# Automated Termination Analysis

Jürgen Giesl

LuFG Informatik 2, RWTH Aachen University, Germany

VTSA '12, Saarbrücken, Germany

# Overview

## I. Termination of Term Rewriting

1. Termination of Term Rewrite Systems
2. Non-Termination of Term Rewrite Systems
3. Complexity of Term Rewrite Systems
4. Termination of Integer Term Rewrite Systems

## II. Termination of Programs

1. Termination of Functional Programs (Haskell)
2. Termination of Logic Programs (Prolog)
3. Termination of Imperative Programs (Java)

# Termination Analysis for TRSs

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

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

**Computation of "2 + 1":**

$$\begin{array}{rcl} \mathsf{plus}(\mathsf{s}(\mathsf{s}(\mathsf{0})), \mathsf{s}(\mathsf{0})) & \to_{\mathcal{R}} & \mathsf{s}(\mathsf{plus}(\mathsf{s}(\mathsf{s}(\mathsf{0})), \mathsf{0})) \\ & \to_{\mathcal{R}} & \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{0}))) \end{array}$$

# Termination Analysis for TRSs

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

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

- 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

# 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
- Integrating Other Termination Techniques

## 3. Other Recent Techniques

- Semantic Labeling *(Zantema, 95)*
- Match-Bounds *(Geser, Hofbauer, Waldmann, Zantema, 03 – today)*

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

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

LPO does lexicographic comparison from left to right:

- $s = f(s_1 ... s_{j-1} s_j ... s_n)$, $t = f(s_1 ... s_{j-1} t_j ... t_n)$, $s_j \succ t_j$, $s \succ t_i$ 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:

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

# RPO with Status (RPOS)

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

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

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

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

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

$\mathcal{R}$ 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 $l \to r \in \mathcal{R}$

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

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

# 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

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

- 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

$$
\begin{array}{rcl}
\mathsf{minus}(x, 0) & \rightarrow & x \\
\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) & \rightarrow & \mathsf{minus}(x, y) \\
\mathsf{div}(0, \mathsf{s}(y)) & \rightarrow & 0 \\
\mathsf{div}(\mathsf{s}(x), \mathsf{s}(y)) & \rightarrow & \mathsf{s}(\mathsf{div}(\mathsf{minus}(x, y), \mathsf{s}(y)))
\end{array}
$$

- Defined Symbols: minus, div
  Constructors: 0, s

- **Definition**
  If $f(s_1, \ldots, s_n) \rightarrow C[g(t_1, \ldots, t_m)]$ is a rule and $g$ is defined, then $\mathsf{F}(s_1, \ldots, s_n) \rightarrow \mathsf{G}(t_1, \ldots, t_m)$ is a *dependency pair*

$$
\begin{array}{rcl}
\mathsf{M}(\mathsf{s}(x), \mathsf{s}(y)) & \rightarrow & \mathsf{M}(x, y) \\
\mathsf{D}(\mathsf{s}(x), \mathsf{s}(y)) & \rightarrow & \mathsf{M}(x, y) \\
\mathsf{D}(\mathsf{s}(x), \mathsf{s}(y)) & \rightarrow & \mathsf{D}(\mathsf{minus}(x, y), \mathsf{s}(y))
\end{array}
$$

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

- **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

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

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

$$\mathsf{D}(\mathsf{s}(\mathsf{s}(0)),\mathsf{s}(0)) \to \mathsf{D}(\mathsf{minus}(\mathsf{s}(0),0),\mathsf{s}(0)) \qquad \mathsf{D}(\mathsf{s}(0),\mathsf{s}(0)) \to \mathsf{D}(\mathsf{minus}(0,0),\mathsf{s}(0))$$

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

- 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 there is no infinite $(DP(\mathcal{R}), \mathcal{R})$-chain.

# 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.

# 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 from Other Termination Techniques

# I. DP Processors for Proving Termination

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

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

- **Rule Removal Processor**

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

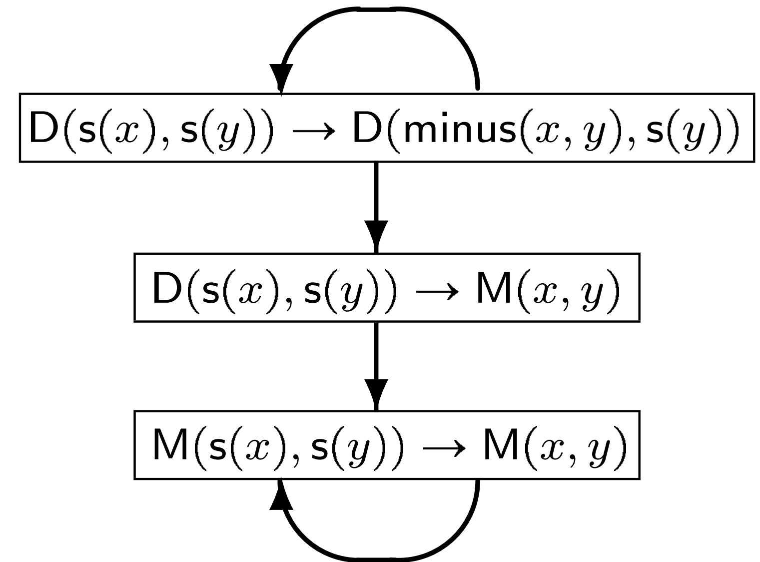
$\mathcal{P}$: $\mathsf{M}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{M}(x, y)$
$\mathsf{D}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{M}(x, y)$
$\mathsf{D}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{D}(\mathsf{minus}(x, y), \mathsf{s}(y))$

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

- **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

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

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

$\mathcal{R}:$

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

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

$$s_1 \to t_1, \quad s_2 \to t_2, \quad s_3 \to t_3, \ldots$$

$$s_1\sigma \;{}_{(}\!\succsim\!{}_{)}\; t_1\sigma \succsim s_2\sigma \;{}_{(}\!\succsim\!{}_{)}\; t_2\sigma \succsim s_3\sigma \;{}_{(}\!\succsim\!{}_{)}\; t_3\sigma \succsim \ldots$$

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

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

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

- **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 $(\mathcal{P}_1, \mathcal{R})$:

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

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

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

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

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

$\mathcal{R}$ :

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

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

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

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

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

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

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

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

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

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

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

$(DP(\mathcal{R}), \mathcal{R})$

Dep. Graph

$(\mathcal{P}_1, \mathcal{R})$ — Red. Pair — $(\varnothing, \mathcal{R})$ — Dep. Graph — $\varnothing$

$(\mathcal{P}_2, \mathcal{R})$ — Red. Pair — $(\varnothing, \mathcal{R})$ — Dep. Graph — $\varnothing$

Termination is proved automatically!

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

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

DP problem for minus hard to solve automatically!

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

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

- **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 $\mathcal{R}$ — $\mathcal{R}_{\succ} \cup \mathcal{R}_{\succsim} = \mathcal{R}$
  - $s \succ t$ or $s \succsim t$ for all $s \to t$ in $\mathcal{P}$ — $\mathcal{P}_{\succ} \cup \mathcal{P}_{\succsim} = \mathcal{P}$
  - $\succ$ 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 $\mathcal{P}ol(f(t_1,\ldots,t_n)) = 1 + \mathcal{P}ol(t_1) + \ldots + \mathcal{P}ol(t_n)$

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

$$\mathsf{minus}(x, 0) \rightarrow x$$
$$\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) \rightarrow \mathsf{minus}(\mathsf{p}(\mathsf{s}(x)), \mathsf{p}(\mathsf{s}(y)))$$

- **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 $l \rightarrow r$ in $\mathcal{R}$ — $\mathcal{R}_{\succ} \cup \mathcal{R}_{\succsim} = \mathcal{R}$
  - $s \succ t$ or $s \succsim t$ for all $s \rightarrow t$ in $\mathcal{P}$ — $\mathcal{P}_{\succ} \cup \mathcal{P}_{\succsim} = \mathcal{P}$
  - $\succ$ 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

$$
\begin{array}{rcl}
\mathsf{f}(\mathsf{g}(x), \mathsf{s}(0), y) & \rightarrow & \mathsf{f}(y, y, \mathsf{g}(x)) \\
\mathsf{g}(\mathsf{s}(x)) & \rightarrow & \mathsf{s}(\mathsf{g}(x)) \\
\mathsf{g}(0) & \rightarrow & 0
\end{array}
$$

- Infinite (non-innermost) reduction:

$$
\begin{array}{rl}
\mathsf{f}(\mathrm{gs0}, \mathrm{s0}, \mathrm{gs0}) \rightarrow & \mathsf{f}(\mathrm{gs0}, \mathrm{gs0}, \mathrm{gs0}) \\
\rightarrow & \mathsf{f}(\mathrm{gs0}, \mathrm{sg0}, \mathrm{gs0}) \\
\rightarrow & \mathsf{f}(\mathrm{gs0}, \mathrm{s0}, \mathrm{gs0}) \rightarrow \ldots
\end{array}
$$

# 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.

$\mathcal{P}$ :

$$\mathsf{G}(\mathsf{s}(x)) \rightarrow \mathsf{G}(x)$$

$\mathcal{R}$ :

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

- **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

$\mathsf{F}(\mathsf{g}(x), \mathsf{s}(0), y) \rightarrow \mathsf{F}(y, y, \mathsf{g}(x))$

$\mathsf{F}(\mathsf{g}(x), \mathsf{s}(0), y) \rightarrow \mathsf{G}(x)$

$\mathsf{G}(\mathsf{s}(x)) \rightarrow \mathsf{G}(x)$

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

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

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

- **Usable Rule Processor** (sound)

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

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

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

- **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 $e$

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

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

- **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

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

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

- **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 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*:

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

# III. 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 $\mathcal{R}' = TT(\mathcal{P} \cup \mathcal{R})$

# III. 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 $\mathcal{R}' = TT(\mathcal{P} \cup \mathcal{R})$

# Challenging Example

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

- 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

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

$\mathcal{R}$ :

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

- **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

$\mathcal{P}:\quad \mathsf{FACT}(\mathsf{s}(x)) \;\rightarrow\; \mathsf{FACT}(\mathsf{p}(\mathsf{s}(x)))$

$\mathcal{U}(\mathcal{P},\mathcal{R}):$

$$\begin{aligned} \mathsf{p}(\mathsf{s}(0)) &\rightarrow 0 \\ \mathsf{p}(\mathsf{s}(\mathsf{s}(x))) &\rightarrow \mathsf{s}(\mathsf{p}(\mathsf{s}(x))) \end{aligned}$$

- **Usable Rule Processor** (sound)

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

- $\mathcal{U}(\mathcal{P},\mathcal{R}) \;=\; \{\;\mathsf{p}(\mathsf{s}(0)) \rightarrow 0,\quad \mathsf{p}(\mathsf{s}(\mathsf{s}(x))) \rightarrow \mathsf{s}(\mathsf{p}(\mathsf{s}(x)))\;\}$

