dependency Pairs for Termination

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

- Defined Symbols: minus, div
  Constructors: 0, s
- **Definition**
  If $f(s_1,\ldots,s_n) \to C[g(t_1,\ldots,t_m)]$ is a rule and $g$ is defined, then
  $\mathsf{F}(s_1,\ldots,s_n) \to \mathsf{G}(t_1,\ldots,t_m)$ is a *dependency pair*

$$\begin{aligned}
\mathsf{M}(\mathsf{s}(x),\mathsf{s}(y)) &\to \mathsf{M}(x,y)\\
\mathsf{D}(\mathsf{s}(x),\mathsf{s}(y)) &\to \mathsf{M}(x,y)\\
\mathsf{D}(\mathsf{s}(x),\mathsf{s}(y)) &\to \mathsf{D}(\mathsf{minus}(x,y),\mathsf{s}(y))
\end{aligned}$$

$\mathsf{M}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{M}(x,y)$
$\mathsf{D}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{M}(x,y)$
$\mathsf{D}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{D}(\mathsf{minus}(x,y),\mathsf{s}(y))$

$\mathsf{minus}(x,0) \to x$
$\mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{minus}(x,y)$
$\mathsf{div}(0,\mathsf{s}(y)) \to 0$
$\mathsf{div}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{s}(\mathsf{div}(\mathsf{minus}(x,y),\mathsf{s}(y)))$

- **Definition**
  A sequence of dependency pairs $s_1 \to t_1,\ s_2 \to t_2,\ s_3 \to t_3, \ldots$ is a *chain* iff there exists a substitution $\sigma$ such that

  $t_1\sigma \to^* s_2\sigma, \quad t_2\sigma \to^* s_3\sigma, \quad \ldots$

$\mathsf{D}(\mathsf{s}(x_1),\mathsf{s}(y_1)) \to \mathsf{D}(\mathsf{minus}(x_1,y_1),\mathsf{s}(y_1))$, $\mathsf{D}(\mathsf{s}(x_2),\mathsf{s}(y_2)) \to \mathsf{D}(\mathsf{minus}(x_2,y_2),\mathsf{s}(y_2))$
$\mathsf{D}(\mathsf{s}(\mathsf{s}(0)),\mathsf{s}(0)) \to \mathsf{D}(\mathsf{minus}(\mathsf{s}(0),0),\mathsf{s}(0))$ $\mathsf{D}(\mathsf{s}(0),\mathsf{s}(0)) \to \mathsf{D}(\mathsf{minus}(0,0),\mathsf{s}(0))$

with $\sigma = [x_1/\mathsf{s}(0),\ y_1/0,\ x_2/0,\ y_2/0]$

- **Theorem**
  A TRS terminates iff there is no infinite chain.

# Dependency Pair Framework

- **Basic Idea**
  - examine *DP problems* $(\mathcal{P},\mathcal{R})$
  - a DP problem $(\mathcal{P},\mathcal{R})$ is *finite* iff there is no infinite $(\mathcal{P},\mathcal{R})$-chain
- **Definition**
  A sequence of dependency pairs $s_1 \to t_1,\ s_2 \to t_2,\ s_3 \to t_3, \ldots$ is a *chain* iff there exists a substitution $\sigma$ such that

  $t_1\sigma \to^* s_2\sigma, \quad t_2\sigma \to^* s_3\sigma, \quad \ldots$
- **Theorem**
  A TRS terminates iff there is no infinite chain.

# Dependency Pair Framework

- Apply the general idea of problem solving for termination analysis
  - transform problems into simpler sub-problems repeatedly until all problems are solved
- What *objects* do we work on, i.e., what are the "*problems*"?
  - ~~TRSs $\mathcal{R}$~~ not powerful enough
  - ~~DPs $\mathcal{P}$~~ not expressive enough
  - DP problems $(\mathcal{P},\mathcal{R})$
- What *techniques* do we use for transformation?
  - DP processors: $Proc(\,(\mathcal{P},\mathcal{R})\,) = \{(\mathcal{P}_1,\mathcal{R}_1),\ldots,(\mathcal{P}_n,\mathcal{R}_n)\}$
- When is a problem *solved*?
  - $(\mathcal{P},\mathcal{R})$ is *finite* iff there is no infinite $(\mathcal{P},\mathcal{R})$-chain

# Dependency Pair Framework

- **Basic Idea**
  - examine *DP problems* $(\mathcal{P},\mathcal{R})$
  - a DP problem $(\mathcal{P},\mathcal{R})$ is *finite* iff there is no infinite $(\mathcal{P},\mathcal{R})$-chain
- **Definition**
  A sequence of pairs $s_1 \to t_1,\ s_2 \to t_2,\ s_3 \to t_3, \ldots$ from $\mathcal{P}$ is a $(\mathcal{P},\mathcal{R})$-*chain* iff there exists a substitution $\sigma$ such that

  $t_1\sigma \to^*_{\mathcal{R}} s_2\sigma, \quad t_2\sigma \to^*_{\mathcal{R}} s_3\sigma, \quad \ldots$
- **Theorem**
  A TRS terminates iff $(DP(\mathcal{R}),\mathcal{R})$ is *finite*.

# Dependency Pair Framework

- **Procedure**
  1. Start with the *initial* DP problem $(DP(\mathcal{R}), \mathcal{R})$.
  2. Transform a remaining DP problem by a sound processor.
  3. If result is **"no"** and all processors were complete, return **"no"**.
     If there is no remaining DP problem, then return **"yes"**.
     Otherwise go to 2.

  *DP processor:* $Proc(\,(\mathcal{P},\mathcal{R})\,) = \{(\mathcal{P}_1,\mathcal{R}_1),\ldots,(\mathcal{P}_n,\mathcal{R}_n)\}$ or **"no"**

  $Proc$ is *sound:* if all $(\mathcal{P}_i,\mathcal{R}_i)$ are finite, then $(\mathcal{P},\mathcal{R})$ is finite
  $Proc$ is *complete:* if some $(\mathcal{P}_i,\mathcal{R}_i)$ is infinite or $Proc((\mathcal{P},\mathcal{R}))$ = **"no"**, then $(\mathcal{P},\mathcal{R})$ is infinite

- **Theorem**
  A TRS terminates iff $(DP(\mathcal{R}),\mathcal{R})$ is finite.

# I. DP Processors for Proving Termination

- **Dependency Graph Processor**
- **Reduction Pair Processor**

  Processors only modify $\mathcal{P}$: $Proc(\,(\mathcal{P},\mathcal{R})\,) = \{(\mathcal{P}_1,\mathcal{R}),\ldots,(\mathcal{P}_n,\mathcal{R})\}$

- **Rule Removal Processor**

  Processor modifies $\mathcal{P}$ and $\mathcal{R}$: $Proc(\,(\mathcal{P},\mathcal{R})\,) = \{(\mathcal{P}_1,\mathcal{R}_1)\}$

# Dependency Pair Framework

- **Procedure**
  1. Start with the *initial* DP problem $(DP(\mathcal{R}), \mathcal{R})$.
  2. Transform a remaining DP problem by a sound processor.
  3. If result is **"no"** and all processors were complete, return **"no"**.
     If there is no remaining DP problem, then return **"yes"**.
     Otherwise go to 2.

- **Remaining Lecture on Dependency Pairs**
  - I. DP Processors for Proving Termination
  - II. DP Processors for Proving Innermost Termination
  - III. DP Processors for Disproving Termination
  - IV. DP Processors from Other Termination Techniques

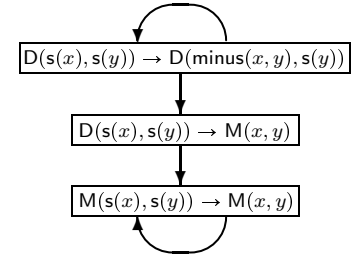
$\mathcal{P}$: $\mathsf{M}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{M}(x,y)$
$\mathsf{D}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{M}(x,y)$
$\mathsf{D}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{D}(\mathsf{minus}(x,y),\mathsf{s}(y))$

$\mathcal{R}$: $\mathsf{minus}(x,0) \to x$
$\mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{minus}(x,y)$
$\mathsf{div}(0,\mathsf{s}(y)) \to 0$
$\mathsf{div}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{s}(\mathsf{div}(\mathsf{minus}(x,y),\mathsf{s}(y)))$

- **Dependency Graph Processor** (sound & complete)

  $Proc(\,(\mathcal{P},\mathcal{R})\,) = \{\,(\mathcal{P}_1,\mathcal{R}),\ldots,(\mathcal{P}_n,\mathcal{R})\,\}$

  where $\mathcal{P}_1,\ldots,\mathcal{P}_n$ are the SCCs of the $(\mathcal{P},\mathcal{R})$-dependency graph




- $(\mathcal{P},\mathcal{R})$-Dependency Graph
  - directed graph whose nodes are the pairs of $\mathcal{P}$
  - arc from $s \to t$ to $v \to w$ iff $s \to t$, $v \to w$ is a $(\mathcal{P},\mathcal{R})$-chain

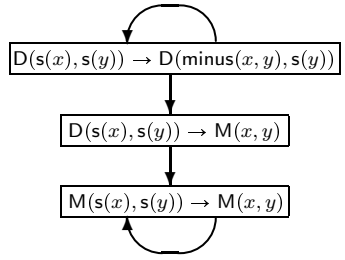
$\mathcal{P}$: $\mathsf{M}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{M}(x,y)$
$\mathsf{D}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{M}(x,y)$
$\mathsf{D}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{D}(\mathsf{minus}(x,y),\mathsf{s}(y))$

$\mathcal{R}$: $\mathsf{minus}(x,0) \to x$
$\mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{minus}(x,y)$
$\mathsf{div}(0,\mathsf{s}(y)) \to 0$
$\mathsf{div}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{s}(\mathsf{div}(\mathsf{minus}(x,y),\mathsf{s}(y)))$

- **Dependency Graph Processor** (sound & complete)

$Proc(\,(\mathcal{P},\mathcal{R})\,) = \{\,(\mathcal{P}_1,\mathcal{R}),\ldots,(\mathcal{P}_n,\mathcal{R})\,\}$

where $\mathcal{P}_1,\ldots,\mathcal{P}_n$ are the SCCs of the $(\mathcal{P},\mathcal{R})$-dependency graph




$\mathcal{P}_1$: $\mathsf{M}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{M}(x,y)$ $\quad\mathcal{R}$: ...

$\mathcal{P}_2$: $\mathsf{D}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{D}(\mathsf{minus}(x,y),\mathsf{s}(y))$ $\quad\mathcal{R}$: ...

---

$\mathcal{P}_1$: $\mathsf{M}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{M}(x,y)$

$\mathcal{P}_2$: $\mathsf{D}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{D}(\mathsf{minus}(x,y),\mathsf{s}(y))$

$\mathcal{R}$: $\mathsf{minus}(x,0) \to x$
$\mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{minus}(x,y)$
$\mathsf{div}(0,\mathsf{s}(y)) \to 0$
$\mathsf{div}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{s}(\mathsf{div}(\mathsf{minus}(x,y),\mathsf{s}(y)))$

- **Reduction Pair Processor** (sound & complete)

$Proc(\,(\mathcal{P},\mathcal{R})\,) = \{\,(\mathcal{P}\backslash\mathcal{P}_{\succ},\,\mathcal{R})\,\}$ if

- $l \gtrsim r$ for all rules $l \to r$ in $\mathcal{R}$ $\qquad \mathcal{R}_{\gtrsim} = \mathcal{R}$
- $s \succ t$ or $s \gtrsim t$ for all $s \to t$ in $\mathcal{P}$ $\qquad \mathcal{P}_{\succ} \cup \mathcal{P}_{\gtrsim} = \mathcal{P}$

- Resulting Inequalities for $(\mathcal{P}_1,\mathcal{R})$:

$\mathcal{P}_1$: $\mathsf{M}(\mathsf{s}(x),\mathsf{s}(y)) \;(\gtrsim)\; \mathsf{M}(x,y)$

$\mathcal{P}ol(\mathsf{s}(t)) = 1 + \mathcal{P}ol(t)$
$\mathcal{P}ol(f(t_1,t_2)) = \mathcal{P}ol(t_1)$

$\mathcal{R}$: $\mathsf{minus}(x,0) \gtrsim x$
$\mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) \gtrsim \mathsf{minus}(x,y)$
$\mathsf{div}(0,\mathsf{s}(y)) \gtrsim 0$
$\mathsf{div}(\mathsf{s}(x),\mathsf{s}(y)) \gtrsim \mathsf{s}(\mathsf{div}(\mathsf{minus}(x,y),\mathsf{s}(y)))$

---

$\mathcal{P}_1$: $\mathsf{M}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{M}(x,y)$

$\mathcal{P}_2$: $\mathsf{D}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{D}(\mathsf{minus}(x,y),\mathsf{s}(y))$

$\mathcal{R}$: $\mathsf{minus}(x,0) \to x$
$\mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{minus}(x,y)$
$\mathsf{div}(0,\mathsf{s}(y)) \to 0$
$\mathsf{div}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{s}(\mathsf{div}(\mathsf{minus}(x,y),\mathsf{s}(y)))$

- $(\gtrsim,\succ)$ is a *reduction pair* iff
  - $\succ$ is stable and well founded
  - $\gtrsim$ is stable and monotonic
  - $\succ$ and $\gtrsim$ are compatible ($\succ \circ \gtrsim \subseteq \succ$ or $\gtrsim \circ \succ \subseteq \succ$)