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

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

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

- **Automation** — count number of $\mathsf{s}$-symbols

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

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

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

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

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

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

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

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

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

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

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

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

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

$\mathcal{P}:$

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

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

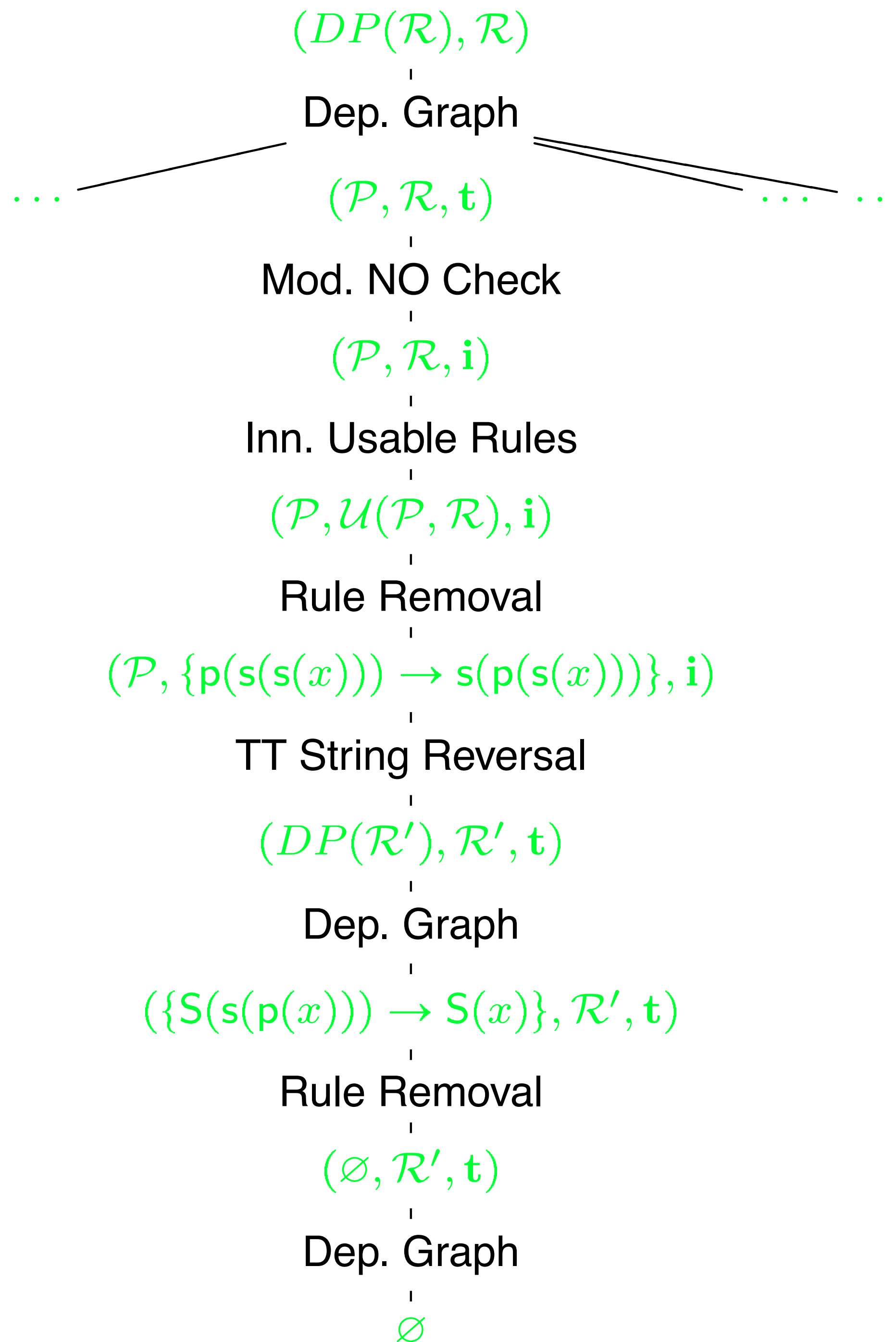
$\mathsf{s}(\mathsf{s}(\mathsf{p}(x))) \to \mathsf{s}(\mathsf{p}(\mathsf{s}(x)))$

- **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

Termination is proved automatically!



# Semantic Labeling

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

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

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

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 Labeling

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

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

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

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

$$\begin{array}{lcl} \mathsf{f}_0(\mathsf{f}_0(x)) & \rightarrow & \mathsf{f}_1(\mathsf{g}_0(\mathsf{f}_0(x))) \\ \mathsf{f}_0(\mathsf{f}_1(x)) & \rightarrow & \mathsf{f}_1(\mathsf{g}_0(\mathsf{f}_1(x))) \end{array}$$

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

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

# Semantic Labeling

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

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

# Semantic Labeling

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

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

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

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

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

# Match–Bounds for String Rewriting

$$\mathcal{R}: \qquad \mathsf{a}(\mathsf{b}(\mathsf{a}(x))) \;\rightarrow\; \mathsf{a}(\mathsf{b}(\mathsf{b}(\mathsf{b}(\mathsf{a}(x)))))$$

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

$$\mathsf{a}_0(\mathsf{b}_0(\mathsf{a}_0(x))) \rightarrow \mathsf{a}_1(\mathsf{b}_1(\mathsf{b}_1(\mathsf{b}_1(\mathsf{a}_1(x)))))$$
$$\mathsf{a}_0(\mathsf{b}_1(\mathsf{a}_5(x))) \rightarrow \mathsf{a}_1(\mathsf{b}_1(\mathsf{b}_1(\mathsf{b}_1(\mathsf{a}_1(x)))))$$
$$\mathsf{a}_3(\mathsf{b}_1(\mathsf{a}_5(x))) \rightarrow \mathsf{a}_2(\mathsf{b}_2(\mathsf{b}_2(\mathsf{b}_2(\mathsf{a}_2(x)))))$$
$$\vdots$$

# Match–Bounds for String Rewriting

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

- **Thm 1:** If all symbols in $t$ are labeled with 0 and for all $t \rightarrow^*_{\mathbf{match}(\mathcal{R})} s$, the labels in $s$ are $\leqslant n$ (*match-bound*), then $t$ is terminating w.r.t. $\mathcal{R}$.
- **Right-Forward Closures** (Dershowitz, 81):

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

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

# Match–Bounds for String Rewriting

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

- **Thm 1:** If all symbols in $t$ are labeled 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 labeled 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.

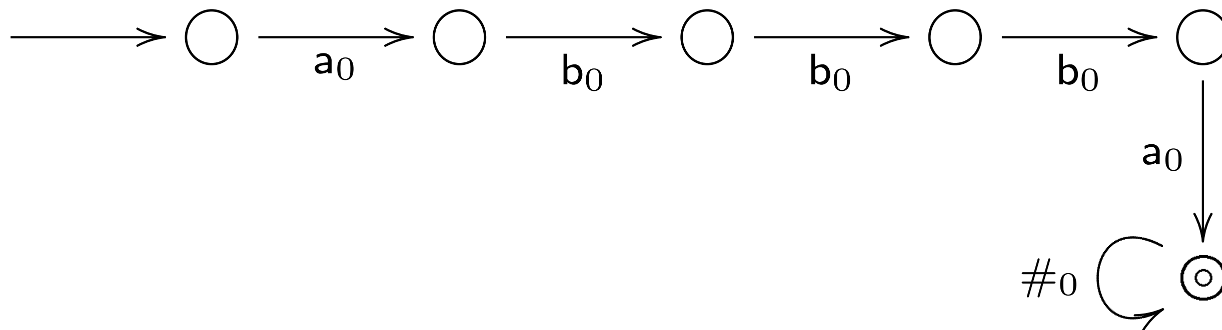
**Construct finite automaton accepting** $\{s \mid r\ \#^k \to^*_{\mathbf{match}(\mathcal{R}_\#)} s\}$

$\mathcal{R}_\#$ :

a(b(a($x$))) $\to$ a(b(b(b(a($x$)))))
a(b(#($x$))) $\to$ a(b(b(b(a($x$)))))
a(#($x$)) $\to$ a(b(b(b(a($x$)))))

# 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 $a\ w$.

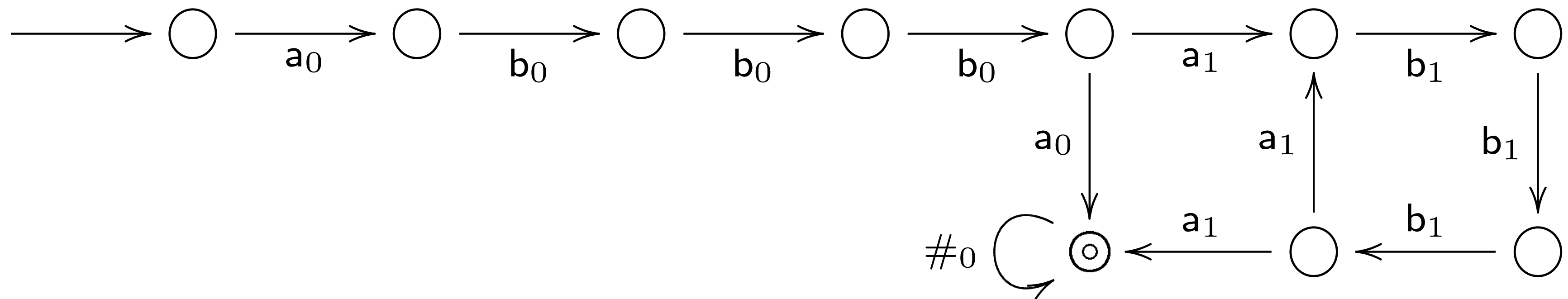



**Construct finite automaton accepting** $\{s \mid r\ \#^k \to^*_{\mathbf{match}(\mathcal{R}_\#)} s\}$

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

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

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

# 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

# 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 Labeling

# 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

- Improvements
  - 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. Termination Competition '04 - '12*
    both for *termination* and *non-termination* of TRSs

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

# Overview

## I. Termination of Term Rewriting

1. Termination of Term Rewrite Systems
2. Non-Termination of Term Rewrite Systems (FroCoS '05, IJCAR '12)
3. Complexity of Term Rewrite Systems
4. Termination of Integer Term Rewrite Systems

## II. Termination of Programs

1. Termination of Functional Programs (Haskell)
2. Termination of Logic Programs (Prolog)
3. Termination of Imperative Programs (Java)

# DP Processors for Disproving Termination

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

$\mathcal{R}$ 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 $s$

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

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

## 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 $s_k\sigma, \quad k > 1$ $\qquad (s_1\sigma\ \mu = s_k\sigma)$

## 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 $s$

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

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

## 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 $s_k\sigma, \quad k > 1$ $\qquad (s_1\sigma\, \mu = s_k\sigma)$

## 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

$$
\begin{aligned}
\mathcal{P}: \quad \mathsf{M}(\mathsf{s}(x), \mathsf{s}(y)) &\rightarrow \mathsf{M}(x, y) \\
\mathsf{D}(\mathsf{s}(x), y) &\rightarrow \mathsf{M}(\mathsf{s}(x), y) \\
\mathsf{D}(\mathsf{s}(x), y) &\rightarrow \mathsf{D}(\mathsf{minus}(\mathsf{s}(x), y), y)
\end{aligned}
$$

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

## Definition (Looping DP Problem)

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

- $t_1\sigma \rightarrow^*_{\mathcal{R}} s_2\sigma, \quad t_2\sigma \rightarrow^*_{\mathcal{R}} s_3\sigma, \quad \ldots$
- $s_1\sigma$ 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”** $\qquad$ if $(\mathcal{P}, \mathcal{R})$ is looping

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

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

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

$\mathcal{R}: \quad \mathsf{minus}(x, 0) \to x$

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

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

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

# 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

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

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

Dependency Graph Processor results in

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

  easy to prove

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

  potentially infinite

Termination techniques help to identify non-terminating parts!

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

$\mathcal{R}:$

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

## 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 $s\sigma$ matches $s'$

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

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

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

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

## Detect loops by narrowing:

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

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

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

$\mathcal{P}: \mathsf{F}(0,1,x) \to \mathsf{F}(x,x,x)$

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

## 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

$\mathsf{F}(0,1,x) \to \mathsf{F}(x,x,x)$ cannot be narrowed!