$s_1 \to t_1$, $\quad s_2 \to t_2$, $\quad s_3 \to t_3$, ...
$s_1\sigma \;(\gtrsim)\; t_1\sigma \gtrsim s_2\sigma \;(\gtrsim)\; t_2\sigma \gtrsim s_3\sigma \;(\gtrsim)\; t_3\sigma \gtrsim \ldots$

---

$\mathcal{P}_1$: $\mathsf{M}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{M}(x,y)$

$\mathcal{P}_2$: $\mathsf{D}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{D}(\mathsf{minus}(x,y),\mathsf{s}(y))$

$\mathcal{R}$: $\mathsf{minus}(x,0) \to x$
$\mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{minus}(x,y)$
$\mathsf{div}(0,\mathsf{s}(y)) \to 0$
$\mathsf{div}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{s}(\mathsf{div}(\mathsf{minus}(x,y),\mathsf{s}(y)))$

- **Reduction Pair Processor** (sound & complete)

$Proc(\,(\mathcal{P},\mathcal{R})\,) = \{\,(\mathcal{P}\backslash\mathcal{P}_{\succ},\,\mathcal{R})\,\}$ if

- $l \gtrsim r$ for all rules $l \to r$ in $\mathcal{R}$ $\qquad \mathcal{R}_{\gtrsim} = \mathcal{R}$
- $s \succ t$ or $s \gtrsim t$ for all $s \to t$ in $\mathcal{P}$ $\qquad \mathcal{P}_{\succ} \cup \mathcal{P}_{\gtrsim} = \mathcal{P}$

- Resulting Inequalities for $(\mathcal{P}_1,\mathcal{R})$:

$\mathcal{P}_1$: $1 + x > x$

$\mathcal{P}ol(\mathsf{s}(t)) = 1 + \mathcal{P}ol(t)$
$\mathcal{P}ol(f(t_1,t_2)) = \mathcal{P}ol(t_1)$

$\mathcal{R}$: $x \geqslant x$
$1 + x \geqslant x$
$0 \geqslant 0$
$1 + x \geqslant 1 + x$

$\mathcal{P}_1$: $\mathsf{M}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{M}(x,y)$

$\mathcal{P}_2$: $\mathsf{D}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{D}(\mathsf{minus}(x,y),\mathsf{s}(y))$

$\mathcal{R}$:
$$\begin{aligned}\mathsf{minus}(x,0) &\to x\\ \mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) &\to \mathsf{minus}(x,y)\\ \mathsf{div}(0,\mathsf{s}(y)) &\to 0\\ \mathsf{div}(\mathsf{s}(x),\mathsf{s}(y)) &\to \mathsf{s}(\mathsf{div}(\mathsf{minus}(x,y),\mathsf{s}(y)))\end{aligned}$$

- **Reduction Pair Processor** (sound & complete)

  $Proc(\,(\mathcal{P},\mathcal{R})\,) = \{\,(\mathcal{P}\backslash\mathcal{P}_{\succ},\,\mathcal{R})\,\}$ if

  - $l \gtrsim r$ for all rules $l \to r$ in $\mathcal{R}$ $\qquad \mathcal{R}_{\gtrsim} = \mathcal{R}$
  - $s \succ t$ or $s \gtrsim t$ for all $s \to t$ in $\mathcal{P}$ $\qquad \mathcal{P}_{\succ} \cup \mathcal{P}_{\gtrsim} = \mathcal{P}$

- Resulting Inequalities for $(\mathcal{P}_2,\mathcal{R})$:

$\mathcal{P}_2$: $\mathsf{D}(\mathsf{s}(x),\mathsf{s}(y)) \;(\gtrsim)\; \mathsf{D}(\mathsf{minus}(x,y),\mathsf{s}(y))$

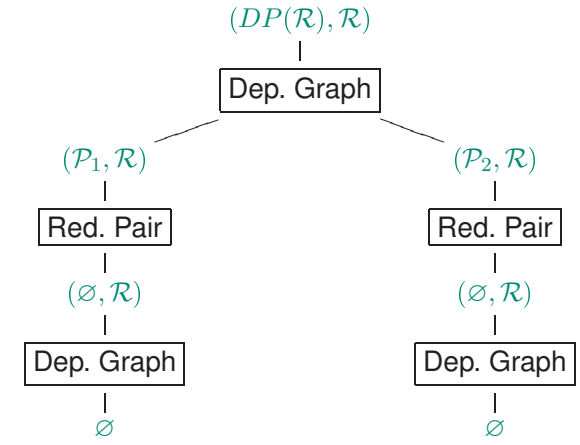
$$\begin{aligned}\mathsf{minus}(0,\mathsf{s}(y)) &\gtrsim 0\\ \mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) &\gtrsim \mathsf{minus}(x,y)\\ \mathsf{div}(0,\mathsf{s}(y)) &\gtrsim 0\\ \mathsf{div}(\mathsf{s}(x),\mathsf{s}(y)) &\gtrsim \mathsf{s}(\mathsf{div}(\mathsf{minus}(x,y),\mathsf{s}(y)))\end{aligned}$$

$\mathcal{P}_1$: $\mathsf{M}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{M}(x,y)$

$\mathcal{P}_2$: $\mathsf{D}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{D}(\mathsf{minus}(x,y),\mathsf{s}(y))$

$\mathcal{R}$:
$$\begin{aligned}\mathsf{minus}(x,0) &\to x\\ \mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) &\to \mathsf{minus}(x,y)\\ \mathsf{div}(0,\mathsf{s}(y)) &\to 0\\ \mathsf{div}(\mathsf{s}(x),\mathsf{s}(y)) &\to \mathsf{s}(\mathsf{div}(\mathsf{minus}(x,y),\mathsf{s}(y)))\end{aligned}$$

Termination is proved automatically!




$\mathcal{P}_1$: $\mathsf{M}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{M}(x,y)$

$\mathcal{P}_2$: $\mathsf{D}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{D}(\mathsf{minus}(x,y),\mathsf{s}(y))$

$\mathcal{R}$:
$$\begin{aligned}\mathsf{minus}(x,0) &\to x\\ \mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) &\to \mathsf{minus}(x,y)\\ \mathsf{div}(0,\mathsf{s}(y)) &\to 0\\ \mathsf{div}(\mathsf{s}(x),\mathsf{s}(y)) &\to \mathsf{s}(\mathsf{div}(\mathsf{minus}(x,y),\mathsf{s}(y)))\end{aligned}$$

- **Reduction Pair Processor** (sound & complete)

  $Proc(\,(\mathcal{P},\mathcal{R})\,) = \{\,(\mathcal{P}\backslash\mathcal{P}_{\succ},\,\mathcal{R})\,\}$ if

  - $l \gtrsim r$ for all rules $l \to r$ in $\mathcal{R}$ $\qquad \mathcal{R}_{\gtrsim} = \mathcal{R}$
  - $s \succ t$ or $s \gtrsim t$ for all $s \to t$ in $\mathcal{P}$ $\qquad \mathcal{P}_{\succ} \cup \mathcal{P}_{\gtrsim} = \mathcal{P}$

- Resulting Inequalities for $(\mathcal{P}_2,\mathcal{R})$:

$\mathcal{P}_2$: $\quad x+1 \;>\; x$

$$\begin{aligned}\mathcal{P}ol(\mathsf{s}(t)) &= \mathcal{P}ol(t)+1\\ \mathcal{P}ol(f(t_1,t_2)) &= \mathcal{P}ol(t_1)\end{aligned}$$

$\mathcal{R}$:
$$\begin{aligned}x &\geqslant x\\ x+1 &\geqslant x\\ 0 &\geqslant 0\\ x+1 &\geqslant x+1\end{aligned}$$

$\mathcal{P}$: $\mathsf{M}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{M}(\mathsf{p}(\mathsf{s}(x)),\mathsf{p}(\mathsf{s}(y)))$

$\mathcal{R}$:
$$\begin{aligned}\mathsf{p}(\mathsf{s}(x)) &\to x\\ \mathsf{minus}(x,0) &\to x\\ \mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) &\to \mathsf{minus}(\mathsf{p}(\mathsf{s}(x)),\mathsf{p}(\mathsf{s}(y)))\end{aligned}$$

DP problem for minus hard to solve automatically!

$\mathcal{P}$ : $\mathsf{M}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{M}(\mathsf{p}(\mathsf{s}(x)),\mathsf{p}(\mathsf{s}(y)))$ $\mathcal{R}$ : $\mathsf{p}(\mathsf{s}(x)) \to x$, $\mathsf{minus}(x,0) \to x$, $\mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{minus}(\mathsf{p}(\mathsf{s}(x)),\mathsf{p}(\mathsf{s}(y)))$

- **Rule Removal Processor** (sound & complete)

  $Proc(\,(\mathcal{P},\mathcal{R})\,) = \{\,(\mathcal{P}\backslash\mathcal{P}_\succ,\ \mathcal{R}\backslash\mathcal{R}_\succ)\,\}$ if

  - $l \succ r$ or $l \succsim r$ for all $l \to r$ in $\mathcal{R}$ $\quad \mathcal{R}_\succ \cup \mathcal{R}_\succsim = \mathcal{R}$
  - $s \succ t$ or $s \succsim t$ for all $s \to t$ in $\mathcal{P}$ $\quad \mathcal{P}_\succ \cup \mathcal{P}_\succsim = \mathcal{P}$
  - $\succ$ is monotonic

- **Automation** count number of s-symbols

  try $\mathcal{P}ol(f(t_1,\ldots,t_n)) = \mathcal{P}ol(t_1) + \ldots + \mathcal{P}ol(t_n)$
  or $\mathcal{P}ol(f(t_1,\ldots,t_n)) = 1 + \mathcal{P}ol(t_1) + \ldots + \mathcal{P}ol(t_n)$

# II. DP Processors for Proving Innermost Termination

- **Component for the *evaluation strategy***

  $(\mathcal{P},\mathcal{R},e)$ with $e \in \{\mathbf{t},\mathbf{i}\}$ for **t**ermination or **i**nnermost termination

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

- Infinite (non-innermost) reduction:

$$\begin{aligned}\mathsf{f}(\mathsf{gs0},\mathsf{s0},\mathsf{gs0}) &\to \mathsf{f}(\mathsf{gs0},\mathsf{gs0},\mathsf{gs0})\\ &\to \mathsf{f}(\mathsf{gs0},\mathsf{sg0},\mathsf{gs0})\\ &\to \mathsf{f}(\mathsf{gs0},\mathsf{s0},\mathsf{gs0}) \to \ldots\end{aligned}$$

$\mathcal{P}$ : $\mathsf{M}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{M}(\mathsf{p}(\mathsf{s}(x)),\mathsf{p}(\mathsf{s}(y)))$ $\mathcal{R}\backslash\mathcal{R}_\succ$ : $\mathsf{minus}(x,0) \to x$, $\mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{minus}(\mathsf{p}(\mathsf{s}(x)),\mathsf{p}(\mathsf{s}(y)))$

- **Rule Removal Processor** (sound & complete)

  $Proc(\,(\mathcal{P},\mathcal{R})\,) = \{\,(\mathcal{P}\backslash\mathcal{P}_\succ,\ \mathcal{R}\backslash\mathcal{R}_\succ)\,\}$ if

  - $l \succ r$ or $l \succsim r$ for all $l \to r$ in $\mathcal{R}$ $\quad \mathcal{R}_\succ \cup \mathcal{R}_\succsim = \mathcal{R}$
  - $s \succ t$ or $s \succsim t$ for all $s \to t$ in $\mathcal{P}$ $\quad \mathcal{P}_\succ \cup \mathcal{P}_\succsim = \mathcal{P}$
  - $\succ$ is monotonic

- $(\mathcal{P},\mathcal{R}\backslash\mathcal{R}_\succ)$ is transformed into $\varnothing$ by the Dep. Graph Processor

  Termination is proved automatically!

# II. DP Processors for Proving Innermost Termination

- **Component for the *evaluation strategy***

  $(\mathcal{P},\mathcal{R},e)$ with $e \in \{\mathbf{t},\mathbf{i}\}$ for **t**ermination or **i**nnermost termination

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

- Dependency Pairs:

$$\begin{aligned}\mathsf{F}(\mathsf{g}(x),\mathsf{s}(0),y) &\to \mathsf{F}(y,y,\mathsf{g}(x))\\ \mathsf{F}(\mathsf{g}(x),\mathsf{s}(0),y) &\to \mathsf{G}(x)\\ \mathsf{G}(\mathsf{s}(x)) &\to \mathsf{G}(x)\end{aligned}$$

# Dependency Pair Framework

- **Basic Idea**
  - examine *DP problems* $(\mathcal{P}, \mathcal{R}, e)$
  - a DP problem $(\mathcal{P}, \mathcal{R}, e)$ is *finite* iff
    there is no infinite $(\mathcal{P}, \mathcal{R}, e)$-chain
  - termination techniques should operate on DP problems:

    *DP processor:* $Proc(\,(\mathcal{P}, \mathcal{R}, e)\,) = \{(\mathcal{P}_1, \mathcal{R}_1, e_1), \ldots, (\mathcal{P}_n, \mathcal{R}_n, e_n)\}$

- **Theorem**
  A TRS $\mathcal{R}$ terminates (innermost) iff $(DP(\mathcal{R}), \mathcal{R}, e)$ is finite.