$\mathcal{P}: \mathsf{F}(0,1,x) \to \mathsf{F}(x,x,x)$

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

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

$\mathsf{F}(\mathsf{g}(0,z),\mathsf{g}(y,1),x)$ 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: $t' \leftsquigarrow^*_\sigma s \to t$
- check whether $t'$ and $t\sigma$ semi-unify

## 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

# Looping vs. Non-Looping Non-Termination

- Most existing approaches detect loops

$$s \to_{\mathcal{R}}^{n} C[\, s\,\mu \,] \to_{\mathcal{R}}^{n} C[\, C\mu[\, s\,\mu^2 \,]\,] \to_{\mathcal{R}}^{n} \ldots$$

- cannot capture non-periodic infinite rewrite sequences

$$\begin{aligned}
\mathsf{f}(\mathsf{tt}, x, y) &\to \mathsf{f}(\mathsf{gt}(x,y), \mathsf{dbl}(x), \mathsf{s}(y)) & \mathsf{dbl}(x) &\to \mathsf{mul}(\mathsf{s}^2(0), x)\\
\mathsf{gt}(\mathsf{s}(x), 0) &\to \mathsf{tt} & \mathsf{mul}(x, 0) &\to 0\\
\mathsf{gt}(0, y) &\to \mathsf{ff} & \mathsf{mul}(x, \mathsf{s}(y)) &\to \mathsf{plus}(\mathsf{mul}(x,y), x)\\
\mathsf{gt}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{gt}(x, y) & \mathsf{plus}(x, 0) &\to x\\
& & \mathsf{plus}(x, \mathsf{s}(y)) &\to \mathsf{plus}(\mathsf{s}(x), y)
\end{aligned}$$

$$\begin{aligned}
&\mathsf{f}(\mathsf{tt}, \mathsf{s}^n(0), \mathsf{s}^m(0)) && \to_{\mathcal{R}}\\
&\mathsf{f}(\mathsf{gt}(\mathsf{s}^n(0), \mathsf{s}^m(0)), \mathsf{dbl}(\mathsf{s}^n(0)), \mathsf{s}^{m+1}(0)) && \to_{\mathcal{R}}^{m+1}\\
&\mathsf{f}(\mathsf{tt}, \mathsf{dbl}(\mathsf{s}^n(0)), \mathsf{s}^{m+1}(0)) && \to_{\mathcal{R}}\\
&\mathsf{f}(\mathsf{tt}, \mathsf{mul}(\mathsf{s}^2(0), \mathsf{s}^n(0)), \mathsf{s}^{m+1}(0)) && \to_{\mathcal{R}}^{4\cdot n}\\
&\mathsf{f}(\mathsf{tt}, \mathsf{s}^{2\cdot n}(\mathsf{mul}(\mathsf{s}^2(0), 0)), \mathsf{s}^{m+1}(0)) && \to_{\mathcal{R}}\\
&\mathsf{f}(\mathsf{tt}, \mathsf{s}^{2\cdot n}(0), \mathsf{s}^{m+1}(0)) && \to_{\mathcal{R}} \ldots
\end{aligned}$$

```
while (gt(x,y)) {
  x = dbl(x);
  y = y + 1;     }
```

**non-terminating, but not looping**

# Looping vs. Non-Looping Non-Termination

- Most existing approaches detect loops

$$t \to_{\mathcal{R}}^{n} C[\, s\,\mu \,] \to_{\mathcal{R}}^{n} C[\, C\mu[\, s\,\mu^2 \,]\,] \to_{\mathcal{R}}^{n} \dots$$

## Method for Loop Detection

1. Let $\mathcal{S} := \mathcal{R}$.
2. Check if some $s \to t \in \mathcal{S}$ is a loop. If yes: stop with "non-termination".
3. Modify some $s \to t \in \mathcal{S}$ by narrowing to obtain $s' \to_{\mathcal{R}}^{+} t'$.
4. Let $\mathcal{S} := \mathcal{S} \cup \{s' \to t'\}$ and go to Step 2.

## Method for Non-Looping Non-Termination

1. Let $\mathcal{S}$ be a set of pattern rules $p \hookrightarrow q$ corresponding to $\mathcal{R}$.
2. Check if some $p \hookrightarrow q \in \mathcal{S}$ is obviously non-terminating.
   If yes: stop with "non-termination".
3. Modify some $p \hookrightarrow q \in \mathcal{S}$ by narrowing to obtain $p' \hookrightarrow q'$.
4. Let $\mathcal{S} := \mathcal{S} \cup \{p' \hookrightarrow q'\}$ and go to Step 2.

# Pattern Terms and Pattern Rules

## Pattern Term

*Pattern term* $p$ : $n \mapsto t\,\sigma^n\,\mu$

represents $\{\underbrace{t\,\mu}_{p(0)}, \underbrace{t\,\sigma\,\mu}_{p(1)}, \underbrace{t\,\sigma^2\,\mu}_{p(2)}, \underbrace{t\,\sigma^3\,\mu}_{p(3)}, \ldots\}$.

- base term $t$
- pumping substitution $\sigma$
- closing substitution $\mu$

# Pattern Terms and Pattern Rules

## Pattern Term

*Pattern term* $p$ : $t\,\sigma^n\,\mu$

represents $\{\underbrace{t\,\mu}_{p(0)}, \underbrace{t\,\sigma\,\mu}_{p(1)}, \underbrace{t\,\sigma^2\,\mu}_{p(2)}, \underbrace{t\,\sigma^3\,\mu}_{p(3)}, \ldots\}$.

- base term $t$
- pumping substitution $\sigma$
- closing substitution $\mu$

## Pattern Rule

*Pattern rule*: $p \hookrightarrow q$

where $p, q$ are pattern terms

$p \hookrightarrow q$ is *correct* w.r.t. TRS $\mathcal{R}$

if $\forall n \in \mathbb{N}: \; p(n) \to_{\mathcal{R}}^{+} q(n)$

**Example:** $p = \mathsf{gt}(\mathsf{s}(x), \mathsf{s}(y))\;\; [x/\mathsf{s}(x), y/\mathsf{s}(y)]^n\;\; [x/\mathsf{s}(x), y/0]$

represents $\{\underbrace{\mathsf{gt}(\mathsf{s}^2(x), \mathsf{s}(0))}_{p(0)}, \underbrace{\mathsf{gt}(\mathsf{s}^3(x), \mathsf{s}^2(0))}_{p(1)}, \underbrace{\mathsf{gt}(\mathsf{s}^4(x), \mathsf{s}^3(0))}_{p(2)}, \ldots\}$

**Example:** $\mathsf{gt}(\mathsf{s}(x), \mathsf{s}(y))\;\; [x/\mathsf{s}(x), y/\mathsf{s}(y)]^n\;\; [x/\mathsf{s}(x), y/0] \quad \hookrightarrow \quad \mathsf{tt}\;\; \varnothing^n\;\; \varnothing$

correct, since $\forall n \in \mathbb{N}: \;\; \mathsf{gt}(\mathsf{s}^{n+2}(x), \mathsf{s}^{n+1}(0)) \to_{\mathcal{R}}^{+} \mathsf{tt}$

# Proving Non-Termination of TRSs Automatically

**Method for Non-Looping Non-Termination**

1. Let $\mathcal{S}$ be a set of pattern rules corresponding to $\mathcal{R}$.
2. Check if some $p \hookrightarrow q \in \mathcal{S}$ is obviously non-terminating.
   If yes: stop with "non-termination".
3. Modify some $p \hookrightarrow q \in \mathcal{S}$ by narrowing to obtain $p' \hookrightarrow q'$.
4. Let $\mathcal{S} := \mathcal{S} \cup \{p' \hookrightarrow q'\}$ and go to Step 2.

## Contributions

- **pattern rules** $p \hookrightarrow q$
  to represent *sets* of rewrite sequences $\{p(n) \to_{\mathcal{R}}^{+} q(n) \mid n \in \mathbb{N}\}$
- **inference rules** to deduce new correct pattern rules

# Inference Rules

$$\frac{p_1 \hookrightarrow q_1 \quad \ldots \quad p_k \hookrightarrow q_k}{p \hookrightarrow q}$$

If $p_1 \hookrightarrow q_1, \ \ldots, \ p_k \hookrightarrow q_k$ are correct w.r.t. $\mathcal{R}$ then $p \hookrightarrow q$ is also correct w.r.t. $\mathcal{R}$

**(1) Pattern Rule from TRS**

$$\frac{}{\ell \, \varnothing^n \, \varnothing \;\hookrightarrow\; r \, \varnothing^n \, \varnothing} \quad \text{if } \ell \to r \in \mathcal{R}$$

$$\begin{aligned}
\mathsf{f}(\mathsf{tt}, x, y) &\to \mathsf{f}(\mathsf{gt}(x, y), \mathsf{dbl}(x), \mathsf{s}(y)) \\
\mathsf{gt}(\mathsf{s}(x), 0) &\to \mathsf{tt} \\
\mathsf{gt}(0, y) &\to \mathsf{ff} \\
\mathsf{gt}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{gt}(x, y) \\
\mathsf{dbl}(x) &\to \mathsf{mul}(\mathsf{s}^2(0), x) \\
\mathsf{mul}(x, 0) &\to 0 \\
\mathsf{mul}(x, \mathsf{s}(y)) &\to \mathsf{plus}(\mathsf{mul}(x, y), x) \\
\mathsf{plus}(x, 0) &\to x \\
\mathsf{plus}(x, \mathsf{s}(y)) &\to \mathsf{plus}(\mathsf{s}(x), y)
\end{aligned}$$

$$\mathsf{gt}(\mathsf{s}(x), \mathsf{s}(y)) \;\; \varnothing^n \;\; \varnothing \quad \hookrightarrow \quad \mathsf{gt}(x, y) \;\; \varnothing^n \;\; \varnothing$$

# Inference Rules

**(2) Pattern Creation**

$$\frac{s\,\varnothing^n\,\varnothing \;\hookrightarrow\; t\,\varnothing^n\,\varnothing}{s\,\sigma^n\,\varnothing \;\hookrightarrow\; t\,\theta^n\,\varnothing}$$

if $s\theta = t\sigma$, and
$\theta$ commutes with $\sigma$

$\theta$ and $\sigma$ *commute*
if $\theta\,\sigma = \sigma\,\theta$

$$\begin{array}{llllllll}
s\ \sigma^n & \to^+_{\mathcal{R}} & t\ \sigma^n & = & s\ \theta\,\sigma^{n-1} & = & & \\
s\ \sigma^{n-1}\,\theta & \to^+_{\mathcal{R}} & t\ \sigma^{n-1}\,\theta & = & s\ \theta\,\sigma^{n-2}\,\theta & = & & \\
s\ \sigma^{n-2}\,\theta^2 & \to^+_{\mathcal{R}} & t\ \sigma^{n-2}\,\theta^2 & = & s\ \theta\,\sigma^{n-3}\,\theta^2 & \to^+_{\mathcal{R}} \ldots \to^+_{\mathcal{R}} & t\ \theta^n &
\end{array}$$

# Inference Rules

**(2) Pattern Creation**

$$\frac{s\,\varnothing^n\,\varnothing \;\hookrightarrow\; t\,\varnothing^n\,\varnothing}{s\,\sigma^n\,\varnothing \;\hookrightarrow\; t\,\theta^n\,\varnothing}$$

if $s\theta = t\sigma$, and
$\theta$ commutes with $\sigma$

$\theta$ and $\sigma$ *commute*
if $\theta\,\sigma = \sigma\,\theta$

$$\underbrace{\mathsf{gt}(\mathsf{s}(x),\mathsf{s}(y))}_{s}\;\varnothing^n\;\varnothing \;\hookrightarrow\; \underbrace{\mathsf{gt}(x,y)}_{t}\;\varnothing^n\;\varnothing$$

$$\Downarrow$$

$$\mathsf{gt}(\mathsf{s}(x),\mathsf{s}(y))\;[x/\mathsf{s}(x), y/\mathsf{s}(y)]^n\;\varnothing \;\hookrightarrow\; \mathsf{gt}(x,y)\;\varnothing^n\;\varnothing$$

since $s\,\underbrace{\varnothing}_{\theta} = t\,\underbrace{[x/\mathsf{s}(x), y/\mathsf{s}(y)]}_{\sigma}$

# Inference Rules

**(3) Equivalence**

$$\frac{p \hookrightarrow q}{p' \hookrightarrow q'}$$ if $p$ is equivalent to $p'$ and $q$ is equivalent to $q'$

$p$ and $p'$ are *equivalent* if
$\forall n \in \mathbb{N}$: $p(n) = p'(n)$

**Goal:**

| | | | | |
|---|---|---|---|---|
| narrow | $\mathsf{gt}(\mathsf{s}(x), \mathsf{s}(y))\ [x/\mathsf{s}(x), y/\mathsf{s}(y)]^n\ \varnothing$ | $\hookrightarrow$ | $\mathsf{gt}(x, y)\ \varnothing^n\ \varnothing$ |
| with | $\mathsf{gt}(\mathsf{s}(x), 0)\ \varnothing^n\ \varnothing$ | $\hookrightarrow$ | $\mathsf{tt}\ \varnothing^n\ \varnothing$ |

**Problem:** rules have different pumping and closing substitutions

**Strategy:**

1. Instantiate base terms.
   (Base term of 1st rhs should *contain* base term of 2nd lhs.)
2. Make all 4 pumping substitutions equal.
3. Make all 4 closing substitutions equal.

# Inference Rules

## (3) Equivalence

$$\frac{p \hookrightarrow q}{p' \hookrightarrow q'}$$ if $p$ is equivalent to $p'$ and $q$ is equivalent to $q'$

**Criteria for Equivalence**

- renaming of *domain variables*

$$\begin{array}{rccc} \mathsf{gt}(\mathsf{s}(x),\mathsf{s}(y)) & [x/\mathsf{s}(x), y/\mathsf{s}(y)]^n & \varnothing & \hookrightarrow \\ \mathsf{gt}(x,y) & \varnothing^n & \varnothing & \end{array}$$

$$\begin{array}{rccc} \mathsf{gt}(\mathsf{s}(x'),\mathsf{s}(y')) & [x'/\mathsf{s}(x'),\ y'/\mathsf{s}(y')]^n & [x'/x,\ y'/y] & \hookrightarrow \\ \mathsf{gt}(x,y) & \varnothing^n & \varnothing & \end{array}$$

# Inference Rules

**(3) Equivalence**

$$\frac{p \hookrightarrow q}{p' \hookrightarrow q'}$$ if $p$ is equivalent to $p'$ and $q$ is equivalent to $q'$

**Criteria for Equivalence**

- renaming of *domain variables*
- modifying substitutions of *irrelevant variables*

$$\begin{array}{rlll} \mathsf{gt}(\mathsf{s}(x),\mathsf{s}(y)) & [x/\mathsf{s}(x), y/\mathsf{s}(y)]^n & \varnothing & \hookrightarrow \\ \mathsf{gt}(x,y) & \varnothing^n & \varnothing & \end{array}$$

$$\begin{array}{rlll} \mathsf{gt}(\mathsf{s}(x'),\mathsf{s}(y')) & [x'/\mathsf{s}(x'),\ y'/\mathsf{s}(y')]^n & [x'/x,\ y'/y] & \hookrightarrow \\ \mathsf{gt}(x,y) & [x'/\mathsf{s}(x'),\ y'/\mathsf{s}(y')]^n & [x'/x,\ y'/y] & \end{array}$$

# Inference Rules

**(4) Instantiation**

$$\frac{s\,\delta^n\,\tau \;\hookrightarrow\; t\,\sigma^n\,\mu}{(s\,\rho)\;\delta^n_\rho\;\tau_\rho \;\hookrightarrow\; (t\,\rho)\;\sigma^n_\rho\;\mu_\rho}$$

if $\mathcal{V}(\rho) \cap (\mathrm{dom}(\delta) \cup \mathrm{dom}(\tau) \cup \mathrm{dom}(\sigma) \cup \mathrm{dom}(\mu)) = \varnothing$

$$\begin{aligned} \sigma_\rho &= [x/s\rho \mid x/s \in \sigma] \\ &= (\sigma\,\rho)|_{\mathrm{dom}(\sigma)} \\ \rho &= [x/\mathsf{s}(x),\ y/0] \end{aligned}$$

narrow $\quad \mathsf{gt}(\mathsf{s}(x'),\mathsf{s}(y'))\ [x'/\mathsf{s}(x'),\ y'/\mathsf{s}(y')]^n\ \ [x'/x,\ y'/y] \hookrightarrow$
$\quad \mathsf{gt}(x,y)\ [x'/\mathsf{s}(x'),\ y'/\mathsf{s}(y')]^n\ \ [x'/x,\ y'/y]$

with $\quad \mathsf{gt}(\mathsf{s}(x),0)\ \ \varnothing^n\ \ \varnothing \hookrightarrow$
$\quad \mathsf{tt}\ \ \varnothing^n\ \ \varnothing$

**Strategy:**

1. Instantiate base terms.
   (Base term of 1st rhs should *contain* base term of 2nd lhs.)
2. Make all 4 pumping substitutions equal.
3. Make all 4 closing substitutions equal.

# Inference Rules

**(4) Instantiation**

$$\frac{s\,\delta^n\,\tau \;\hookrightarrow\; t\,\sigma^n\,\mu}{(s\,\rho)\;\delta_\rho^n\;\tau_\rho \;\hookrightarrow\; (t\,\rho)\;\sigma_\rho^n\;\mu_\rho}$$
if $\mathcal{V}(\rho) \cap (\mathrm{dom}(\delta) \cup \mathrm{dom}(\tau) \cup \mathrm{dom}(\sigma) \cup \mathrm{dom}(\mu)) = \varnothing$

$$\sigma_\rho = [x/s\rho \mid x/s \in \sigma] = (\sigma\,\rho)|_{\mathrm{dom}(\sigma)}$$

$$\rho = [x/\mathsf{s}(x),\, y/0]$$

narrow $\mathsf{gt}(\mathsf{s}(x'), \mathsf{s}(y'))\; [x'/\mathsf{s}(x'),\, y'/\mathsf{s}(y')]^n\; [x'/\mathsf{s}(x),\, y'/0] \hookrightarrow$
$\mathsf{gt}(\mathsf{s}(x), 0)\; [x'/\mathsf{s}(x'),\, y'/\mathsf{s}(y')]^n\; [x'/\mathsf{s}(x),\, y'/0]$

with $\mathsf{gt}(\mathsf{s}(x), 0)\; \varnothing^n\; \varnothing \hookrightarrow$
$\mathsf{tt}\; \varnothing^n\; \varnothing$

**Strategy:**

1. Instantiate base terms.
   (Base term of 1st rhs should *contain* base term of 2nd lhs.)
2. Make all 4 pumping substitutions equal.
3. Make all 4 closing substitutions equal.

# Inference Rules

**(4) Instantiation**

$$\frac{s\,\delta^n\,\tau \;\hookrightarrow\; t\,\sigma^n\,\mu}{(s\,\rho)\;\delta^n_\rho\;\tau_\rho \;\hookrightarrow\; (t\,\rho)\;\sigma^n_\rho\;\mu_\rho}$$

if $\mathcal{V}(\rho) \cap (\mathrm{dom}(\delta) \cup \mathrm{dom}(\tau) \cup \mathrm{dom}(\sigma) \cup \mathrm{dom}(\mu)) = \varnothing$

$$\sigma_\rho = [x/s\rho \mid x/s \in \sigma] = (\sigma\,\rho)|_{\mathrm{dom}(\sigma)}$$

$$\rho = [x/\mathsf{s}(x),\, y/0]$$

narrow $\mathsf{gt}(\mathsf{s}(x'), \mathsf{s}(y'))\ [x'/\mathsf{s}(x'),\ y'/\mathsf{s}(y')]^n\ [x'/\mathsf{s}(x),\ y'/0] \hookrightarrow$
$\mathsf{gt}(\mathsf{s}(x), 0)\ [x'/\mathsf{s}(x'),\ y'/\mathsf{s}(y')]^n\ [x'/\mathsf{s}(x),\ y'/0]$

with $\mathsf{gt}(\mathsf{s}(x), 0)\ [x'/\mathsf{s}(x'),\ y'/\mathsf{s}(y')]^n\ \varnothing \hookrightarrow$
$\mathsf{tt}\ [x'/\mathsf{s}(x'),\ y'/\mathsf{s}(y')]^n\ \varnothing$

**Strategy:**

1. Instantiate base terms.
   (Base term of 1st rhs should *contain* base term of 2nd lhs.)
2. Make all 4 pumping substitutions equal.
3. Make all 4 closing substitutions equal.

# Inference Rules

**(4) Instantiation**

$$\frac{s\,\delta^n\,\tau \;\hookrightarrow\; t\,\sigma^n\,\mu}{(s\,\rho)\;\delta^n_\rho\;\tau_\rho \;\hookrightarrow\; (t\,\rho)\;\sigma^n_\rho\;\mu_\rho}$$

if $\mathcal{V}(\rho) \cap$
$(\mathrm{dom}(\delta) \cup \mathrm{dom}(\tau) \cup$
$\mathrm{dom}(\sigma) \cup \mathrm{dom}(\mu)) = \varnothing$

$$\sigma_\rho = [x/s\rho \mid x/s \in \sigma] = (\sigma\,\rho)|_{\mathrm{dom}(\sigma)}$$

$$\rho = [x/\mathsf{s}(x),\, y/0]$$

narrow $\quad \mathsf{gt}(\mathsf{s}(x'),\mathsf{s}(y'))\;[x'/\mathsf{s}(x'),\, y'/\mathsf{s}(y')]^n\;[x'/\mathsf{s}(x),\, y'/0] \;\hookrightarrow$
$\mathsf{gt}(\mathsf{s}(x),0)\;[x'/\mathsf{s}(x'),\, y'/\mathsf{s}(y')]^n\;[x'/\mathsf{s}(x),\, y'/0]$

with $\quad \mathsf{gt}(\mathsf{s}(x),0)\;[x'/\mathsf{s}(x'),\, y'/\mathsf{s}(y')]^n\;[x'/\mathsf{s}(x),\, y'/0] \;\hookrightarrow$
$\mathsf{tt}\;[x'/\mathsf{s}(x'),\, y'/\mathsf{s}(y')]^n\;[x'/\mathsf{s}(x),\, y'/0]$

**Strategy:**

1. Instantiate base terms.
   (Base term of 1st rhs should *contain* base term of 2nd lhs.)
2. Make all 4 pumping substitutions equal.
3. Make all 4 closing substitutions equal.

# Inference Rules

## (5) Narrowing

$$\frac{s\,\sigma^n\,\mu \;\hookrightarrow\; t\,\sigma^n\,\mu \qquad u\,\sigma^n\,\mu \;\hookrightarrow\; v\,\sigma^n\,\mu}{s\,\sigma^n\,\mu \;\hookrightarrow\; t[v]_\pi\,\sigma^n\,\mu} \quad \text{if } t|_\pi = u$$

narrow $\quad \mathsf{gt}(\mathsf{s}(x'),\mathsf{s}(y'))\;[x'/\mathsf{s}(x'),\, y'/\mathsf{s}(y')]^n\;[x'/\mathsf{s}(x),\, y'/0] \;\hookrightarrow$
$\qquad \mathsf{gt}(\mathsf{s}(x),0)\;[x'/\mathsf{s}(x'),\, y'/\mathsf{s}(y')]^n\;[x'/\mathsf{s}(x),\, y'/0]$

with $\quad \mathsf{gt}(\mathsf{s}(x),0)\;[x'/\mathsf{s}(x'),\, y'/\mathsf{s}(y')]^n\;[x'/\mathsf{s}(x),\, y'/0] \;\hookrightarrow$
$\qquad \mathsf{tt}\;[x'/\mathsf{s}(x'),\, y'/\mathsf{s}(y')]^n\;[x'/\mathsf{s}(x),\, y'/0]$

$\mathsf{gt}(\mathsf{s}(x'),\mathsf{s}(y'))\;[x'/\mathsf{s}(x'),\, y'/\mathsf{s}(y')]^n\;[x'/\mathsf{s}(x),\, y'/0] \;\hookrightarrow$
$\qquad \mathsf{tt}\;[x'/\mathsf{s}(x'),\, y'/\mathsf{s}(y')]^n\;[x'/\mathsf{s}(x),\, y'/0]$

# Inference Rules

**(4) Instantiation**

$$\frac{s\,\delta^n\,\tau \;\hookrightarrow\; t\,\sigma^n\,\mu}{(s\,\rho)\;\delta^n_\rho\;\tau_\rho \;\hookrightarrow\; (t\,\rho)\;\sigma^n_\rho\;\mu_\rho}$$ if $\mathcal{V}(\rho) \cap (\mathrm{dom}(\delta) \cup \mathrm{dom}(\tau) \cup \mathrm{dom}(\sigma) \cup \mathrm{dom}(\mu)) = \varnothing$

$$\sigma_\rho = [x/s\rho \mid x/s \in \sigma] = (\sigma\,\rho)|_{\mathrm{dom}(\sigma)}$$

$$\rho = [x/\mathsf{s}(x'),\ y/\mathsf{s}(y')]$$

narrow

$$\mathsf{f}(\mathsf{tt}, x, y) \quad \varnothing^n \quad \varnothing \quad \hookrightarrow$$
$$\mathsf{f}(\mathsf{gt}(x,y), \mathsf{dbl}(x), \mathsf{s}(y)) \quad \varnothing^n \quad \varnothing$$

with

$$\mathsf{gt}(\mathsf{s}(x'), \mathsf{s}(y'))\ [x'/\mathsf{s}(x'),\ y'/\mathsf{s}(y')]^n\ [x'/\mathsf{s}(x),\ y'/0] \quad \hookrightarrow$$
$$\mathsf{tt}\ [x'/\mathsf{s}(x'),\ y'/\mathsf{s}(y')]^n\ [x'/\mathsf{s}(x),\ y'/0]$$

**Strategy:**

1. Instantiate base terms.
   (Base term of 1st rhs should *contain* base term of 2nd lhs.)
2. Make all 4 pumping substitutions equal.
3. Make all 4 closing substitutions equal.