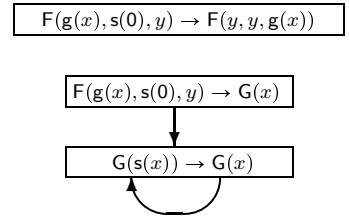
$\mathcal{P}$ :
$$\begin{array}{rcl} \mathsf{F}(\mathsf{g}(x), \mathsf{s}(\mathsf{0}), y) & \to & \mathsf{F}(y, y, \mathsf{g}(x)) \\ \mathsf{F}(\mathsf{g}(x), \mathsf{s}(\mathsf{0}), y) & \to & \mathsf{G}(x) \\ \mathsf{G}(\mathsf{s}(x)) & \to & \mathsf{G}(x) \end{array}$$

$\mathcal{R}$ :
$$\begin{array}{rcl} \mathsf{f}(\mathsf{g}(x), \mathsf{s}(\mathsf{0}), y) & \to & \mathsf{f}(y, y, \mathsf{g}(x)) \\ \mathsf{g}(\mathsf{s}(x)) & \to & \mathsf{s}(\mathsf{g}(x)) \\ \mathsf{g}(\mathsf{0}) & \to & \mathsf{0} \end{array}$$

- **Dependency Graph Processor** (sound & complete)

  $Proc(\,(\mathcal{P}, \mathcal{R}, e)\,) = \{\,(\mathcal{P}_1, \mathcal{R}, e), \ldots, (\mathcal{P}_n, \mathcal{R}, e)\,\}$

  where $\mathcal{P}_1, \ldots, \mathcal{P}_n$
  are the SCCs of the
  $(\mathcal{P}, \mathcal{R}, e)$-dependency graph




- $(\mathcal{P}, \mathcal{R}, e)$-Dependency Graph
  - directed graph whose nodes are the pairs of $\mathcal{P}$
  - arc from $s \to t$ to $v \to w$ iff $s \to t,\ v \to w$ is a $(\mathcal{P}, \mathcal{R}, e)$-chain


# Dependency Pair Framework

- **Basic Idea**
  - examine *DP problems* $(\mathcal{P}, \mathcal{R}, e)$
  - a DP problem $(\mathcal{P}, \mathcal{R}, e)$ is *finite* iff
    there is no infinite $(\mathcal{P}, \mathcal{R}, e)$-chain
  - termination techniques should operate on DP problems:

    *DP processor:* $Proc(\,(\mathcal{P}, \mathcal{R}, e)\,) = \{(\mathcal{P}_1, \mathcal{R}_1, e_1), \ldots, (\mathcal{P}_n, \mathcal{R}_n, e_n)\}$


- **Advantage**

  innermost termination is *easier* to prove than termination

  $\Rightarrow$ $Proc$ often more powerful if $e = \mathbf{i}$

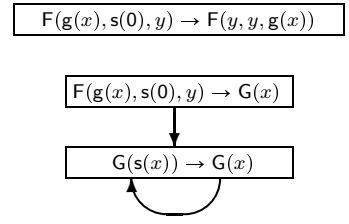

$\mathcal{P}$ :
$$\mathsf{G}(\mathsf{s}(x)) \to \mathsf{G}(x)$$

$\mathcal{R}$ :
$$\begin{array}{rcl} \mathsf{f}(\mathsf{g}(x), \mathsf{s}(\mathsf{0}), y) & \to & \mathsf{f}(y, y, \mathsf{g}(x)) \\ \mathsf{g}(\mathsf{s}(x)) & \to & \mathsf{s}(\mathsf{g}(x)) \\ \mathsf{g}(\mathsf{0}) & \to & \mathsf{0} \end{array}$$

- **Dependency Graph Processor** (sound & complete)

  $Proc(\,(\mathcal{P}, \mathcal{R}, e)\,) = \{\,(\mathcal{P}_1, \mathcal{R}, e), \ldots, (\mathcal{P}_n, \mathcal{R}, e)\,\}$

  where $\mathcal{P}_1, \ldots, \mathcal{P}_n$
  are the SCCs of the
  $(\mathcal{P}, \mathcal{R}, e)$-dependency graph




- $(\mathcal{P}, \mathcal{R}, e)$-Dependency Graph
  - directed graph whose nodes are the pairs of $\mathcal{P}$
  - arc from $s \to t$ to $v \to w$ iff $s \to t,\ v \to w$ is a $(\mathcal{P}, \mathcal{R}, e)$-chain

$\mathcal{P}$ : $\mathsf{G}(\mathsf{s}(x)) \to \mathsf{G}(x)$ $\mathcal{U}(\mathcal{P},\mathcal{R})$ :

- **Usable Rule Processor** (sound)

  $Proc(\,(\mathcal{P},\mathcal{R},\mathbf{i})\,) = \{\,(\mathcal{P},\ \mathcal{U}(\mathcal{P},\mathcal{R}),\ \mathbf{i})\,\}$

- $\mathcal{U}(\mathcal{P},\mathcal{R}) = \varnothing$

$\mathcal{P}$ : $\mathsf{G}(\mathsf{s}(x)) \to \mathsf{G}(x)$ $\mathcal{U}(\mathcal{P},\mathcal{R})$ :

- **Usable Rule Processor** (sound)

  $Proc(\,(\mathcal{P},\mathcal{R},\mathbf{i})\,) = \{\,(\mathcal{P},\ \mathcal{U}(\mathcal{P},\mathcal{R}),\ \mathbf{i})\,\}$

- example is trivial with Reduction Pair Processor: $\mathsf{G}(\mathsf{s}(x)) \succ \mathsf{G}(x)$
  Innermost termination is proved automatically!

- Completeness of processor can be achieved:
  sophisticated representation of the evaluation strategy, not just flag $e$

$\mathcal{P}$: $\mathsf{M}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{M}(x,y)$
$\mathsf{D}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{M}(x,y)$
$\mathsf{D}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{D}(\mathsf{minus}(x,y),\mathsf{s}(y))$

$\mathcal{R}$ : $\mathsf{minus}(x,0) \to x$
$\mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{minus}(x,y)$
$\mathsf{div}(0,\mathsf{s}(y)) \to 0$
$\mathsf{div}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{s}(\mathsf{div}(\mathsf{minus}(x,y),\mathsf{s}(y)))$

- **Advantage of $e = \mathbf{i}$:**

  innermost termination is *easier* to prove than termination

  $\Rightarrow$ $Proc$ often more powerful if $e = \mathbf{i}$

  $\Rightarrow$ prove innermost termination instead of termination, if possible

- **Modular Non-Overlap Check Processor** (sound & complete)

  $Proc(\,(\mathcal{P},\mathcal{R},\mathbf{t})\,) = \{\,(\mathcal{P},\mathcal{R},\mathbf{i})\,\}$ if

  - $\mathcal{R}$ has no critical pairs with $\mathcal{P}$
  - $\mathcal{R}$ is locally confluent

$\mathcal{P}$: $\mathsf{M}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{M}(x,y)$
$\mathsf{D}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{M}(x,y)$
$\mathsf{D}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{D}(\mathsf{minus}(x,y),\mathsf{s}(y))$

$\mathcal{R}$ : $\mathsf{minus}(x,0) \to x$
$\mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{minus}(x,y)$
$\mathsf{div}(0,\mathsf{s}(y)) \to 0$
$\mathsf{div}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{s}(\mathsf{div}(\mathsf{minus}(x,y),\mathsf{s}(y)))$

- **Example:**

  $(\mathcal{P},\mathcal{R},\mathbf{t})$ is replaced by $(\mathcal{P},\mathcal{R},\mathbf{i})$

  it suffices to prove innermost termination

- **Modular Non-Overlap Check Processor** (sound & complete)

  $Proc(\,(\mathcal{P},\mathcal{R},\mathbf{t})\,) = \{\,(\mathcal{P},\mathcal{R},\mathbf{i})\,\}$ if

  - $\mathcal{R}$ has no critical pairs with $\mathcal{P}$
  - $\mathcal{R}$ is locally confluent

# III. DP Processors for Disproving Termination

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

$\mathcal{R}$ is *non-terminating*:

$\mathsf{div}(\mathsf{s}(x),0) \to_{\mathcal{R}} \mathsf{s}(\mathsf{div}(\mathsf{minus}(\mathsf{s}(x),0),0)) \to_{\mathcal{R}} \mathsf{s}(\mathsf{div}(\mathsf{s}(x),0)) \to_{\mathcal{R}} \ldots$

$$\begin{aligned}
\mathcal{P}\colon\ & \mathsf{M}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{M}(x,y)\\
& \mathsf{D}(\mathsf{s}(x),y) \to \mathsf{M}(\mathsf{s}(x),y)\\
& \mathsf{D}(\mathsf{s}(x),y) \to \mathsf{D}(\mathsf{minus}(\mathsf{s}(x),y),y)
\end{aligned}
\qquad
\begin{aligned}
\mathcal{R}\colon\ & \mathsf{minus}(x,0) \to x\\
& \mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{minus}(x,y)\\
& \mathsf{div}(0,y) \to 0\\
& \mathsf{div}(\mathsf{s}(x),y) \to \mathsf{s}(\mathsf{div}(\mathsf{minus}(\mathsf{s}(x),y),y))
\end{aligned}$$

**Definition (Looping DP Problem)**

$(\mathcal{P},\mathcal{R})$ is *looping* if there is a chain $s_1 \to t_1,\ s_2 \to t_2, \ldots, s_k \to t_k$ with

- $t_1\sigma \to_{\mathcal{R}}^* s_2\sigma,\quad t_2\sigma \to_{\mathcal{R}}^* s_3\sigma,\quad \ldots$
- $s_1\sigma$ matches $s_k\sigma,\quad k > 1$ $\qquad (s_1\sigma\ \mu = s_k\sigma)$

**Theorem**

A TRS $\mathcal{R}$ is *looping* iff $(DP(\mathcal{R}),\mathcal{R})$ is *looping*

**Definition (Looping TRS)**

A TRS $\mathcal{R}$ is *looping* if $s \to_{\mathcal{R}}^+ C[s\mu]$ for some term $s$


# III. DP Processors for Disproving Termination

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


$\mathcal{R}$ is *looping*:


$\mathsf{div}(\mathsf{s}(x),0) \to_{\mathcal{R}} \mathsf{s}(\mathsf{div}(\mathsf{minus}(\mathsf{s}(x),0),0)) \to_{\mathcal{R}} \mathsf{s}(\mathsf{div}(\mathsf{s}(x),0)) \to_{\mathcal{R}} \ldots$

**Definition (Looping TRS)**

A TRS $\mathcal{R}$ is *looping* if $s \to_{\mathcal{R}}^+ C[s\mu]$ for some term $s$

$$\begin{aligned}
\mathcal{P}\colon\ & \mathsf{M}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{M}(x,y)\\
& \mathsf{D}(\mathsf{s}(x),y) \to \mathsf{M}(\mathsf{s}(x),y)\\
& \mathsf{D}(\mathsf{s}(x),y) \to \mathsf{D}(\mathsf{minus}(\mathsf{s}(x),y),y)
\end{aligned}
\qquad
\begin{aligned}
\mathcal{R}\colon\ & \mathsf{minus}(x,0) \to x\\
& \mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{minus}(x,y)\\
& \mathsf{div}(0,y) \to 0\\
& \mathsf{div}(\mathsf{s}(x),y) \to \mathsf{s}(\mathsf{div}(\mathsf{minus}(\mathsf{s}(x),y),y))
\end{aligned}$$

**Definition (Looping DP Problem)**

$(\mathcal{P},\mathcal{R})$ is *looping* if there is a chain $s_1 \to t_1,\ s_2 \to t_2, \ldots, s_k \to t_k$ with

- $t_1\sigma \to_{\mathcal{R}}^* s_2\sigma,\quad t_2\sigma \to_{\mathcal{R}}^* s_3\sigma,\quad \ldots$
- $s_1\sigma$ matches $s_k\sigma,\quad k > 1$ $\qquad (s_1\sigma\ \mu = s_k\sigma)$


**Example**

$\mathsf{D}(\mathsf{s}(x_1),\mathsf{s}(y_1)) \to \mathsf{D}(\mathsf{minus}(\mathsf{s}(x_1),y_1),\mathsf{s}(y_1)),\ \mathsf{D}(\mathsf{s}(x_2),\mathsf{s}(y_2)) \to \ldots$
$\mathsf{D}(\mathsf{s}(x_1),\mathsf{s}(0)) \to \mathsf{D}(\mathsf{minus}(\mathsf{s}(x_1),0),\mathsf{s}(0))\qquad \mathsf{D}(\mathsf{s}(x_1),\mathsf{s}(0)) \to \ldots$

loops with $\sigma = [y_1/0,\ x_2/x_1,\ y_2/0]$, $\mu$ =identity

$\mathcal{P}$: $\mathsf{M}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{M}(x,y)$
$\mathsf{D}(\mathsf{s}(x),y) \to \mathsf{M}(\mathsf{s}(x),y)$
$\mathsf{D}(\mathsf{s}(x),y) \to \mathsf{D}(\mathsf{minus}(\mathsf{s}(x),y),y)$

$\mathcal{R}$: $\mathsf{minus}(x,0) \to x$
$\mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{minus}(x,y)$
$\mathsf{div}(0,y) \to 0$
$\mathsf{div}(\mathsf{s}(x),y) \to \mathsf{s}(\mathsf{div}(\mathsf{minus}(\mathsf{s}(x),y),y))$

**Definition (Looping DP Problem)**

$(\mathcal{P},\mathcal{R})$ is *looping* if there is a chain $s_1 \to t_1,\ s_2 \to t_2, \ldots, s_k \to t_k$ with

- $t_1\sigma \to_{\mathcal{R}}^* s_2\sigma, \quad t_2\sigma \to_{\mathcal{R}}^* s_3\sigma, \quad \ldots$
- $s_1\sigma$ matches $s_k\sigma, \quad k > 1 \qquad (s_1\sigma\ \mu = s_k\sigma)$

Loopingness $\Rightarrow$ Infiniteness

**Non-Termination Processor** (sound & complete)

$Proc(\ (\mathcal{P},\mathcal{R})\ )$ = **"no"** if $(\mathcal{P},\mathcal{R})$ is looping

---

$\mathcal{P}$: $\mathsf{M}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{M}(x,y)$
$\mathsf{D}(\mathsf{s}(x),y) \to \mathsf{M}(\mathsf{s}(x),y)$
$\mathsf{D}(\mathsf{s}(x),y) \to \mathsf{D}(\mathsf{minus}(\mathsf{s}(x),y),y)$

$\mathcal{R}$: $\mathsf{minus}(x,0) \to x$
$\mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{minus}(x,y)$
$\mathsf{div}(0,y) \to 0$
$\mathsf{div}(\mathsf{s}(x),y) \to \mathsf{s}(\mathsf{div}(\mathsf{minus}(\mathsf{s}(x),y),y))$

Dependency Graph Processor results in

- $\mathcal{P}_1$: $\mathsf{M}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{M}(x,y)$ $\quad \mathcal{R}$: $\ldots$

  easy to prove

- $\mathcal{P}_2$: $\mathsf{D}(\mathsf{s}(x),y) \to \mathsf{D}(\mathsf{minus}(\mathsf{s}(x),y),y)$ $\quad \mathcal{R}$: $\ldots$

  potentially infinite

Termination techniques help to identify non-terminating parts!