# Inference Rules

**(4) Instantiation**

$$\frac{s\,\delta^n\,\tau \;\hookrightarrow\; t\,\sigma^n\,\mu}{(s\,\rho)\;\delta^n_\rho\;\tau_\rho \;\hookrightarrow\; (t\,\rho)\;\sigma^n_\rho\;\mu_\rho}$$

if $\mathcal{V}(\rho) \cap (\mathrm{dom}(\delta) \cup \mathrm{dom}(\tau) \cup \mathrm{dom}(\sigma) \cup \mathrm{dom}(\mu)) = \varnothing$

$$\sigma_\rho = [x/s\rho \mid x/s \in \sigma] = (\,\sigma\,\rho\,)|_{\mathrm{dom}(\sigma)}$$

$$\rho = [x/\mathsf{s}(x'),\; y/\mathsf{s}(y')]$$

narrow

$$\mathsf{f}(\mathsf{tt}, \mathsf{s}(x'), \mathsf{s}(y')) \quad \varnothing^n \quad \varnothing \quad \hookrightarrow$$
$$\mathsf{f}(\mathsf{gt}(\mathsf{s}(x'), \mathsf{s}(y')), \mathsf{dbl}(\mathsf{s}(x')), \mathsf{s}^2(y')) \quad \varnothing^n \quad \varnothing$$

with

$$\mathsf{gt}(\mathsf{s}(x'), \mathsf{s}(y'))\; [x'/\mathsf{s}(x'),\; y'/\mathsf{s}(y')]^n\; [x'/\mathsf{s}(x),\; y'/0] \quad \hookrightarrow$$
$$\mathsf{tt}\; [x'/\mathsf{s}(x'),\; y'/\mathsf{s}(y')]^n\; [x'/\mathsf{s}(x),\; y'/0]$$

**Strategy:**

1. Instantiate base terms.
   (Base term of 1st rhs should *contain* base term of 2nd lhs.)
2. Make all 4 pumping substitutions equal.
3. Make all 4 closing substitutions equal.

# Inference Rules

**(6) Instantiating $\sigma$**

$$\frac{s\,\delta^n\,\tau \;\hookrightarrow\; t\,\sigma^n\,\mu}{s\,(\delta\,\rho)^n\,\tau \;\hookrightarrow\; t\,(\sigma\,\rho)^n\,\mu}$$ if $\rho$ commutes with $\delta$, $\tau$, $\sigma$, and $\mu$

$\rho = [x'/\mathsf{s}(x'),\ y'/\mathsf{s}(y')]$

narrow
$$\mathsf{f}(\mathsf{tt}, \mathsf{s}(x'), \mathsf{s}(y'))\ [x'/\mathsf{s}(x'),\ y'/\mathsf{s}(y')]^n \quad \varnothing \quad \hookrightarrow$$
$$\mathsf{f}(\mathsf{gt}(\mathsf{s}(x'), \mathsf{s}(y')), \mathsf{dbl}(\mathsf{s}(x')), \mathsf{s}^2(y'))\ [x'/\mathsf{s}(x'),\ y'/\mathsf{s}(y')]^n \quad \varnothing$$

with
$$\mathsf{gt}(\mathsf{s}(x'), \mathsf{s}(y'))\ [x'/\mathsf{s}(x'),\ y'/\mathsf{s}(y')]^n\ [x'/\mathsf{s}(x),\ y'/0] \quad \hookrightarrow$$
$$\mathsf{tt}\ [x'/\mathsf{s}(x'),\ y'/\mathsf{s}(y')]^n\ [x'/\mathsf{s}(x),\ y'/0]$$

**Strategy:**

1. Instantiate base terms.
   (Base term of 1st rhs should *contain* base term of 2nd lhs.)
2. Make all 4 pumping substitutions equal.
3. Make all 4 closing substitutions equal.

# Inference Rules

**(7) Instantiating $\mu$**

$$\frac{s\,\delta^n\,\tau \;\hookrightarrow\; t\,\sigma^n\,\mu}{s\,\delta^n\,(\tau\,\rho) \;\hookrightarrow\; t\,\sigma^n\,(\mu\,\rho)}$$

$\rho = [x'/\mathsf{s}(x'),\; y'/0]$

narrow

$$\mathsf{f}(\mathsf{tt}, \mathsf{s}(x'), \mathsf{s}(y'))\; [x'/\mathsf{s}(x'),\; y'/\mathsf{s}(y')]^n\; [x'/\mathsf{s}(x),\; y'/0] \;\hookrightarrow$$
$$\mathsf{f}(\mathsf{gt}(\mathsf{s}(x'), \mathsf{s}(y')), \mathsf{dbl}(\mathsf{s}(x')), \mathsf{s}^2(y'))\; [x'/\mathsf{s}(x'),\; y'/\mathsf{s}(y')]^n\; [x'/\mathsf{s}(x),\; y'/0]$$

with

$$\mathsf{gt}(\mathsf{s}(x'), \mathsf{s}(y'))\; [x'/\mathsf{s}(x'),\; y'/\mathsf{s}(y')]^n\; [x'/\mathsf{s}(x),\; y'/0] \;\hookrightarrow$$
$$\mathsf{tt}\; [x'/\mathsf{s}(x'),\; y'/\mathsf{s}(y')]^n\; [x'/\mathsf{s}(x),\; y'/0]$$

**Strategy:**

1. Instantiate base terms.
   (Base term of 1st rhs should *contain* base term of 2nd lhs.)
2. Make all 4 pumping substitutions equal.
3. Make all 4 closing substitutions equal.

# Inference Rules

**(5) Narrowing**

$$\frac{s\,\sigma^n\,\mu \;\hookrightarrow\; t\,\sigma^n\,\mu \qquad u\,\sigma^n\,\mu \;\hookrightarrow\; v\,\sigma^n\,\mu}{s\,\sigma^n\,\mu \;\hookrightarrow\; t[v]_\pi\,\sigma^n\,\mu} \quad \text{if } t|_\pi = u$$

narrow

$$\mathsf{f}(\mathsf{tt},\mathsf{s}(x'),\mathsf{s}(y'))\;[x'/\mathsf{s}(x'),\;y'/\mathsf{s}(y')]^n\;[x'/\mathsf{s}(x),\;y'/0] \;\hookrightarrow$$
$$\mathsf{f}(\mathsf{gt}(\mathsf{s}(x'),\mathsf{s}(y')),\mathsf{dbl}(\mathsf{s}(x')),\mathsf{s}^2(y'))\;[x'/\mathsf{s}(x'),\;y'/\mathsf{s}(y')]^n\;[x'/\mathsf{s}(x),\;y'/0]$$

with

$$\mathsf{gt}(\mathsf{s}(x'),\mathsf{s}(y'))\;[x'/\mathsf{s}(x'),\;y'/\mathsf{s}(y')]^n\;[x'/\mathsf{s}(x),\;y'/0] \;\hookrightarrow$$
$$\mathsf{tt}\;[x'/\mathsf{s}(x'),\;y'/\mathsf{s}(y')]^n\;[x'/\mathsf{s}(x),\;y'/0]$$

$$\mathsf{f}(\mathsf{tt},\mathsf{s}(x'),\mathsf{s}(y'))\;[x'/\mathsf{s}(x'),\;y'/\mathsf{s}(y')]^n\;[x'/\mathsf{s}(x),\;y'/0] \;\hookrightarrow$$
$$\mathsf{f}(\mathsf{tt},\mathsf{dbl}(\mathsf{s}(x')),\mathsf{s}^2(y'))\;[x'/\mathsf{s}(x'),\;y'/\mathsf{s}(y')]^n\;[x'/\mathsf{s}(x),\;y'/0]$$

# Proving Non-Termination of TRSs Automatically

**Method for Non-Looping Non-Termination**

1. Let $\mathcal{S}$ be a set of pattern rules corresponding to $\mathcal{R}$.
2. Check if some $p \hookrightarrow q \in \mathcal{S}$ is obviously non-terminating.
   If yes: stop with "non-termination".
3. Modify some $p \hookrightarrow q \in \mathcal{S}$ by narrowing to obtain $p' \hookrightarrow q'$.
4. Let $\mathcal{S} := \mathcal{S} \cup \{p' \hookrightarrow q'\}$ and go to Step 2.

## Contributions

- **pattern rules** $p \hookrightarrow q$
  to represent *sets* of rewrite sequences $\{p(n) \to_{\mathcal{R}}^{+} q(n) \mid n \in \mathbb{N}\}$
- **inference rules** to deduce new correct pattern rules
- **criterion for obvious non-termination** of pattern rules

# Non-Termination Criterion

## Criterion for Non-Termination of $\mathcal{R}$

- $s\,\delta^n\,\tau \hookrightarrow t\,\sigma^n\,\mu$ correct w.r.t. $\mathcal{R}$
- $s\,\delta^a = t|_\pi$ for some $a \in \mathbb{N}$
- $\sigma = \delta^b\,\delta'$, $\mu = \tau\,\tau'$ for some $\delta', \tau'$ and some $b \in \mathbb{N}$ where $\delta'$ commutes with $\delta$ and $\tau$

$$
\begin{array}{llllllll}
s & \delta^n & \tau & \to_{\mathcal{R}}^+ & t & & \sigma^n & \mu \\
 & & & \trianglerighteq & t|_\pi & & \sigma^n & \mu \\
 & & & = & s & \delta^a & \sigma^n & \mu \\
 & & & = & s & \delta^a & (\delta^b\,\delta')^n & (\tau\,\tau') \\
 & & & = & s & \delta^{a+b\cdot n} & & \tau\,\delta'^n\,\tau'
\end{array}
$$

Thus: $s\,\delta^n\,\tau$ rewrites to an instance of $s\,\delta^{a+b\cdot n}\,\tau$

# Non-Termination Criterion

**Criterion for Non-Termination of $\mathcal{R}$**

- $s\,\delta^n\,\tau \;\hookrightarrow t\,\sigma^n\,\mu$ — correct w.r.t. $\mathcal{R}$
- $s\,\delta^a \;=\; t|_\pi$ — for some $a \in \mathbb{N}$
- $\sigma = \delta^b\,\delta'$, $\quad \mu = \tau\,\tau'$ — for some $\delta', \tau'$ and some $b \in \mathbb{N}$ where $\delta'$ commutes with $\delta$ and $\tau$

$$\mathsf{f}(\mathsf{tt}, \mathsf{s}^2(x'), \mathsf{s}(y'))\ [x'/\mathsf{s}(x'),\ y'/\mathsf{s}(y')]^n\ \ [y'/0] \quad \hookrightarrow$$
$$\mathsf{f}(\mathsf{tt}, \mathsf{s}^3(x'), \mathsf{s}^2(y'))\ [x'/\mathsf{s}^2(x'),\ y'/\mathsf{s}(y')]^n\ [x'/\mathsf{s}(\mathsf{mul}(\mathsf{s}^2(0), x')),\ y'/0]$$

Thus: $\mathsf{f}(\mathsf{tt}, \mathsf{s}^{n+2}(x'), \mathsf{s}^{n+1}(0)) \quad \to^+_{\mathcal{R}} \quad \mathsf{f}(\mathsf{tt}, \mathsf{s}^{2\cdot n+4}(\mathsf{mul}(\mathsf{s}^2(0), x')), \mathsf{s}^{n+2}(0))$