---

$\mathcal{P}$: $\mathsf{M}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{M}(x,y)$
$\mathsf{D}(\mathsf{s}(x),y) \to \mathsf{M}(\mathsf{s}(x),y)$
$\mathsf{D}(\mathsf{s}(x),y) \to \mathsf{D}(\mathsf{minus}(\mathsf{s}(x),y),y)$

$\mathcal{R}$: $\mathsf{minus}(x,0) \to x$
$\mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{minus}(x,y)$
$\mathsf{div}(0,y) \to 0$
$\mathsf{div}(\mathsf{s}(x),y) \to \mathsf{s}(\mathsf{div}(\mathsf{minus}(\mathsf{s}(x),y),y))$

**Definition (Looping DP Problem)**

$(\mathcal{P},\mathcal{R})$ is *looping* if there is a chain $s_1 \to t_1,\ s_2 \to t_2, \ldots, s_k \to t_k$ with

- $t_1\sigma \to_{\mathcal{R}}^* s_2\sigma, \quad t_2\sigma \to_{\mathcal{R}}^* s_3\sigma, \quad \ldots$
- $s_1\sigma$ matches $s_k\sigma, \quad k > 1 \qquad (s_1\sigma\ \mu = s_k\sigma)$

**Advantages of Loop Detection in DP Framework**

- no need to search for context $C[\ldots]$
- other processors remove terminating parts:
  Termination techniques help for disproving termination

---

$\mathcal{P}$: $\mathsf{D}(\mathsf{s}(x),y) \to \mathsf{D}(\mathsf{minus}(\mathsf{s}(x),y),y)$

$\mathcal{R}$: $\mathsf{minus}(x,0) \to x$
$\mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{minus}(x,y)$
$\mathsf{div}(0,y) \to 0$
$\mathsf{div}(\mathsf{s}(x),y) \to \mathsf{s}(\mathsf{div}(\mathsf{minus}(\mathsf{s}(x),y),y))$

**Detect loops by narrowing:**

- start with rhs $t$ of a dependency pair $s \to t$
- narrow $t$ repeatedly: $\quad s \to t \quad \rightsquigarrow_\sigma^* \quad s'$
- check whether $\quad s\sigma$ matches $s'$

$\mathsf{D}(\mathsf{s}(x),y) \to \mathsf{D}(\mathsf{minus}(\mathsf{s}(x),y),y) \quad \rightsquigarrow_{\sigma=[y/0]} \quad \mathsf{D}(\mathsf{s}(x),0)$

$\underbrace{\mathsf{D}(\mathsf{s}(x),y)\ \sigma}_{\mathsf{D}(\mathsf{s}(x),0)}$ matches $\mathsf{D}(\mathsf{s}(x),0)$

$\mathcal{P}$: $\mathsf{F}(x,y) \to \mathsf{G}(x,y)$, $\mathsf{G}(\mathsf{s}(x),y) \to \mathsf{F}(y,y)$ $\quad$ $\mathcal{R}$: $\mathsf{f}(x,y) \to \mathsf{g}(x,y)$, $\mathsf{g}(\mathsf{s}(x),y) \to \mathsf{f}(y,y)$

**Detect loops by narrowing:**

- start with rhs $t$ of a dependency pair $s \to t$
- narrow $t$ repeatedly: $s \to t \quad \rightsquigarrow_\sigma^* \quad s'$
- check whether $s\sigma$ and $s'$ semi-unify

$\mathsf{F}(x,y) \to \mathsf{G}(x,y) \quad \rightsquigarrow_{\sigma=[x/\mathsf{s}(x)]} \quad \mathsf{F}(y,y)$

$\underbrace{\mathsf{F}(x,y)\,\sigma}_{\mathsf{F}(\mathsf{s}(x),y)}$ unifies with $\mathsf{F}(y,y)$

**Heuristic for loop detection of $(\mathcal{P},\mathcal{R})$ in AProVE:**

- If $\mathcal{P} \cup \mathcal{R}$ left-linear: backward narrowing
- If $\mathcal{P} \cup \mathcal{R}$ right-linear: forward narrowing
- If $\mathcal{P} \cup \mathcal{R}$ not left- or right-linear: backward narrowing into variables

**To obtain finite search space:**

each rule may only applied $n$ times for narrowing

**Analogous approach to disprove *innermost* termination**

$\mathcal{P}$: $\mathsf{F}(0,1,x) \to \mathsf{F}(x,x,x)$ $\quad$ $\mathcal{R}$: $\mathsf{f}(0,1,x) \to \mathsf{f}(x,x,x)$, $\mathsf{g}(y,z) \to y$, $\mathsf{g}(y,z) \to z$

$\mathsf{F}(\mathsf{g}(0,z),\mathsf{g}(y,1),x) \leftsquigarrow \mathsf{F}(\mathsf{g}(0,z),\underline{1},x) \leftsquigarrow \mathsf{F}(\underline{0},1,x) \to \mathsf{F}(x,x,x)$

$\mathsf{F}(\mathsf{g}(0,z),\mathsf{g}(y,1),x)$ unifies with $\mathsf{F}(x,x,x)$

**Detect loops by backward narrowing:**

- start with lhs $s$ of a dependency pair $s \to t$
- narrow $s$ with reversed rules: $t' \quad \leftsquigarrow_\sigma^* \quad s \to t$
- check whether $t'$ and $t\sigma$ semi-unify

## IV. DP Processors from Other Techniques

- **Termination Technique**

  $TT$ maps TRSs to TRSs

  $TT$ is is *sound:* if termination of $TT(\mathcal{R})$ implies termination of $\mathcal{R}$

  $TT$ is *complete:* if termination of $\mathcal{R}$ implies termination of $TT(\mathcal{R})$

- Termination techniques can be transformational or *conventional*:

$$TT(\mathcal{R}) = \begin{cases} \varnothing, & \text{if termination of } \mathcal{R} \text{ can be proved} \\ \mathcal{R}, & \text{otherwise} \end{cases}$$

## IV. DP Processors from Other Techniques

- **Advantages**
  - different techniques can be used for different sub-problems
  - combines benefits of different methods and of dependency pair techniques
  - termination techniques with restricted applicability can be used, even if they are not applicable to the whole TRS

- **Termination Technique Processor** (sound & complete)

  $Proc(\,(\mathcal{P},\mathcal{R},e)\,) = \{\,(DP(\mathcal{R}'),\mathcal{R}',\mathbf{t})\,\}$

  where $\mathcal{R}' = TT(\mathcal{P} \cup \mathcal{R})$

## Challenging Example

$$
\begin{aligned}
\mathsf{p}(\mathsf{s}(0)) &\to 0\\
\mathsf{p}(\mathsf{s}(\mathsf{s}(x))) &\to \mathsf{s}(\mathsf{p}(\mathsf{s}(x)))\\
\mathsf{fact}(0) &\to \mathsf{s}(0)\\
\mathsf{fact}(\mathsf{s}(x)) &\to \mathsf{times}(\mathsf{s}(x),\mathsf{fact}(\mathsf{p}(\mathsf{s}(x))))\\
\mathsf{times}(\ldots) &\to \ldots\\
\mathsf{plus}(\ldots) &\to \ldots
\end{aligned}
$$

- Dependency Graph Processor: 4 DP problems for p, fact, times, plus
- DP problem for fact hard to solve automatically!
- String reversal not applicable: no SRS

## IV. DP Processors from Other Techniques

- **Example**: String Reversal
  - only applicable on *string rewrite systems (SRS)*
    arity$(f) = 1$ for all $f$
  - $TT(\mathcal{R}) = \mathcal{R}^{-1} \qquad = \{l^{-1} \to r^{-1} \mid l \to r \in \mathcal{R}\}$ (sound & complete)
  - $\mathcal{R}$ : $\mathsf{a}(\mathsf{b}(\mathsf{b}(x))) \to \mathsf{b}(\mathsf{a}(x))$
    $\mathcal{R}^{-1}$ : $\mathsf{b}(\mathsf{b}(\mathsf{a}(x))) \to \mathsf{a}(\mathsf{b}(x))$

- **Termination Technique Processor** (sound & complete)

  $Proc(\,(\mathcal{P},\mathcal{R},e)\,) = \{\,(DP(\mathcal{R}'),\mathcal{R}',\mathbf{t})\,\}$

  where $\mathcal{R}' = TT(\mathcal{P} \cup \mathcal{R})$

$\mathcal{P}$ : $\mathsf{FACT}(\mathsf{s}(x)) \to \mathsf{FACT}(\mathsf{p}(\mathsf{s}(x)))$

$$
\mathcal{R}: \quad
\begin{aligned}
\mathsf{p}(\mathsf{s}(0)) &\to 0\\
\mathsf{p}(\mathsf{s}(\mathsf{s}(x))) &\to \mathsf{s}(\mathsf{p}(\mathsf{s}(x)))\\
\mathsf{fact}(\ldots) &\to \ldots\\
\mathsf{times}(\ldots) &\to \ldots\\
\mathsf{plus}(\ldots) &\to \ldots
\end{aligned}
$$

- **Modular Non-Overlap Check Processor** (sound & complete)

  $Proc(\,(\mathcal{P},\mathcal{R},\mathbf{t})\,) = \{\,(\mathcal{P},\mathcal{R},\mathbf{i})\,\}$ if

  - $\mathcal{R}$ has no critical pairs with $\mathcal{P}$
  - $\mathcal{R}$ is locally confluent

- $(\mathcal{P},\mathcal{R},\mathbf{t})$ is replaced by $(\mathcal{P},\mathcal{R},\mathbf{i})$

  it suffices to prove innermost termination

$\mathcal{P}$ : FACT(s($x$)) → FACT(p(s($x$)))

$\mathcal{R}$ :
p(s(0)) → 0
p(s(s($x$))) → s(p(s($x$)))
fact(...) → ...
times(...) → ...
plus(...) → ...

- **Usable Rule Processor** (sound)

  $Proc(\,(\mathcal{P},\mathcal{R},\mathbf{i})\,) = \{\,(\mathcal{P},\ \mathcal{U}(\mathcal{P},\mathcal{R}),\ \mathbf{i})\,\}$

- $\mathcal{U}(\mathcal{P},\mathcal{R})$ = { p(s(0)) → 0, p(s(s($x$))) → s(p(s($x$))) }

$\mathcal{P}$ : FACT(s($x$)) → FACT(p(s($x$)))

$\mathcal{R}$ : p(s(s($x$))) → s(p(s($x$)))

- **Termination Technique Processor** (sound & complete)

  $Proc(\,(\mathcal{P},\mathcal{R},e)\,) = \{\,(DP(\mathcal{R}'),\mathcal{R}',\mathbf{t})\,\}$

  where $\mathcal{R}' = TT(\mathcal{P}\cup\mathcal{R})$

- String reversal applicable: arity($f$) = 1 for all $f$

$\mathcal{P}$ : FACT(s($x$)) → FACT(p(s($x$)))

$\mathcal{R}$ :
p(s(0)) → 0
p(s(s($x$))) → s(p(s($x$)))

- **Rule Removal Processor** (sound & complete)

  $Proc(\,(\mathcal{P},\mathcal{R},e)\,) = \{\,(\mathcal{P}\backslash\mathcal{P}_{\succ},\ \mathcal{R}\backslash\mathcal{R}_{\succ},\ e)\,\}$ if

  - $l \succ r$ or $l \succsim r$ for all $l \to r$ in $\mathcal{R}$
  - $s \succ t$ or $s \succsim t$ for all $s \to t$ in $\mathcal{P}$
  - $\succ$ is monotonic

  $\mathcal{R}_{\succ} \cup \mathcal{R}_{\succsim} = \mathcal{R}$
  $\mathcal{P}_{\succ} \cup \mathcal{P}_{\succsim} = \mathcal{P}$

- **Automation** count number of s-symbols

  try $\mathcal{P}ol(f(t_1,\ldots,t_n)) = \mathcal{P}ol(t_1) + \ldots + \mathcal{P}ol(t_n)$
  or $\mathcal{P}ol(f(t_1,\ldots,t_n)) = \mathcal{P}ol(t_1) + \ldots + \mathcal{P}ol(t_n) + 1$