# Non-Termination Criterion

**Criterion for Non-Termination of $\mathcal{R}$**

- $s\,\delta^n\,\tau \hookrightarrow t\,\sigma^n\,\mu$ correct w.r.t. $\mathcal{R}$
- $s\,\delta^a = t|_\pi$ for some $a \in \mathbb{N}$
- $\sigma = \delta^b\,\delta'$, $\mu = \tau\,\tau'$ for some $\delta', \tau'$ and some $b \in \mathbb{N}$ where $\delta'$ commutes with $\delta$ and $\tau$

$$\mathsf{f}(\mathsf{tt}, \mathsf{s}^2(x'), \mathsf{s}(y'))\,[x'/\mathsf{s}(x'),\, y'/\mathsf{s}(y')]^n\;[y'/\mathsf{0}] \hookrightarrow$$
$$\mathsf{f}(\mathsf{tt}, \mathsf{s}^3(x'), \mathsf{s}^2(y'))\,[x'/\mathsf{s}^2(x'),\, y'/\mathsf{s}(y')]^n\;[x'/\mathsf{s}(\mathsf{mul}(\mathsf{s}^2(\mathsf{0}), x')),\, y'/\mathsf{0}]$$

- $\underbrace{\mathsf{f}(\mathsf{tt}, \mathsf{s}^2(x'), \mathsf{s}(y'))}_{s}\;\underbrace{[x'/\mathsf{s}(x'),\, y'/\mathsf{s}(y')]}_{\delta} = \underbrace{\mathsf{f}(\mathsf{tt}, \mathsf{s}^3(x'), \mathsf{s}^2(y'))}_{t}$
- $\underbrace{[x'/\mathsf{s}^2(x'),\, y'/\mathsf{s}(y')]}_{\sigma} = \underbrace{[x'/\mathsf{s}(x'),\, y'/\mathsf{s}(y')]}_{\delta}\;\underbrace{[x'/\mathsf{s}(x')]}_{\delta'}$

  $\underbrace{[x'/\mathsf{s}(\mathsf{mul}(\mathsf{s}^2(\mathsf{0}), x')),\, y'/\mathsf{0}]}_{\mu} = \underbrace{[y'/\mathsf{0}]}_{\tau}\;\underbrace{[x'/\mathsf{s}(\mathsf{mul}(\mathsf{s}^2(\mathsf{0}), x'))]}_{\tau'}$

# Proving Non-Termination of TRSs Automatically

**Method for Non-Looping Non-Termination**

1. Let $\mathcal{S}$ be a set of pattern rules corresponding to $\mathcal{R}$.
2. Check if some $p \hookrightarrow q \in \mathcal{S}$ is obviously non-terminating.
   If yes: stop with "non-termination".
3. Modify some $p \hookrightarrow q \in \mathcal{S}$ by narrowing to obtain $p' \hookrightarrow q'$.
4. Let $\mathcal{S} := \mathcal{S} \cup \{p' \hookrightarrow q'\}$ and go to Step 2.

## Contributions

- **pattern rules** $p \hookrightarrow q$
  to represent *sets* of rewrite sequences $\{p(n) \to_{\mathcal{R}}^{+} q(n) \mid n \in \mathbb{N}\}$
- **inference rules** to deduce new correct pattern rules
- **criterion for obvious non-termination** of pattern rules
- **strategy** to apply the inference rules and the non-termination criterion
- **implementation** and evaluation in **AProVE**

# Proving Non-Termination of TRSs Automatically

- DP framework proves and disproves termination of TRSs
- approach detects also non-looping non-terminating TRSs by combining
  - techniques for looping non-termination of TRSs
  - techniques for non-looping non-termination of SRSs (Oppelt '08)

## Contributions

- **pattern rules** $p \hookrightarrow q$
  to represent *sets* of rewrite sequences $\{p(n) \to_{\mathcal{R}}^{+} q(n) \mid n \in \mathbb{N}\}$
- **inference rules** to deduce new correct pattern rules
- **criterion for obvious non-termination** of pattern rules
- **strategy** to apply the inference rules and the non-termination criterion
- **implementation** and evaluation in **AProVE** (also as **DP processor**)

# Overview

## I. Termination of Term Rewriting

1. Termination of Term Rewrite Systems
2. Non-Termination of Term Rewrite Systems
3. Complexity of Term Rewrite Systems (CADE '11)
4. Termination of Integer Term Rewrite Systems

## II. Termination of Programs

1. Termination of Functional Programs (Haskell)
2. Termination of Logic Programs (Prolog)
3. Termination of Imperative Programs (Java)

## Termination Analysis of TRSs

- useful for termination of programs (Java, Haskell, Prolog, ...)
- Dependency Pair Framework
  - modular combination of different techniques
  - automatable

## Complexity Analysis of TRSs

- should be useful for of programs ⇒ *Innermost Runtime Complexity*
- adapt Dependency Pair Framework
  - Hirokawa & Moser (IJCAR '08, LPAR '08)
    - first adaption of DPs for complexity
    - not modular
  - Zankl & Korp (RTA '10)
    - modular approach based on relative rewriting
    - for *Derivational Complexity*
      (cannot exploit strength of DPs for innermost rewriting)
  - new approach: *direct* adaption of DP framework
    - modular combination of different techniques
    - automated and more powerful than previous approaches

# Innermost Runtime Complexity

$$\mathcal{R}: \quad \mathsf{double}(0) \to 0$$
$$\mathsf{double}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{s}(\mathsf{double}(x)))$$

- **Derivation Height** $\mathsf{dh}(t)$**:** length of longest $\xrightarrow{\mathrm{i}}_{\mathcal{R}}$-sequence with $t$
  - $\mathsf{dh}(\,\mathsf{double}(\mathsf{s}^k(0))\,) = k+1$
  - $\mathsf{dh}(\,\mathsf{double}^k(\mathsf{s}(0))\,) \approx 2^k$

- **Basic Terms** $f(t_1, \ldots, t_n)$

  $f$ defined symbol (double), $t_1, \ldots, t_n$ no defined symbols (s, 0)

- **Complexity** $\iota_{\mathcal{R}}$ **of TRS** $\mathcal{R}$**:**
  length of longest $\xrightarrow{\mathrm{i}}_{\mathcal{R}}$-sequence with basic term $t$ where $|t| \leq n$

  $\iota_{\mathcal{R}} = \mathcal{P}ol_0$ iff length $\in \mathcal{O}(1)$ $\qquad$ $\iota_{\mathcal{R}} = \mathcal{P}ol_1$ iff length $\in \mathcal{O}(n)$

  $\iota_{\mathcal{R}} = \mathcal{P}ol_2$ iff length $\in \mathcal{O}(n^2)$ $\qquad$ $\ldots$

- Example: $\iota_{\mathcal{R}} = \mathcal{P}ol_1$

# Dependency Tuples

$$\begin{aligned} \mathsf{m}(x,y) &\to \mathsf{if}(\mathsf{gt}(x,y),x,y) & \mathsf{gt}(0,k) &\to \mathsf{false} & \mathsf{p}(0) &\to 0 \\ \mathsf{if}(\mathsf{true},x,y) &\to \mathsf{s}(\mathsf{m}(\mathsf{p}(x),y)) & \mathsf{gt}(\mathsf{s}(n),0) &\to \mathsf{true} & \mathsf{p}(\mathsf{s}(n)) &\to n \\ \mathsf{if}(\mathsf{false},x,y) &\to 0 & \mathsf{gt}(\mathsf{s}(n),\mathsf{s}(k)) &\to \mathsf{gt}(n,k) \end{aligned}$$

- **Termination Analysis: Dependency Pairs**

  compare lhs with subterms of rhs that start with defined symbol

$$\begin{aligned} \mathsf{m}^\sharp(x,y) &\to \mathsf{if}^\sharp(\mathsf{gt}(x,y),x,y) & \mathsf{if}^\sharp(\mathsf{true},x,y) &\to \mathsf{m}^\sharp(\mathsf{p}(x),y) \\ \mathsf{m}^\sharp(x,y) &\to \mathsf{gt}^\sharp(x,y) & \mathsf{if}^\sharp(\mathsf{true},x,y) &\to \mathsf{p}^\sharp(x) \\ & & \mathsf{gt}^\sharp(\mathsf{s}(n),\mathsf{s}(k)) &\to \mathsf{gt}^\sharp(n,k) \end{aligned}$$

- **Complexity Analysis: Dependency Tuples**

  compare lhs with *all* defined subterms of rhs *at once*

$$\begin{aligned} \mathsf{m}^\sharp(x,y) &\to \mathrm{Com}_2(\mathsf{if}^\sharp(\mathsf{gt}(x,y),x,y),\mathsf{gt}^\sharp(x,y)) & \mathsf{p}^\sharp(0) &\to \mathrm{Com}_0 \\ \mathsf{if}^\sharp(\mathsf{true},x,y) &\to \mathrm{Com}_2(\mathsf{m}^\sharp(\mathsf{p}(x),y),\mathsf{p}^\sharp(x)) & \mathsf{p}^\sharp(\mathsf{s}(n)) &\to \mathrm{Com}_0 \\ \mathsf{if}^\sharp(\mathsf{false},x,y) &\to \mathrm{Com}_0 & \mathsf{gt}^\sharp(0,k) &\to \mathrm{Com}_0 \\ & & \mathsf{gt}^\sharp(\mathsf{s}(n),0) &\to \mathrm{Com}_0 \\ & & \mathsf{gt}^\sharp(\mathsf{s}(n),\mathsf{s}(k)) &\to \mathrm{Com}_1(\mathsf{gt}^\sharp(n,k)) \end{aligned}$$

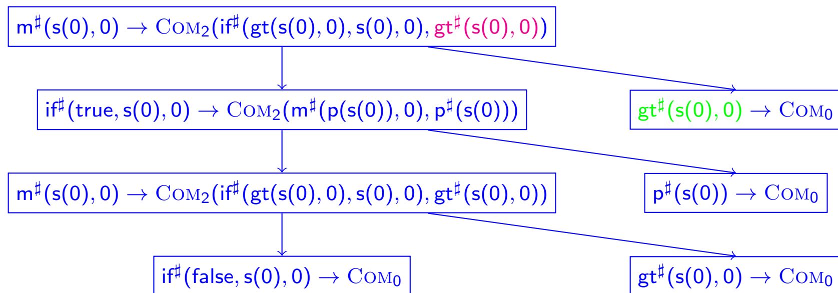
# Chain Trees

$\mathcal{DT}(\mathcal{R})$ :

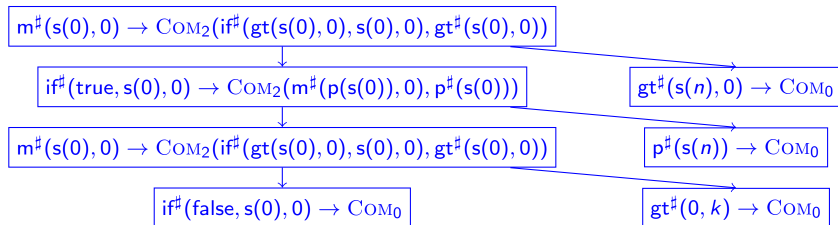
$$\mathsf{m}^\sharp(x,y) \to \mathrm{Com}_2(\mathsf{if}^\sharp(\mathsf{gt}(x,y),x,y),\mathsf{gt}^\sharp(x,y))$$
$$\mathsf{if}^\sharp(\mathsf{true},x,y) \to \mathrm{Com}_2(\mathsf{m}^\sharp(\mathsf{p}(x),y),\mathsf{p}^\sharp(x))$$
$$\mathsf{if}^\sharp(\mathsf{false},x,y) \to \mathrm{Com}_0$$
$$\mathsf{p}^\sharp(0) \to \mathrm{Com}_0$$
$$\mathsf{p}^\sharp(\mathsf{s}(n)) \to \mathrm{Com}_0$$
$$\mathsf{gt}^\sharp(0,k) \to \mathrm{Com}_0$$
$$\mathsf{gt}^\sharp(\mathsf{s}(n),0) \to \mathrm{Com}_0$$
$$\mathsf{gt}^\sharp(\mathsf{s}(n),\mathsf{s}(k)) \to \mathrm{Com}_1(\mathsf{gt}^\sharp(n,k))$$

**$(\mathcal{D},\mathcal{R})$-Chain Tree:**

Edge $\sigma_1(u^\sharp \to \mathrm{Com}_n(v_1^\sharp,\ldots,v_n^\sharp))$ to $\sigma_2(w^\sharp \to \mathrm{Com}_m(\ldots))$ if $v_i^\sharp\sigma_1 \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} w^\sharp\sigma_2$



# Chain Trees and Complexity




$\iota_{\mathcal{R}}$: length of longest $\xrightarrow{\mathrm{i}}_{\mathcal{R}}$-sequence for $|t| \leq n$

$\iota_{\langle \mathcal{D},\mathcal{S},\mathcal{R}\rangle}$: maximal number of nodes from $\mathcal{S}$ in chain tree with root $t^\sharp \to \textsc{Com}(\ldots)$ for $|t| \leq n$

**Theorem**

If $\mathcal{D} = DT(\mathcal{R})$, then $\iota_{\mathcal{R}} \leq \iota_{\langle \mathcal{D},\mathcal{D},\mathcal{R}\rangle}$.