$\mathcal{P}$ : FACT(s($x$)) → FACT(p(s($x$)))

$\mathcal{R}$ : p(s(s($x$))) → s(p(s($x$)))

- **Termination Technique Processor** (sound & complete)

  $Proc(\,(\mathcal{P},\mathcal{R},e)\,) = \{\,(DP(\mathcal{R}'),\mathcal{R}',\mathbf{t})\,\}$

  where $\mathcal{R}' = \mathcal{P}^{-1}\cup\mathcal{R}^{-1}$

$\mathcal{P}^{-1}$ : s(FACT($x$)) → s(p(FACT($x$)))

$\mathcal{R}^{-1}$ : s(s(p($x$))) → s(p(s($x$)))

$DP(\mathcal{R}')$ : $\mathsf{S}(\mathsf{FACT}(x)) \to \mathsf{S}(\mathsf{p}(\mathsf{FACT}(x)))$
$\mathsf{S}(\mathsf{s}(\mathsf{p}(x))) \to \mathsf{S}(\mathsf{p}(\mathsf{s}(x)))$
$\mathsf{S}(\mathsf{s}(\mathsf{p}(x))) \to \mathsf{S}(x)$

$\mathcal{R}'$ : $\mathsf{s}(\mathsf{FACT}(x)) \to \mathsf{s}(\mathsf{p}(\mathsf{FACT}(x)))$
$\mathsf{s}(\mathsf{s}(\mathsf{p}(x))) \to \mathsf{s}(\mathsf{p}(\mathsf{s}(x)))$

- **Termination Technique Processor** (sound & complete)

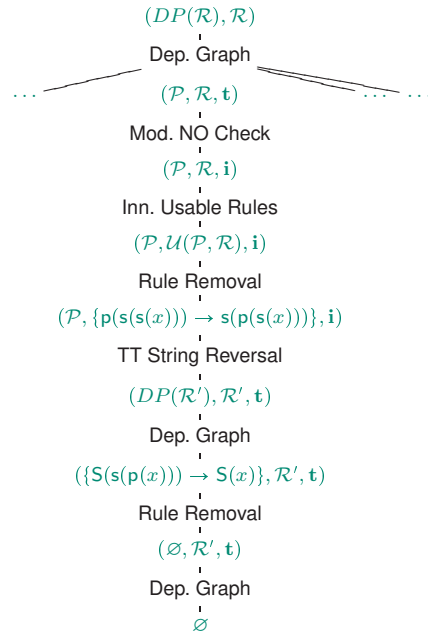
$Proc(\,(\mathcal{P},\mathcal{R},e)\,) = \{\,(DP(\mathcal{R}'),\mathcal{R}',\mathbf{t})\,\}$

where $\mathcal{R}' = \mathcal{P}^{-1} \cup \mathcal{R}^{-1}$

$\mathcal{P}^{-1}$ : $\mathsf{s}(\mathsf{FACT}(x)) \to \mathsf{s}(\mathsf{p}(\mathsf{FACT}(x)))$

$\mathcal{R}^{-1}$ : $\mathsf{s}(\mathsf{s}(\mathsf{p}(x))) \to \mathsf{s}(\mathsf{p}(\mathsf{s}(x)))$

Termination is proved automatically!




$\mathcal{P}$ :

$\mathsf{S}(\mathsf{s}(\mathsf{p}(x))) \to \mathsf{S}(x)$

$\mathcal{R}$ : $\mathsf{s}(\mathsf{FACT}(x)) \to \mathsf{s}(\mathsf{p}(\mathsf{FACT}(x)))$
$\mathsf{s}(\mathsf{s}(\mathsf{p}(x))) \to \mathsf{s}(\mathsf{p}(\mathsf{s}(x)))$

- **Dependency Graph Processor** (sound & complete)

$Proc(\,(\mathcal{P},\mathcal{R},e)\,) = \{\,(\mathcal{P}_1,\mathcal{R},e),\ldots,(\mathcal{P}_n,\mathcal{R},e)\,\}$

where $\mathcal{P}_1,\ldots,\mathcal{P}_n$ are the SCCs of the $(\mathcal{P},\mathcal{R},e)$-dependency graph

# Semantic Labelling

$\mathcal{R}$ : $\mathsf{f}(\mathsf{f}(x)) \to \mathsf{f}(\mathsf{g}(\mathsf{f}(x)))$

- **Choose model for TRS $\mathcal{R}$**
  - carrier set $M = \{0,1\}$
  - for every $n$-ary symbol $f$ choose interpretation $f_M : M^n \to M$

$\mathsf{f}_M(x) = 0 \qquad \mathsf{g}_M(x) = x + 1$

For every variable assignment $\alpha : \mathcal{V} \to M$

$\alpha(\mathsf{f}(\mathsf{f}(x))) = 0 = \alpha(\mathsf{f}(\mathsf{g}(\mathsf{f}(x))))$

$\Rightarrow$ is a model!

# Semantic Labelling

$$\mathcal{R}: \quad \mathsf{f}(\mathsf{f}(x)) \;\to\; \mathsf{f}(\mathsf{g}(\mathsf{f}(x)))$$

- **Choose model for TRS $\mathcal{R}$**
  - carrier set $M = \{0, 1\}$
  - for every $n$-ary symbol $f$ choose interpretation $f_M : M^n \to M$

$$\mathsf{f}_M(x) = 0 \qquad \mathsf{g}_M(x) = x + 1$$

- **Label every symbol by interpretation of its argument(s)**

$$\begin{aligned} \mathsf{f}_0(\mathsf{f}_0(x)) &\to \mathsf{f}_1(\mathsf{g}_0(\mathsf{f}_0(x))) \\ \mathsf{f}_0(\mathsf{f}_1(x)) &\to \mathsf{f}_1(\mathsf{g}_0(\mathsf{f}_1(x))) \end{aligned}$$

- **$\mathcal{R}$ terminates iff labelled TRS $\overline{\mathcal{R}}$ terminates**

  termination can be proved by LPO if $\mathsf{f}_0$ has highest precedence

# Semantic Labelling

$$\begin{aligned} \mathsf{p}_1(\mathsf{s}_0(0)) &\to 0 \\ \mathsf{p}_{n+2}(\mathsf{s}_{n+1}(\mathsf{s}_n(x))) &\to \mathsf{s}_n(\mathsf{p}_{n+1}(\mathsf{s}_n(x))) \\ \mathsf{fact}_0(0) &\to \mathsf{s}_0(0) \\ \mathsf{fact}_{n+1}(\mathsf{s}_n(x)) &\to \mathsf{times}_{(n+1,n+1)}(\mathsf{s}_n(x), \mathsf{fact}_n(\mathsf{p}_{n+1}(\mathsf{s}_n(x)))) \end{aligned}$$

- **Choose model for TRS $\mathcal{R}$**

$$\begin{aligned} 0_M(x) &= 0 \\ \mathsf{s}_M(x) &= x + 1 \\ \mathsf{p}_M(x) &= \begin{cases} 0, & \text{if } x = 0 \\ x - 1, & \text{if } x > 0 \end{cases} \\ \mathsf{fact}_M(x) &= x + 1 \\ \mathsf{times}_M(x, y) &= y + 1 \end{aligned}$$

- **Label every symbol by interpretation of its argument(s)**
- **$\mathcal{R}$ terminates iff labelled TRS $\overline{\mathcal{R}}$ terminates**

  termination can be proved by LPO if $\mathsf{fact}_{n+1} > \mathsf{fact}_n$

# Semantic Labelling

$$\begin{aligned} \mathsf{p}\ (\mathsf{s}\ (0)) &\to 0 \\ \mathsf{p}\ (\mathsf{s}\ (\mathsf{s}\ (x))) &\to \mathsf{s}\ (\mathsf{p}\ (\mathsf{s}\ (x))) \\ \mathsf{fact}\ (0) &\to \mathsf{s}\ (0) \\ \mathsf{fact}\ (\mathsf{s}\ (x)) &\to \mathsf{times}\ (\mathsf{s}\ (x), \mathsf{fact}\ (\mathsf{p}\ (\mathsf{s}\ (x)))) \end{aligned}$$

- **Choose model for TRS $\mathcal{R}$**
  - carrier set $M = \mathbb{N}$
  - for every $n$-ary symbol $f$ choose interpretation $f_M : M^n \to M$

# Match–Bounds for String Rewriting

$$\mathcal{R}: \quad \mathsf{a}(\mathsf{b}(\mathsf{a}(x))) \;\to\; \mathsf{a}(\mathsf{b}(\mathsf{b}(\mathsf{b}(\mathsf{a}(x)))))$$

- **match($\mathcal{R}$)**
  - label symbols of lhs by arbitrary natural numbers
  - label all symbols of rhs by *1 + minimum label of lhs*

$$\begin{aligned} \mathsf{a}_0(\mathsf{b}_0(\mathsf{a}_0(x))) &\to \mathsf{a}_1(\mathsf{b}_1(\mathsf{b}_1(\mathsf{b}_1(\mathsf{a}_1(x))))) \\ \mathsf{a}_0(\mathsf{b}_1(\mathsf{a}_5(x))) &\to \mathsf{a}_1(\mathsf{b}_1(\mathsf{b}_1(\mathsf{b}_1(\mathsf{a}_1(x))))) \\ \mathsf{a}_3(\mathsf{b}_1(\mathsf{a}_5(x))) &\to \mathsf{a}_2(\mathsf{b}_2(\mathsf{b}_2(\mathsf{b}_2(\mathsf{a}_2(x))))) \\ &\vdots \end{aligned}$$

# Match–Bounds for String Rewriting

$\mathcal{R}$ : $\quad \mathsf{a}(\mathsf{b}(\mathsf{a}(x))) \;\to\; \mathsf{a}(\mathsf{b}(\mathsf{b}(\mathsf{b}(\mathsf{a}(x)))))$

- **Thm 1:** If all symbols in $t$ are labelled with 0 and for all $t \to^*_{\mathbf{match}(\mathcal{R})} s$, the labels in $s$ are $\leqslant n$ (*match-bound*), then $t$ is terminating w.r.t. $\mathcal{R}$.

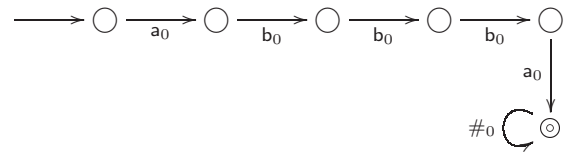
- **Right-Forward Closures** (Dershowitz, 81):

$\mathcal{R}_\# = \mathcal{R} \cup \{l_1\ \# \to r \mid l \to r \in \mathcal{R},\ l = l_1\ l_2,\ l_i \neq \varepsilon\}$

$\mathcal{R}_\#$ :
$$\begin{aligned} \mathsf{a}(\mathsf{b}(\mathsf{a}(x))) &\to \mathsf{a}(\mathsf{b}(\mathsf{b}(\mathsf{b}(\mathsf{a}(x))))) \\ \mathsf{a}(\mathsf{b}(\#(x))) &\to \mathsf{a}(\mathsf{b}(\mathsf{b}(\mathsf{b}(\mathsf{a}(x))))) \\ \mathsf{a}(\#(x)) &\to \mathsf{a}(\mathsf{b}(\mathsf{b}(\mathsf{b}(\mathsf{a}(x))))) \end{aligned}$$

# Match–Bounds for String Rewriting

- Start with automaton accepting all $r\ \#^k$ labelled with 0




**Construct finite automaton accepting** $\{s \mid r\ \#^k \to^*_{\mathbf{match}(\mathcal{R}_\#)} s\}$

$\mathcal{R}_\#$ :
$$\begin{aligned} \mathsf{a}(\mathsf{b}(\mathsf{a}(x))) &\to \mathsf{a}(\mathsf{b}(\mathsf{b}(\mathsf{b}(\mathsf{a}(x))))) \\ \mathsf{a}(\mathsf{b}(\#(x))) &\to \mathsf{a}(\mathsf{b}(\mathsf{b}(\mathsf{b}(\mathsf{a}(x))))) \\ \mathsf{a}(\#(x)) &\to \mathsf{a}(\mathsf{b}(\mathsf{b}(\mathsf{b}(\mathsf{a}(x))))) \end{aligned}$$

# Match–Bounds for String Rewriting

$\mathcal{R}$ : $\quad \mathsf{a}(\mathsf{b}(\mathsf{a}(x))) \;\to\; \mathsf{a}(\mathsf{b}(\mathsf{b}(\mathsf{b}(\mathsf{a}(x)))))$

- **Thm 1:** If all symbols in $t$ are labelled with 0 and for all $t \to^*_{\mathbf{match}(\mathcal{R})} s$, the labels in $s$ are $\leqslant n$ (*match-bound*), then $t$ is terminating w.r.t. $\mathcal{R}$.

- **Thm 2:** If $r\ \#^k$ terminates w.r.t. $\mathcal{R}_\#$ for all rhs's $r$ and all $k \in \mathbb{N}$, then $\mathcal{R}$ is terminating.

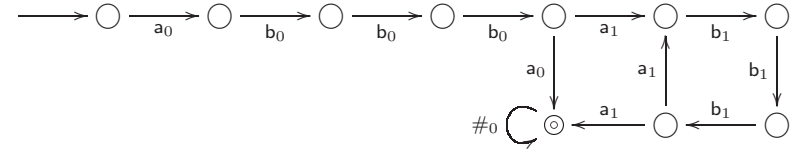
- **Thm 3:** If all symbols in $r\ \#^k$ are labelled with 0 and for all $r\ \#^k \to^*_{\mathbf{match}(\mathcal{R}_\#)} s$, labels in $s$ are $\leqslant n$ (*match-bound*), then $\mathcal{R}$ is terminating.

**Construct finite automaton accepting** $\{s \mid r\ \#^k \to^*_{\mathbf{match}(\mathcal{R}_\#)} s\}$

$\mathcal{R}_\#$ :
$$\begin{aligned} \mathsf{a}(\mathsf{b}(\mathsf{a}(x))) &\to \mathsf{a}(\mathsf{b}(\mathsf{b}(\mathsf{b}(\mathsf{a}(x))))) \\ \mathsf{a}(\mathsf{b}(\#(x))) &\to \mathsf{a}(\mathsf{b}(\mathsf{b}(\mathsf{b}(\mathsf{a}(x))))) \\ \mathsf{a}(\#(x)) &\to \mathsf{a}(\mathsf{b}(\mathsf{b}(\mathsf{b}(\mathsf{a}(x))))) \end{aligned}$$

# Match–Bounds for String Rewriting

- If there is path from $q_1$ to $q_2$ with lhs of $\mathbf{match}(\mathcal{R}_\#)$, then check if there is path from $q_1$ to $q_2$ with corresponding rhs.
  - If rhs = $a\ w$ and there is path from $q_1'$ to $q_2$ with $w$, then add edge from $q_1$ to $q_1'$ with $a$.
  - Otherwise, add new path from $q_1$ to $q_2$ with $a\ w$.




**Construct finite automaton accepting** $\{s \mid r\ \#^k \to^*_{\mathbf{match}(\mathcal{R}_\#)} s\}$

$\mathcal{R}_\#$ :
$$\begin{aligned} \mathsf{a}(\mathsf{b}(\mathsf{a}(x))) &\to \mathsf{a}(\mathsf{b}(\mathsf{b}(\mathsf{b}(\mathsf{a}(x))))) \\ \mathsf{a}(\mathsf{b}(\#(x))) &\to \mathsf{a}(\mathsf{b}(\mathsf{b}(\mathsf{b}(\mathsf{a}(x))))) \\ \mathsf{a}(\#(x)) &\to \mathsf{a}(\mathsf{b}(\mathsf{b}(\mathsf{b}(\mathsf{a}(x))))) \end{aligned}$$

## Implementation

- **Procedure**
  1. Start with the *initial* DP problem $(DP(\mathcal{R}), \mathcal{R})$.
  2. Transform a remaining DP problem by a sound processor.
  3. If result is **"no"** and all processors were complete, return **"no"**.
     If there is no remaining DP problem, then return **"yes"**.
     Otherwise go to 2.
- **Strategy**
  - decide which DP processor to use in Step 2
  - use fast processors first
  - only use slower more powerful processors on DP problems that cannot be solved by fast processors

## Termination of Term Rewriting

- DP Framework combines many different termination techniques
  - termination techniques also help for disproving termination
  - non-termination techniques also help for proving termination
  - also possible for innermost termination
- Current work
  - development of new DP processors
  - development of strategies how to apply DP processors
  - more efficient algorithms (e.g., using SAT-solvers)
- **AProVE**: implements the DP framework
  - winner of *Internat. Competitions of Termination Tools '04 - '07* both for *termination* and *non-termination* of TRSs

`http://aprove.informatik.rwth-aachen.de/`

## Strategy of AProVE

Use the first applicable processor from the following list:

1. Dependency Graph Processor
2. Modular Non-Overlap Check Processor
3. Usable Rule Processor
4. A–Transformation Processor
5. Size–Change Processor
6. DP Transformation Processors (in "safe" cases)
7. Rule Removal Processor
8. Reduction Pair Processor: linear polynomials over $\{0, 1\}$
9. Termination Technique Processor: Match–Bounds
10. Reduction Pair Processor: LPO with strict precedence
11. DP Transformation Processors (up to a certain limit)
12. Reduction Pair Processor: linear polynomials over $\{-1, 0, 1\}$
13. Non-Termination Processor
14. Reduction Pair Processor: non-linear polynomials over $\{0, 1\}$
15. Reduction Pair Processor: LPO with non-strict precedence
16. Termination Technique Processor: String Reversal
17. Forward–Instantiation Processor
18. Termination Technique Processor: Semantic Labelling

## Termination of Logic Programs

- Two kinds of approaches:
  - **Direct** approaches:
    work directly on the logic program (LP)
  - **Transformational** approaches:
    transform LP into TRS and prove termination of TRS instead

    **Advantage:** make all TRS-methods available for LPs as well
- Existing transformational approaches:
  - only work for restricted class of LPs (well-moded LPs) or
  - generate TRSs which are hard for automated termination analysis
- New transformational approach (LOPSTR 2006):
  - works for any LP
  - well suitable for automation
  - also sound for LP without occur check

# Termination of Logic Programs

$\mathsf{app}(\mathsf{nil}, XS, XS).$

$\mathsf{app}(\mathsf{cons}(X, XS), YS, \mathsf{cons}(X, ZS)) :\!- \mathsf{app}(XS, YS, ZS).$

- **Classical Approach:**
  - Distinguish in- and output positions: $p(\vec{s}, \vec{t})$ if $\vec{s}$ is input, $\vec{t}$ is output.
  - For each fact $p(\vec{s}, \vec{t})$, add the rule $p_{in}(\vec{s}) \to p_{out}(\vec{t})$.
  - For each clause $p(\vec{s}, \vec{t}) :\!- q(\vec{u}, \vec{v})$, add the rules:

$$\begin{aligned} p_{in}(\vec{s}) &\to u(q_{in}(\vec{u}), \mathcal{V}(\vec{s})) \\ u(q_{out}(\vec{v}), \mathcal{V}(\vec{s})) &\to p_{out}(\vec{t}) \end{aligned}$$

- For $\mathsf{app}(in, in, out)$:

$$\begin{aligned} \mathsf{app}_{in}(\mathsf{nil}, XS) &\to \mathsf{app}_{out}(XS) \\ \mathsf{app}_{in}(\mathsf{cons}(X, XS), YS) &\to \mathsf{u}(\mathsf{app}_{in}(XS, YS), X, XS, YS) \\ \mathsf{u}(\mathsf{app}_{out}(ZS), X, XS, YS) &\to \mathsf{app}_{out}(\mathsf{cons}(X, ZS)) \end{aligned}$$

# Termination of Logic Programs

$\mathsf{app}(\mathsf{nil}, XS, XS).$

$\mathsf{app}(\mathsf{cons}(X, XS), YS, \mathsf{cons}(X, ZS)) :\!- \mathsf{app}(XS, YS, ZS).$

- **New Approach:**
  - Do not distinguish in- and output positions
  - Variables in TRS may be instantiated by infinite constructor terms
  - **Thm:** If TRS terminates for all terms $\mathsf{app}(t_1, t_2, t_3)$ with $t_1$ finite, then LP terminates for all queries $\mathsf{app}(in, out, out)$.
  - DP Framework easy to extend to infinitary constructor rewriting

$$\begin{aligned} \mathsf{app}_{in}(\mathsf{nil}, XS, XS) &\to \mathsf{app}_{out}(\mathsf{nil}, XS, XS) \\ \mathsf{app}_{in}(\mathsf{cons}(X, XS), YS, \mathsf{cons}(X, ZS)) &\to \mathsf{u}(\mathsf{app}_{in}(XS, YS, ZS), X, XS, YS, ZS) \\ \mathsf{u}(\mathsf{app}_{out}(XS, YS, ZS), X, XS, YS, ZS) &\to \mathsf{app}_{out}(\mathsf{cons}(X, XS), YS, \mathsf{cons}(X, ZS)) \end{aligned}$$

# Termination of Logic Programs

$\mathsf{app}(\mathsf{nil}, XS, XS).$

$\mathsf{app}(\mathsf{cons}(X, XS), YS, \mathsf{cons}(X, ZS)) :\!- \mathsf{app}(XS, YS, ZS).$

- **Classical Approach:**
  - Distinguish in- and output positions: $p(\vec{s}, \vec{t})$ if $\vec{s}$ is input, $\vec{t}$ is output.
  - **Thm:** If TRS terminates, then LP terminates for all queries where input is ground.
  - Only works for well-moded LPs (otherwise $\mathcal{V}(r) \not\subseteq \mathcal{V}(l)$)

- For $\mathsf{app}(in, out, out)$:

$$\begin{aligned} \mathsf{app}_{in}(\mathsf{nil}) &\to \mathsf{app}_{out}(XS, XS) \\ \mathsf{app}_{in}(\mathsf{cons}(X, XS)) &\to \mathsf{u}(\mathsf{app}_{in}(XS), X, XS)) \\ \mathsf{u}(\mathsf{app}_{out}(YS, ZS), X, XS) &\to \mathsf{app}_{out}(YS, \mathsf{cons}(X, ZS)) \end{aligned}$$

# Termination of Logic Programs

$$\mathsf{p}(\mathsf{s}(X)) \quad :\!- \quad \mathsf{p}(X).$$

- Transformed TRS:

$$\begin{aligned} \mathsf{p}_{in}(\mathsf{s}(X)) &\to \mathsf{u}(\mathsf{p}_{in}(X), X) \\ \mathsf{u}(\mathsf{p}_{out}(X), X) &\to \mathsf{p}_{out}(\mathsf{s}(X)) \end{aligned}$$

- TRS terminates for all terms $\mathsf{p}_{in}(t)$ with $t$ finite
  $\Rightarrow$ LP terminates for all queries $\mathsf{p}(in)$.

- TRS does not terminate for all terms $\mathsf{p}_{in}(t)$,
  LP does not terminate for all queries $\mathsf{p}(out)$.

$$\mathsf{p}(X) \vdash_{[X/\mathsf{s}(X_1)]} \mathsf{p}(X_1) \vdash_{[X_1/\mathsf{s}(X_2)]} \mathsf{p}(X_2) \vdash_{[X_2/\mathsf{s}(X_3)]} \dots$$

- Infinite terms simulate unification of LPs by matching in TRSs

# Termination of Logic Programs

$$\mathsf{equal}(X, X).$$
$$\mathsf{p}(X) :\!\!- \mathsf{equal}(X, \mathsf{s}(X)), \mathsf{p}(X).$$

- Transformed TRS:

$$\begin{aligned}
\mathsf{equal}_{in}(X, X) &\to \mathsf{equal}_{out}(X, X)\\
\mathsf{p}_{in}(X) &\to \mathsf{u}_1(\mathsf{equal}_{in}(X, \mathsf{s}(X)), X)\\
\mathsf{u}_1(\mathsf{equal}_{out}(X, \mathsf{s}(X)), X) &\to \mathsf{u}_2(\mathsf{p}_{in}(X), X)\\
\mathsf{u}_2(\mathsf{p}_{out}(X), X) &\to \mathsf{p}_{out}(X)
\end{aligned}$$

- TRS does not terminate for all terms $\mathsf{p}_{in}(t)$,
  LP does not terminate for all queries $\mathsf{p}(out)$.

$$\mathsf{p}(X) \vdash^2_{[X/\mathsf{s}(\mathsf{s}(\ldots))]} \mathsf{p}(\mathsf{s}(\mathsf{s}(\ldots))) \vdash^2_{[X/\mathsf{s}(\mathsf{s}(\ldots))]} \mathsf{p}(\mathsf{s}(\mathsf{s}(\ldots))) \vdash^2_{[X/\mathsf{s}(\mathsf{s}(\ldots))]} \cdots$$

- Infinite terms also simulate occur-check-free unification

# Termination of HASKELL-Programs

- Challenges:
  - HASKELL has a lazy evaluation strategy.
    For TRSs, one proves termination of all (innermost) reductions.
  - HASKELL's equations are handled from top to bottom.
    For TRSs, *any* rule may be used for rewriting.
  - HASKELL has polymorphic types.
    TRSs are untyped.
  - In HASKELL-programs, often only some functions terminate.
    TRS-methods try to prove termination of *all* terms.
- Only few existing approaches,
  do not permit integration of TRS-techniques.
- New approach to use TRS-techniques for HASKELL (RTA 2006)

# Termination of Logic Programs

- New approach implemented in AProVE
- 296 examples (including the TPDB 2005)
  - **TALP** (classical transformation): 163 successes
  - **cTI** (direct approach): 167 successes
  - **TerminWeb** (direct approach): 178 successes
  - **AProVE** (new transformation): 208 successes
- New transformational approach is better than the old one
  (TALP is proper subset of AProVE)
- Transformational and direct approaches both have their advantages:

  16 examples where cTI succeeds and AProVE fails
  16 examples where TerminWeb succeeds and AProVE fails
  57 examples where AProVE succeeds and cTI fails
  46 examples where AProVE succeeds and TerminWeb fails

# Termination of HASKELL-Programs

$$\begin{aligned}
\mathsf{from}\, x &= \mathsf{Cons}\, x\, (\mathsf{from}\, (\mathsf{S}\, x))\\
\mathsf{take}\, \mathsf{Z}\, xs &= \mathsf{Nil}\\
\mathsf{take}\, n\, \mathsf{Nil} &= \mathsf{Nil}\\
\mathsf{take}\, (\mathsf{S}\, n)\, (\mathsf{Cons}\, x\, xs) &= \mathsf{Cons}\, x\, (\mathsf{take}\, n\, xs)
\end{aligned}$$