# Chain Trees and Complexity

**Theorem**

If $\mathcal{D} = DT(\mathcal{R})$, then $\iota_{\mathcal{R}} \leq \iota_{\langle \mathcal{D}, \mathcal{D}, \mathcal{R} \rangle}$.

$\Rightarrow$ Find out $\iota_{\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle}$

$\Rightarrow$ Repeatedly replace DT problem $\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle$ by *simpler* $\langle \mathcal{D}', \mathcal{S}', \mathcal{R}' \rangle$, examine $\iota_{\langle \mathcal{D}', \mathcal{S}', \mathcal{R}' \rangle}$

$\Rightarrow$ Start with canonical DT problem $\langle \mathrm{DT}(\mathcal{R}), \mathrm{DT}(\mathcal{R}), \mathcal{R} \rangle$

**DT Processor:** $Proc(P) = (c, P')$ $\quad$ $P, P'$ DT problems, $c \in \{\mathcal{P}ol_0, \mathcal{P}ol_1, \ldots\}$
where $\iota_P \leq \max(c, \iota_{P'})$

**Proof Chain:** $P_0 \overset{c_1}{\rightsquigarrow} P_1 \overset{c_2}{\rightsquigarrow} P_2 \overset{c_3}{\rightsquigarrow} \ldots \overset{c_k}{\rightsquigarrow} P_k$ $\qquad$ $P_0 = \langle \mathrm{DT}(\mathcal{R}), \mathrm{DT}(\mathcal{R}), \mathcal{R} \rangle$ canonical

$\iota_{\mathcal{R}} \leq \iota_{P_0} \leq \max(c_1, c_2, \ldots, c_k)$ $\qquad$ $P_k = \langle \mathcal{D}_k, \varnothing, \mathcal{R}_k \rangle$ solved

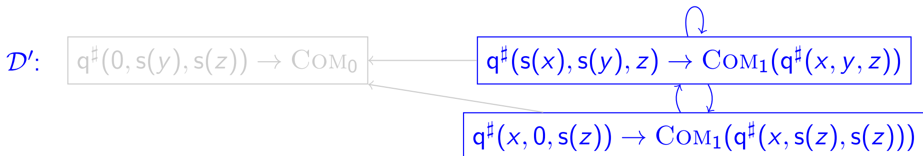
# Leaf Removal Processor

**Dependency Graph:** edge from DT $u \to v$ to $w \to t$ in dep. graph iff
edge from $\sigma_1(u \to v)$ to $\sigma_2(w \to t)$ in chain tree

**Leaf Removal Processor:** $\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle \overset{\mathcal{P}ol_0}{\leadsto} \langle \mathcal{D} \setminus \{w \to t\}, \mathcal{S} \setminus \{w \to t\}, \mathcal{R} \rangle$
if $w \to t$ is leaf in dependency graph

**Example:** $\langle \mathcal{D}, \mathcal{D}, \mathcal{R} \rangle \overset{\mathcal{P}ol_0}{\leadsto} \langle \mathcal{D}', \mathcal{D}', \mathcal{R} \rangle$

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



# Usable Rules Processor

**Usable Rules $\mathcal{U}_\mathcal{R}(\mathcal{D})$:** rules from $\mathcal{R}$ that can reduce rhs of $\mathcal{D}$

**Usable Rules Processor:** $\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle \overset{\mathcal{P}ol_0}{\rightsquigarrow} \langle \mathcal{D},\ \mathcal{S},\ \mathcal{U}_\mathcal{R}(\mathcal{D}) \rangle$

**Example:** $\langle \mathcal{D}, \mathcal{D}, \mathcal{R} \rangle \overset{\mathcal{P}ol_0}{\rightsquigarrow} \langle \mathcal{D}', \mathcal{D}', \mathcal{R} \rangle \overset{\mathcal{P}ol_0}{\rightsquigarrow} \langle \mathcal{D}', \mathcal{D}', \varnothing \rangle$

$\mathcal{U}_\mathcal{R}(\mathcal{D}')$:

$\mathcal{D}'$:

$\mathsf{q}^\sharp(\mathsf{s}(x), \mathsf{s}(y), z) \to \mathrm{COM}_1(\mathsf{q}^\sharp(x, y, z))$

$\mathsf{q}^\sharp(x, 0, \mathsf{s}(z)) \to \mathrm{COM}_1(\mathsf{q}^\sharp(x, \mathsf{s}(z), \mathsf{s}(z)))$

# Extended DT Problems

**Extended DT Problem:** $\langle \mathcal{D}, \mathcal{S}, \mathcal{K}, \mathcal{R} \rangle$

- when computing $\iota_{\langle \mathcal{D},\mathcal{S},\mathcal{R}\rangle}$, we already took $\iota_{\langle \mathcal{D},\mathcal{K},\mathcal{R}\rangle}$ into account
- $\iota_{\langle \mathcal{D},\mathcal{S},\mathcal{K},\mathcal{R}\rangle} = \begin{cases} \iota_{\langle \mathcal{D},\mathcal{S},\mathcal{R}\rangle}, & \text{if } \iota_{\langle \mathcal{D},\mathcal{S},\mathcal{R}\rangle} > \iota_{\langle \mathcal{D},\mathcal{K},\mathcal{R}\rangle} \\ \mathcal{P}ol_0, & \text{if } \iota_{\langle \mathcal{D},\mathcal{S},\mathcal{R}\rangle} \leq \iota_{\langle \mathcal{D},\mathcal{K},\mathcal{R}\rangle} \end{cases}$

**Canonical Extended DT Problem:** $\langle \mathrm{DT}(\mathcal{R}), \mathrm{DT}(\mathcal{R}), \varnothing, \mathcal{R} \rangle$

**Example:** $\langle \mathcal{D}, \mathcal{D}, \varnothing, \mathcal{R} \rangle \overset{\mathcal{P}ol_0}{\rightsquigarrow} \langle \mathcal{D}', \mathcal{D}', \varnothing, \mathcal{R} \rangle \overset{\mathcal{P}ol_0}{\rightsquigarrow} \langle \mathcal{D}', \mathcal{D}', \varnothing, \varnothing \rangle$

$\mathcal{D}'$:

$\mathsf{q}^\sharp(\mathsf{s}(x), \mathsf{s}(y), z) \to \mathrm{Com}_1(\mathsf{q}^\sharp(x, y, z))$

$\mathsf{q}^\sharp(x, 0, \mathsf{s}(z)) \to \mathrm{Com}_1(\mathsf{q}^\sharp(x, \mathsf{s}(z), \mathsf{s}(z)))$

# Reduction Pair Processor

**Termination:** $\ell \succsim r$ for all DPs and rules, remove DPs with $\ell \succ r$

**Complexity:** $\ell \succsim r$ for all DTs and rules, move DTs with $\ell \succ r$ *from* $\mathcal{S}$ *to* $\mathcal{K}$

**Reduction Pair Processor:** $\langle \mathcal{D}, \mathcal{S}, \mathcal{K}, \mathcal{R} \rangle \overset{\mathcal{P}ol_m}{\rightsquigarrow} \langle \mathcal{D}, \mathcal{S} \setminus \mathcal{D}_{\succ}, \mathcal{K} \cup \mathcal{D}_{\succ}, \mathcal{R} \rangle$ if

- $\mathcal{D} \subseteq \succsim \cup \succ, \quad \mathcal{R} \subseteq \succsim$
- $m$ is the maximal degree of polynomials $[f^\sharp]$

**Example:** $\langle \mathcal{D}, \mathcal{D}, \varnothing, \mathcal{R} \rangle \overset{\mathcal{P}ol_0}{\rightsquigarrow} \langle \mathcal{D}', \mathcal{D}', \varnothing, \mathcal{R} \rangle \overset{\mathcal{P}ol_0}{\rightsquigarrow} \langle \mathcal{D}', \mathcal{D}', \varnothing, \varnothing \rangle$

**Polynomial Order**

$[\mathrm{COM}_1](x) = x$

$[\mathsf{q}^\sharp](x, y, z) = x$

$[\mathsf{s}](x) = x + 1$

(1) $\mathsf{q}^\sharp(\mathsf{s}(x), \mathsf{s}(y), z) \to \mathrm{COM}_1(\mathsf{q}^\sharp(x, y, z))$

(2) $\mathsf{q}^\sharp(x, 0, \mathsf{s}(z)) \to \mathrm{COM}_1(\mathsf{q}^\sharp(x, \mathsf{s}(z), \mathsf{s}(z)))$

# Reduction Pair Processor

**Termination:** $\ell \gtrsim r$ for all DPs and rules, remove DPs with $\ell \succ r$

**Complexity:** $\ell \gtrsim r$ for all DTs and rules, move DTs with $\ell \succ r$ *from* $\mathcal{S}$ *to* $\mathcal{K}$

**Reduction Pair Processor:** $\langle \mathcal{D}, \mathcal{S}, \mathcal{K}, \mathcal{R} \rangle \overset{\mathcal{P}ol_m}{\rightsquigarrow} \langle \mathcal{D},\ \mathcal{S} \setminus \mathcal{D}_{\succ},\ \mathcal{K} \cup \mathcal{D}_{\succ},\ \mathcal{R} \rangle$ if

- $\mathcal{D} \subseteq \gtrsim \cup \succ, \quad \mathcal{R} \subseteq \gtrsim$
- $m$ is the maximal degree of polynomials $[f^\sharp]$

**Example:** $\langle \mathcal{D}, \mathcal{D}, \varnothing, \mathcal{R} \rangle \overset{\mathcal{P}ol_0}{\rightsquigarrow} \langle \mathcal{D}', \mathcal{D}', \varnothing, \mathcal{R} \rangle \overset{\mathcal{P}ol_0}{\rightsquigarrow} \langle \mathcal{D}', \mathcal{D}', \varnothing, \varnothing \rangle$

$\overset{\mathcal{P}ol_1}{\rightsquigarrow} \langle \mathcal{D}', \{(2)\}, \{(1)\}, \varnothing \rangle$

**Polynomial Order**

$[\mathrm{COM}_1](x) = x$

$[\mathsf{q}^\sharp](x, y, z) = x$

$[\mathsf{s}](x) = x + 1$

(1) $\mathsf{q}^\sharp(\mathsf{s}(x), \mathsf{s}(y), z) \succ \mathrm{COM}_1(\mathsf{q}^\sharp(x, y, z))$

(2) $\mathsf{q}^\sharp(x, 0, \mathsf{s}(z)) \gtrsim \mathrm{COM}_1(\mathsf{q}^\sharp(x, \mathsf{s}(z), \mathsf{s}(z)))$

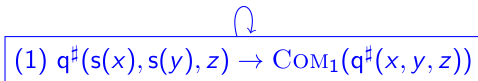
# Knowledge Propagation Processor

**Lemma:** $\iota_{\langle \mathcal{D}, \{w \to t\}, \mathcal{R} \rangle} \leq \iota_{\langle \mathcal{D}, Pre(w \to t), \mathcal{R} \rangle}$

- $Pre(w \to t)$: all predecessors of $w \to t$ in dependency graph
- $\langle \mathcal{D}, \mathcal{S}, \mathcal{K}, \mathcal{R} \rangle$: do not take $\iota_{\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle}$ into account if $\iota_{\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle} \leq \iota_{\langle \mathcal{D}, \mathcal{K}, \mathcal{R} \rangle}$

**KP Processor:** $\langle \mathcal{D}, \mathcal{S}, \mathcal{K}, \mathcal{R} \rangle \overset{\mathcal{P}ol_0}{\rightsquigarrow} \langle \mathcal{D},\ \mathcal{S} \setminus \{w \to t\},\ \mathcal{K} \cup \{w \to t\},\ \mathcal{R} \rangle$
if $w \to t \in \mathcal{S}$ and $Pre(w \to t) \subseteq \mathcal{K}$