- **Evaluation Relation** $\to_{\mathsf{H}}$

$$\begin{aligned}
& \mathsf{from}\, m\\
\to_{\mathsf{H}}\ & \mathsf{Cons}\, m\, (\mathsf{from}\, (\mathsf{S}\, m))\\
\to_{\mathsf{H}}\ & \mathsf{Cons}\, m\, (\mathsf{Cons}\, (\mathsf{S}\, m)\, (\mathsf{from}\, (\mathsf{S}\, m)))\\
\to_{\mathsf{H}}\ & \ldots
\end{aligned}$$

# Termination of HASKELL-Programs

$\mathsf{from}\, x = \mathsf{Cons}\, x\, (\mathsf{from}\, (\mathsf{S}\, x))$

$\mathsf{take}\, \mathsf{Z}\, xs = \mathsf{Nil}$
$\mathsf{take}\, n\, \mathsf{Nil} = \mathsf{Nil}$
$\mathsf{take}\, (\mathsf{S}\, n)\, (\mathsf{Cons}\, x\, xs) = \mathsf{Cons}\, x\, (\mathsf{take}\, n\, xs)$

- **Evaluation Relation** $\to_{\mathsf{H}}$

$\mathsf{take}\, (\mathsf{S}\, \mathsf{Z})\, (\mathsf{from}\, m)$
$\to_{\mathsf{H}}\ \mathsf{take}\, (\mathsf{S}\, \mathsf{Z})\, (\mathsf{Cons}\, m\, (\mathsf{from}\, (\mathsf{S}\, m)))$
$\to_{\mathsf{H}}\ \mathsf{Cons}\, m (\mathsf{take}\, \mathsf{Z}\, (\mathsf{from}\, (\mathsf{S}\, m)))$
$\to_{\mathsf{H}}\ \mathsf{Cons}\, m\, \mathsf{Nil}$

# Termination of HASKELL-Programs

$\mathsf{from}\, x = \mathsf{Cons}\, x\, (\mathsf{from}\, (\mathsf{S}\, x))$

$\mathsf{take}\, \mathsf{Z}\, xs = \mathsf{Nil}$
$\mathsf{take}\, n\, \mathsf{Nil} = \mathsf{Nil}$
$\mathsf{take}\, (\mathsf{S}\, n)\, (\mathsf{Cons}\, x\, xs) = \mathsf{Cons}\, x\, (\mathsf{take}\, n\, xs)$

- **Goal:** Prove termination of *start term* "$\mathsf{take}\, u\, (\mathsf{from}\, m)$"
- **Naive approach:**
  - take defining equations of take and from as TRS
  - fails, since from is not terminating
  - disregards HASKELL's lazy evaluation strategy
- **Our approach:**
  - evaluate start term a few steps ⇒ termination graph
  - do not transform HASKELL into TRS directly, but transform termination graph into TRS

# Termination of HASKELL-Programs

$\mathsf{from}\, x = \mathsf{Cons}\, x\, (\mathsf{from}\, (\mathsf{S}\, x))$

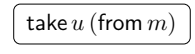
$\mathsf{take}\, \mathsf{Z}\, xs = \mathsf{Nil}$
$\mathsf{take}\, n\, \mathsf{Nil} = \mathsf{Nil}$
$\mathsf{take}\, (\mathsf{S}\, n)\, (\mathsf{Cons}\, x\, xs) = \mathsf{Cons}\, x\, (\mathsf{take}\, n\, xs)$

- **Evaluation Relation** $\to_{\mathsf{H}}$
- **H-Termination** of ground term $t$ if
  - $t$ does not start infinite evaluation $t \to_{\mathsf{H}} \ldots$
  - if $t \to_{\mathsf{H}}^{*} (f\ t_1 \ldots t_n)$, $f$ defined, $n < \mathrm{arity}(f)$, then $(f\ t_1 \ldots t_n\, t')$ is also H-terminating if $t'$ is H-terminating
  - if $t \to_{\mathsf{H}}^{*} (c\ t_1 \ldots t_n)$, $c$ constructor, then $t_1, \ldots, t_n$ are also H-terminating.
- **H-Termination** of arbitrary term $t$ if $t\sigma$ H-terminates for all substitutions $\sigma$ with H-terminating terms.
- "from" not H-terminating ("from Z" has infinite evaluation)
  "take $u$ (from $m$)" is H-terminating

# From HASKELL to Termination Graphs

$\mathsf{from}\, x = \mathsf{Cons}\, x\, (\mathsf{from}\, (\mathsf{S}\, x))$

$\mathsf{take}\, \mathsf{Z}\, xs = \mathsf{Nil}$
$\mathsf{take}\, n\, \mathsf{Nil} = \mathsf{Nil}$
$\mathsf{take}\, (\mathsf{S}\, n)\, (\mathsf{Cons}\, x\, xs) = \mathsf{Cons}\, x\, (\mathsf{take}\, n\, xs)$

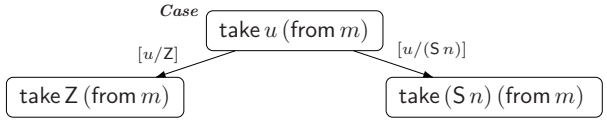



- begin with node marked with start term
- expansion rules to add children to leaves

# From HASKELL to Termination Graphs

$\mathsf{from}\ x = \mathsf{Cons}\ x\ (\mathsf{from}\ (\mathsf{S}\ x))$

$\mathsf{take}\ \mathsf{Z}\ xs = \mathsf{Nil}$
$\mathsf{take}\ n\ \mathsf{Nil} = \mathsf{Nil}$
$\mathsf{take}\ (\mathsf{S}\ n)\ (\mathsf{Cons}\ x\ xs) = \mathsf{Cons}\ x\ (\mathsf{take}\ n\ xs)$

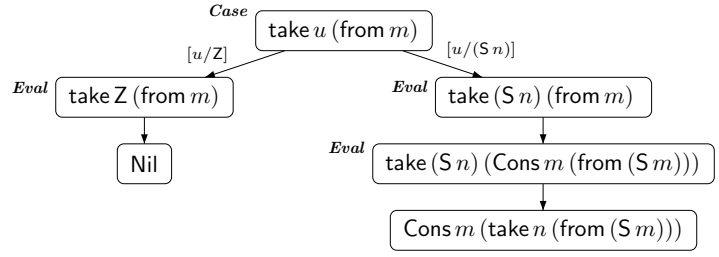



- ***Case*** rule:
  - evaluation has to continue with variable $u$
  - instantiate $u$ by all possible constructor terms of correct type

# From HASKELL to Termination Graphs

$\mathsf{from}\ x = \mathsf{Cons}\ x\ (\mathsf{from}\ (\mathsf{S}\ x))$

$\mathsf{take}\ \mathsf{Z}\ xs = \mathsf{Nil}$
$\mathsf{take}\ n\ \mathsf{Nil} = \mathsf{Nil}$
$\mathsf{take}\ (\mathsf{S}\ n)\ (\mathsf{Cons}\ x\ xs) = \mathsf{Cons}\ x\ (\mathsf{take}\ n\ xs)$



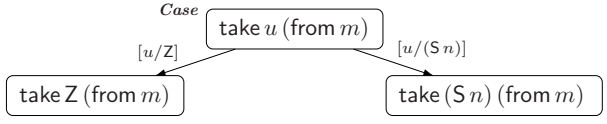

- ***Eval*** rule:

  performs one evaluation step with $\rightarrow_{\mathsf{H}}$

# From HASKELL to Termination Graphs

$\mathsf{from}\ x = \mathsf{Cons}\ x\ (\mathsf{from}\ (\mathsf{S}\ x))$

$\mathsf{take}\ \mathsf{Z}\ xs = \mathsf{Nil}$
$\mathsf{take}\ n\ \mathsf{Nil} = \mathsf{Nil}$
$\mathsf{take}\ (\mathsf{S}\ n)\ (\mathsf{Cons}\ x\ xs) = \mathsf{Cons}\ x\ (\mathsf{take}\ n\ xs)$



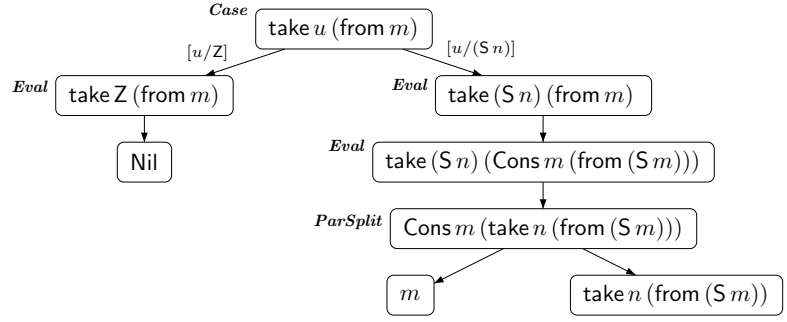

- **Main Property of Termination Graphs**:

  A node is H-terminating if all its children are H-terminating.

# From HASKELL to Termination Graphs

$\mathsf{from}\ x = \mathsf{Cons}\ x\ (\mathsf{from}\ (\mathsf{S}\ x))$

$\mathsf{take}\ \mathsf{Z}\ xs = \mathsf{Nil}$
$\mathsf{take}\ n\ \mathsf{Nil} = \mathsf{Nil}$
$\mathsf{take}\ (\mathsf{S}\ n)\ (\mathsf{Cons}\ x\ xs) = \mathsf{Cons}\ x\ (\mathsf{take}\ n\ xs)$



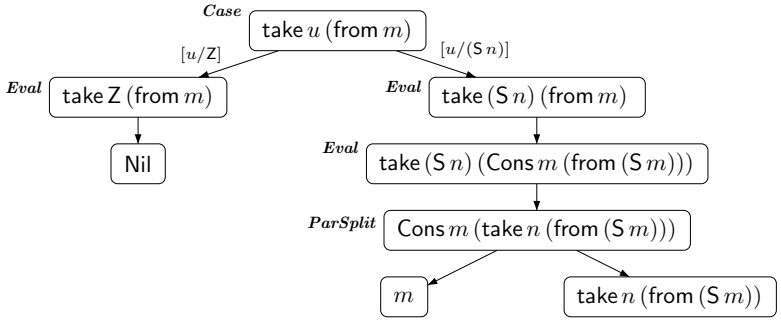

- ***ParSplit*** rule:

  if head of term is a constructor like Cons,
  then continue with the parameters

# From HASKELL to Termination Graphs

from $x$ = Cons $x$ (from (S $x$))

take Z $xs$ = Nil
take $n$ Nil = Nil
take (S $n$) (Cons $x$ $xs$) = Cons $x$ (take $n$ $xs$)

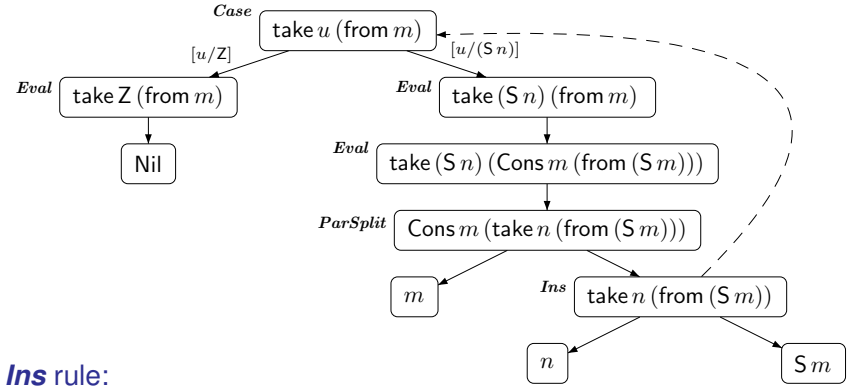



- one could continue with ***Case***, ***Eval***, ***ParSplit***
  $\Rightarrow$ infinite tree
- Instead: ***Ins*** rule to obtain finite graphs

# From HASKELL to Termination Graphs

from $x$ = Cons $x$ (from (S $x$))

take Z $xs$ = Nil
take $n$ Nil = Nil
take (S $n$) (Cons $x$ $xs$) = Cons $x$ (take $n$ $xs$)

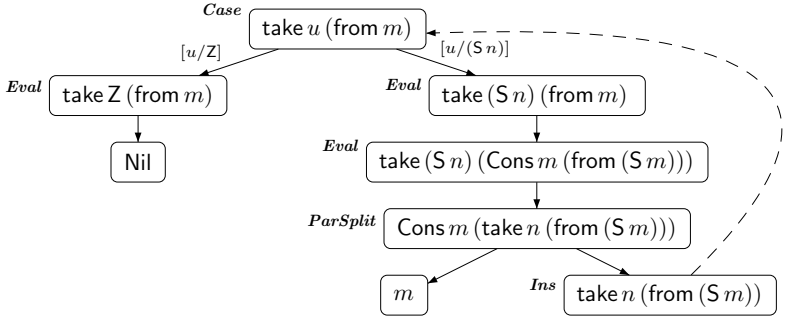



- ***Ins*** rule:
  - if leaf $t$ is instance of $t'$, then add instantiation edge from $t$ to $t'$
  - since instantiation is $[u/n,\ m/(\mathsf{S}\ m)]$, add child nodes $n$ and $(\mathsf{S}\ m)$

# From HASKELL to Termination Graphs

from $x$ = Cons $x$ (from (S $x$))

take Z $xs$ = Nil
take $n$ Nil = Nil
take (S $n$) (Cons $x$ $xs$) = Cons $x$ (take $n$ $xs$)

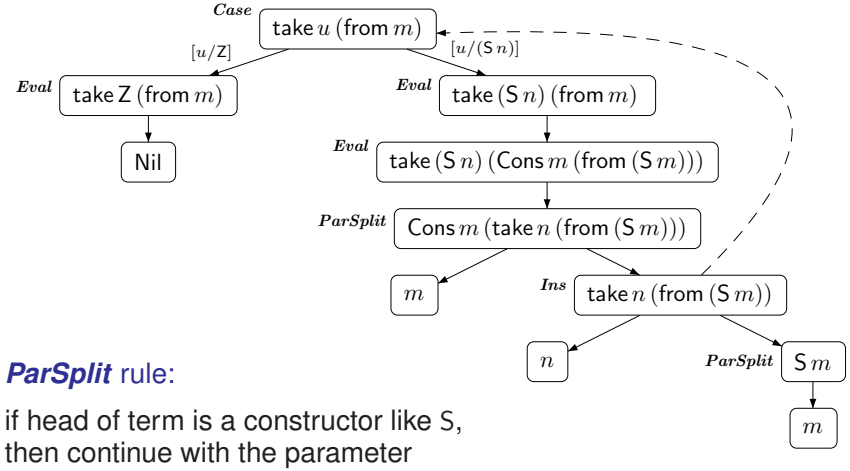



- ***Ins*** rule:
  - if leaf $t$ is instance of $t'$, then add instantiation edge from $t$ to $t'$
  - one may re-use an existing node for $t'$, if possible

# From HASKELL to Termination Graphs

from $x$ = Cons $x$ (from (S $x$))

take Z $xs$ = Nil
take $n$ Nil = Nil
take (S $n$) (Cons $x$ $xs$) = Cons $x$ (take $n$ $xs$)



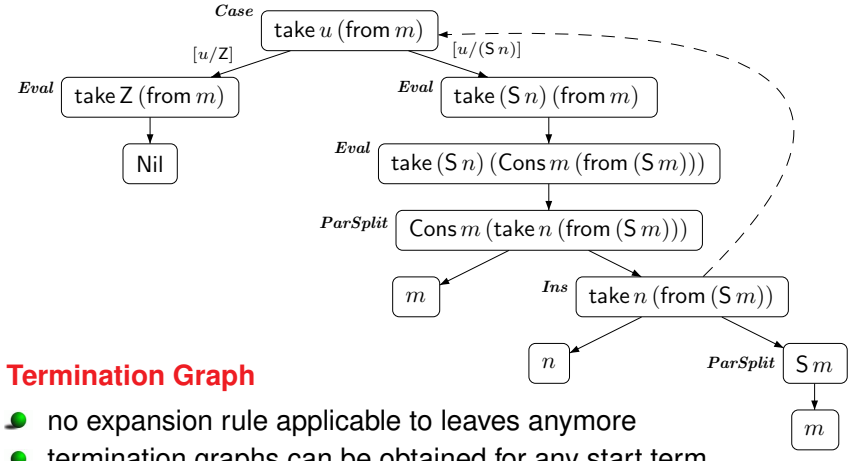

- ***ParSplit*** rule:

  if head of term is a constructor like S,
  then continue with the parameter

# From HASKELL to Termination Graphs

$\mathsf{from}\ x = \mathsf{Cons}\ x\ (\mathsf{from}\ (\mathsf{S}\ x))$

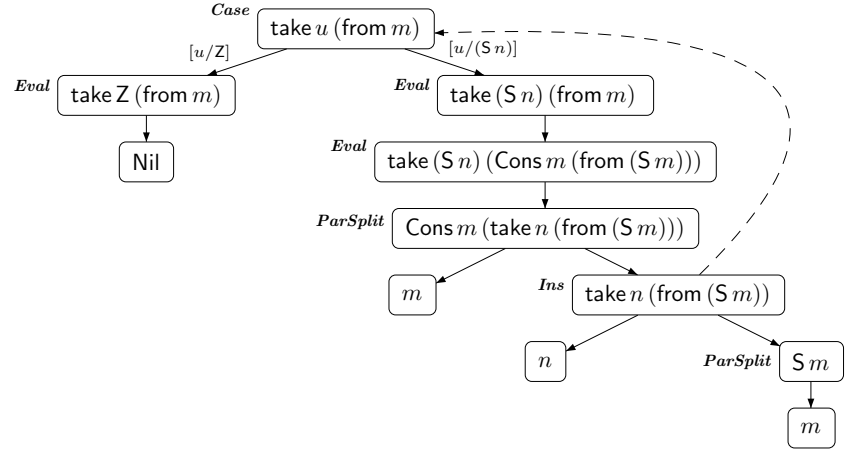
$\mathsf{take}\ \mathsf{Z}\ xs = \mathsf{Nil}$
$\mathsf{take}\ n\ \mathsf{Nil} = \mathsf{Nil}$
$\mathsf{take}\ (\mathsf{S}\ n)\ (\mathsf{Cons}\ x\ xs) = \mathsf{Cons}\ x\ (\mathsf{take}\ n\ xs)$




- **Termination Graph**
  - no expansion rule applicable to leaves anymore
  - termination graphs can be obtained for any start term

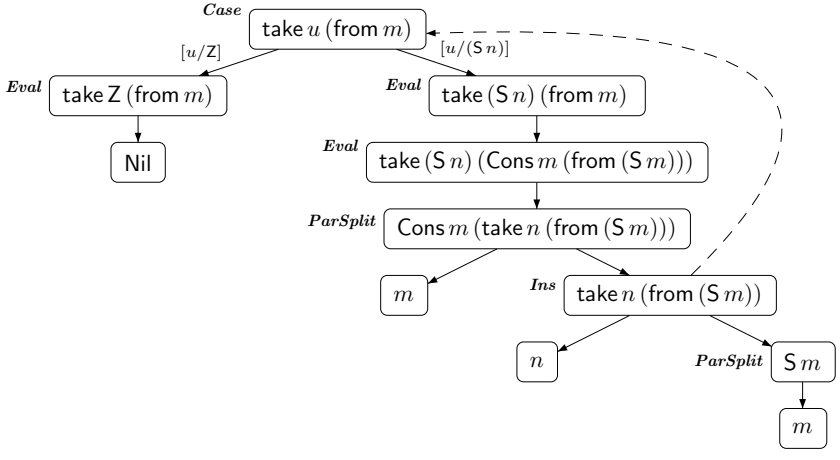
# From Termination Graphs to DP Problems

- **Goal:** Prove H-termination of all terms in termination graph



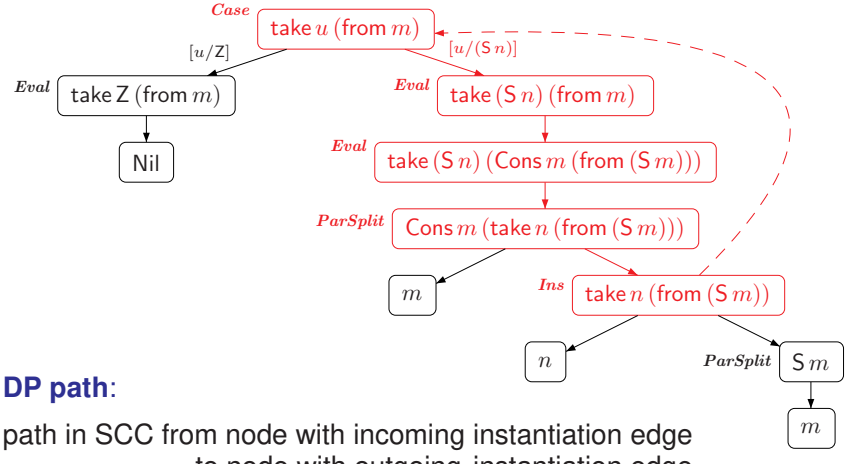

# From Termination Graphs to DP Problems

- higher-order terms can be represented as applicative first-order terms:
  "$x\ y$" becomes "$\mathsf{ap}(x, y)$"




# From Termination Graphs to DP Problems

- **Goal:** Prove H-termination of all terms for each SCC

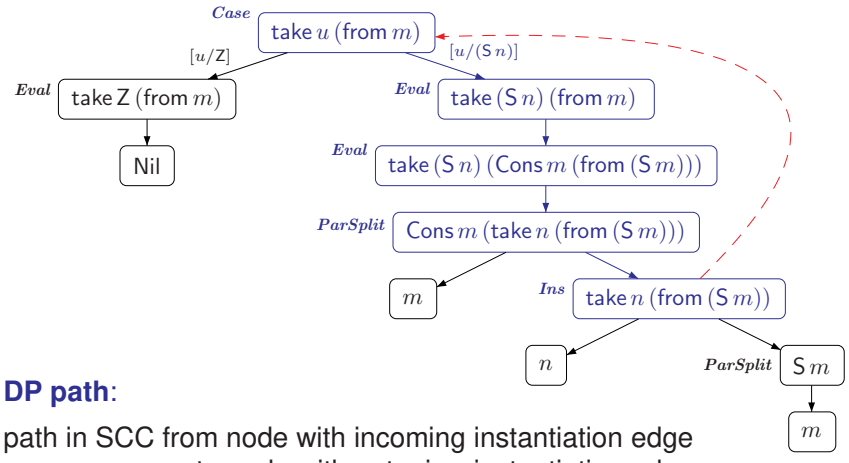



- **DP path:**
  path in SCC from node with incoming instantiation edge
  to node with outgoing instantiation edge

# From Termination Graphs to DP Problems

- every infinite path traverses a DP path infinitely often

  $\Rightarrow$ generate a dependency pair for every DP path



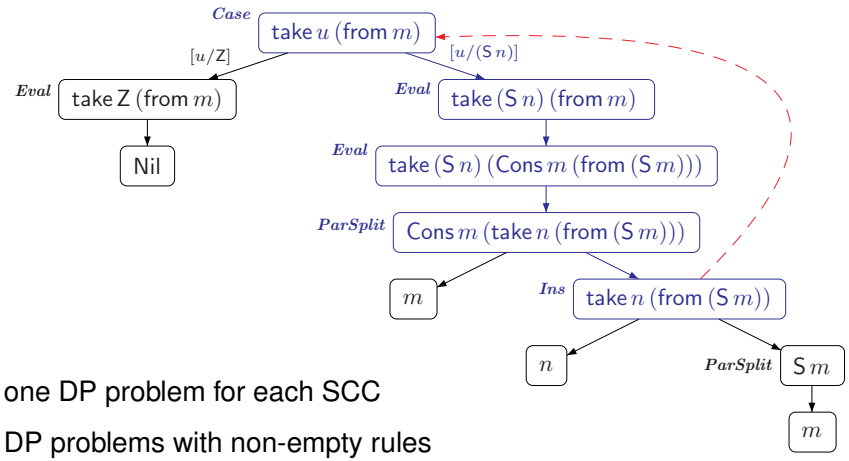

- **DP path**:

  path in SCC from node with incoming instantiation edge
  to node with outgoing instantiation edge

# From Termination Graphs to DP Problems

- **DP Problem**: $(\{\ \mathsf{take}\ (\mathsf{S}\ n)\ (\mathsf{from}\ m)\ \rightarrow\ \mathsf{take}\ n\ (\mathsf{from}\ (\mathsf{S}\ m))\ \},\ \varnothing)$

  $\Rightarrow$ termination is easy to prove

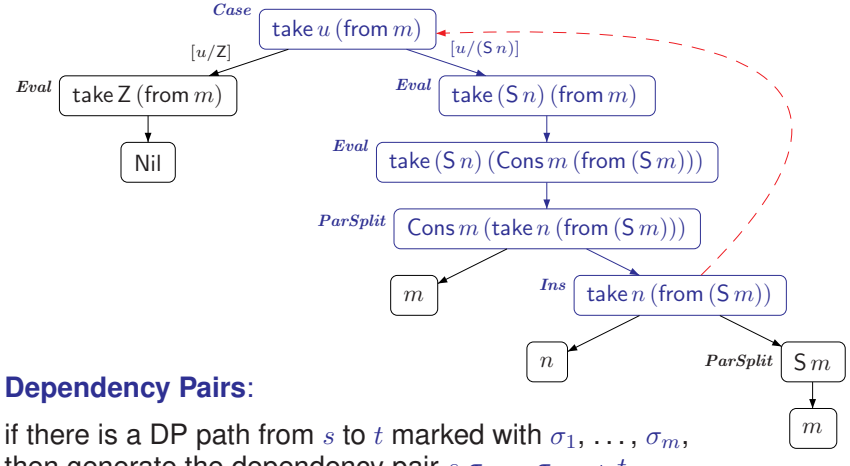



- one DP problem for each SCC
- DP problems with non-empty rules
  if defined symbols in rhs have to be evaluated

# From Termination Graphs to DP Problems

- **DP Problem**: $(\{\ \mathsf{take}\ (\mathsf{S}\ n)\ (\mathsf{from}\ m)\ \rightarrow\ \mathsf{take}\ n\ (\mathsf{from}\ (\mathsf{S}\ m))\ \},\ \varnothing)$

  $\Rightarrow$ termination is easy to prove




- **Dependency Pairs**:

  if there is a DP path from $s$ to $t$ marked with $\sigma_1, \ldots, \sigma_m$,
  then generate the dependency pair $s\ \sigma_1 \ldots \sigma_m \rightarrow t$

# Termination of HASKELL-Programs

- **New approach in order to use TRS-techniques for HASKELL**
  - generate termination graph for given start term
  - extract DP problems from termination graph
  - prove finiteness of DP problems by existing TRS-techniques
- **Implemented in AProVE**
  - accepts full HASKELL 98 language
  - successfully evaluated with standard HASKELL-libraries
- **Future work**

  TRS-techniques for termination of other languages (e.g., JAVA)