**Example:** $\langle \mathcal{D}, \mathcal{D}, \varnothing, \mathcal{R} \rangle \overset{\mathcal{P}ol_0}{\rightsquigarrow} \langle \mathcal{D}', \mathcal{D}', \varnothing, \mathcal{R} \rangle \overset{\mathcal{P}ol_0}{\rightsquigarrow} \langle \mathcal{D}', \mathcal{D}', \varnothing, \varnothing \rangle$
$\overset{\mathcal{P}ol_1}{\rightsquigarrow} \langle \mathcal{D}', \{(2)\}, \{(1)\}, \varnothing \rangle \overset{\mathcal{P}ol_0}{\rightsquigarrow} \langle \mathcal{D}', \varnothing, \{(1), (2)\}, \varnothing \rangle$


(1) $\mathsf{q}^\sharp(\mathsf{s}(x), \mathsf{s}(y), z) \to \mathrm{COM}_1(\mathsf{q}^\sharp(x, y, z))$

$Pre(\ (2)\ ) = \{(1)\}$

(2) $\mathsf{q}^\sharp(x, 0, \mathsf{s}(z)) \to \mathrm{COM}_1(\mathsf{q}^\sharp(x, \mathsf{s}(z), \mathsf{s}(z)))$

# Knowledge Propagation Processor

**Proof Chain:** $P_0 \overset{c_1}{\rightsquigarrow} \ldots \overset{c_k}{\rightsquigarrow} P_k$ $\qquad \iota_{\mathcal{R}} \leq \iota_{P_0} \leq \max(c_1, \ldots, c_k)$

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

**Example:** $\langle \mathcal{D}, \mathcal{D}, \varnothing, \mathcal{R} \rangle \overset{\mathcal{P}ol_0}{\rightsquigarrow} \langle \mathcal{D}', \mathcal{D}', \varnothing, \mathcal{R} \rangle \overset{\mathcal{P}ol_0}{\rightsquigarrow} \langle \mathcal{D}', \mathcal{D}', \varnothing, \varnothing \rangle$

$\overset{\mathcal{P}ol_1}{\rightsquigarrow} \langle \mathcal{D}', \{(2)\}, \{(1)\}, \varnothing \rangle \overset{\mathcal{P}ol_0}{\rightsquigarrow} \langle \mathcal{D}', \varnothing, \{(1), (2)\}, \varnothing \rangle$

$$\iota_{\mathcal{R}} \leq \max(\mathcal{P}ol_0, \mathcal{P}ol_0, \mathcal{P}ol_1, \mathcal{P}ol_0) = \mathcal{P}ol_1$$

# Narrowing Processor

**Narrowing Processor:** $\langle \mathcal{D}, \mathcal{S}, \mathcal{K}, \mathcal{R} \rangle \overset{\mathcal{P}ol_0}{\rightsquigarrow} \langle \mathcal{D}', \mathcal{S}', \mathcal{K}', \mathcal{R} \rangle$ where in $\mathcal{D}'$, $\mathcal{S}'$, some $w \to t$ is replaced by all its narrowings

$\langle \mathcal{D}, \mathcal{D}, \varnothing, \mathcal{R} \rangle$

$$
\mathcal{R} : \begin{array}{rcl} \mathsf{m}(x,y) & \to & \mathsf{if}(\mathsf{gt}(x,y),x,y) \\ \mathsf{if}(\mathsf{false},x,y) & \to & \mathsf{0} \\ \mathsf{if}(\mathsf{true},x,y) & \to & \mathsf{s}(\mathsf{m}(\mathsf{p}(x),y)) \end{array}
\quad
\begin{array}{rcl} \mathsf{gt}(\mathsf{0},k) & \to & \mathsf{false} \\ \mathsf{gt}(\mathsf{s}(n),\mathsf{0}) & \to & \mathsf{true} \\ \mathsf{gt}(\mathsf{s}(n),\mathsf{s}(k)) & \to & \mathsf{gt}(n,k) \end{array}
\quad
\begin{array}{rcl} \mathsf{p}(\mathsf{0}) & \to & \mathsf{0} \\ \mathsf{p}(\mathsf{s}(n)) & \to & n \end{array}
$$

# Narrowing Processor

**Narrowing Processor:** $\langle \mathcal{D}, \mathcal{S}, \mathcal{K}, \mathcal{R} \rangle \overset{\mathcal{P}ol_0}{\rightsquigarrow} \langle \mathcal{D}', \mathcal{S}', \mathcal{K}', \mathcal{R} \rangle$ where in $\mathcal{D}'$, $\mathcal{S}'$, some $w \to t$ is replaced by all its narrowings

Narrowings of $\mathsf{m}^\sharp(x,y) \to \mathrm{Com}_2(\mathsf{if}^\sharp(\mathsf{gt}(x,y),x,y),\ \mathsf{gt}^\sharp(x,y))$

- $\mathsf{m}^\sharp(0,k) \to \mathrm{Com}_2(\mathsf{if}^\sharp(\mathsf{false},0,k),\mathsf{gt}^\sharp(0,k))$
- $\mathsf{m}^\sharp(\mathsf{s}(n),0) \to \mathrm{Com}_2(\mathsf{if}^\sharp(\mathsf{true},\mathsf{s}(n),0),\mathsf{gt}^\sharp(\mathsf{s}(n),0))$
- $\mathsf{m}^\sharp(\mathsf{s}(n),\mathsf{s}(k)) \to \mathrm{Com}_2(\mathsf{if}^\sharp(\mathsf{gt}(n,k),\mathsf{s}(n),\mathsf{s}(k)),\mathsf{gt}^\sharp(\mathsf{s}(n),\mathsf{s}(k)))$

$\langle \mathcal{D}, \mathcal{D}, \varnothing, \mathcal{R} \rangle \overset{\mathcal{P}ol_0{}^*}{\rightsquigarrow} \langle \mathcal{D}_1, \mathcal{D}_1, \varnothing, \mathcal{R}_1 \rangle$

$\mathcal{R}_1$ :

$$\begin{aligned} \mathsf{gt}(0,k) &\to \mathsf{false} & \mathsf{p}(0) &\to 0 \\ \mathsf{gt}(\mathsf{s}(n),0) &\to \mathsf{true} & \mathsf{p}(\mathsf{s}(n)) &\to n \\ \mathsf{gt}(\mathsf{s}(n),\mathsf{s}(k)) &\to \mathsf{gt}(n,k) \end{aligned}$$

$\mathcal{D}_1$ : $\mathsf{m}^\sharp(x,y) \to \mathrm{Com}_2(\mathsf{if}^\sharp(\mathsf{gt}(x,y),x,y),\mathsf{gt}^\sharp(x,y))$

$\mathsf{if}^\sharp(\mathsf{true},x,y) \to \mathrm{Com}_2(\mathsf{m}^\sharp(\mathsf{p}(x),y),\mathsf{p}^\sharp(x))$ $\quad$ $\mathsf{gt}^\sharp(\mathsf{s}(n),\mathsf{s}(k)) \to \mathrm{Com}_1(\mathsf{gt}^\sharp(n,k))$

# Narrowing Processor

**Narrowing Processor:** $\langle \mathcal{D}, \mathcal{S}, \mathcal{K}, \mathcal{R} \rangle \overset{\mathcal{P}ol_0}{\rightsquigarrow} \langle \mathcal{D}', \mathcal{S}', \mathcal{K}', \mathcal{R} \rangle$ where in $\mathcal{D}'$, $\mathcal{S}'$, some $w \to t$ is replaced by all its narrowings

Narrowings of $\mathsf{m}^\sharp(x,y) \;\to\; \mathrm{Com}_2(\mathsf{if}^\sharp(\mathsf{gt}(x,y),x,y),\, \mathsf{gt}^\sharp(x,y))$

- $\mathsf{m}^\sharp(\mathsf{s}(n),0) \to \mathrm{Com}_2(\mathsf{if}^\sharp(\mathsf{true},\mathsf{s}(n),0),\mathsf{gt}^\sharp(\mathsf{s}(n),0))$
- $\mathsf{m}^\sharp(\mathsf{s}(n),\mathsf{s}(k)) \to \mathrm{Com}_2(\mathsf{if}^\sharp(\mathsf{gt}(n,k),\mathsf{s}(n),\mathsf{s}(k)),\mathsf{gt}^\sharp(\mathsf{s}(n),\mathsf{s}(k)))$

$$\langle \mathcal{D}, \mathcal{D}, \varnothing, \mathcal{R} \rangle \quad \overset{\mathcal{P}ol_0{}^*}{\rightsquigarrow} \langle \mathcal{D}_1, \mathcal{D}_1, \varnothing, \mathcal{R}_1 \rangle \overset{\mathcal{P}ol_0{}^*}{\rightsquigarrow} \langle \mathcal{D}_2, \mathcal{D}_2, \varnothing, \mathcal{R}_1 \rangle$$

$\mathcal{R}_1$ :

$$\begin{array}{rcl} \mathsf{gt}(0,k) & \to & \mathsf{false} \\ \mathsf{gt}(\mathsf{s}(n),0) & \to & \mathsf{true} \\ \mathsf{gt}(\mathsf{s}(n),\mathsf{s}(k)) & \to & \mathsf{gt}(n,k) \end{array} \qquad \begin{array}{rcl} \mathsf{p}(0) & \to & 0 \\ \mathsf{p}(\mathsf{s}(n)) & \to & n \end{array}$$

$\mathcal{D}_2$:

$$\begin{array}{l} \mathsf{m}^\sharp(\mathsf{s}(n),0) \to \mathrm{Com}_2(\mathsf{if}^\sharp(\mathsf{true},\mathsf{s}(n),0),\mathsf{gt}^\sharp(\mathsf{s}(n),0)) \\ \mathsf{m}^\sharp(\mathsf{s}(n),\mathsf{s}(k)) \to \mathrm{Com}_2(\mathsf{if}^\sharp(\mathsf{gt}(n,k),\mathsf{s}(n),\mathsf{s}(k)),\mathsf{gt}^\sharp(\mathsf{s}(n),\mathsf{s}(k))) \\ \mathsf{if}^\sharp(\mathsf{true},x,y) \to \mathrm{Com}_2(\mathsf{m}^\sharp(\mathsf{p}(x),y),\mathsf{p}^\sharp(x)) \qquad \mathsf{gt}^\sharp(\mathsf{s}(n),\mathsf{s}(k)) \to \mathrm{Com}_1(\mathsf{gt}^\sharp(n,k)) \end{array}$$

# Narrowing Processor

**Narrowing Processor:** $\langle \mathcal{D}, \mathcal{S}, \mathcal{K}, \mathcal{R} \rangle \overset{\mathcal{P}ol_0}{\rightsquigarrow} \langle \mathcal{D}', \mathcal{S}', \mathcal{K}', \mathcal{R} \rangle$ where in $\mathcal{D}'$, $\mathcal{S}'$, some $w \to t$ is replaced by all its narrowings

$$\langle \mathcal{D}, \mathcal{D}, \varnothing, \mathcal{R} \rangle \overset{\mathcal{P}ol_0}{\rightsquigarrow}^* \langle \mathcal{D}_1, \mathcal{D}_1, \varnothing, \mathcal{R}_1 \rangle \overset{\mathcal{P}ol_0}{\rightsquigarrow}^* \langle \mathcal{D}_2, \mathcal{D}_2, \varnothing, \mathcal{R}_1 \rangle \overset{\mathcal{P}ol_0}{\rightsquigarrow}^* \langle \mathcal{D}_3, \mathcal{D}_3, \varnothing, \mathcal{R}_2 \rangle$$

$\mathcal{R}_2$ :

$$\begin{aligned} \mathsf{gt}(0, k) &\to \mathsf{false} \\ \mathsf{gt}(\mathsf{s}(n), 0) &\to \mathsf{true} \\ \mathsf{gt}(\mathsf{s}(n), \mathsf{s}(k)) &\to \mathsf{gt}(n, k) \end{aligned}$$

$\mathcal{D}_3$ :

$$\begin{aligned} \mathsf{m}^\sharp(\mathsf{s}(n), 0) &\to \mathrm{COM}_2(\mathsf{if}^\sharp(\mathsf{true}, \mathsf{s}(n), 0), \mathsf{gt}^\sharp(\mathsf{s}(n), 0)) \\ \mathsf{m}^\sharp(\mathsf{s}(n), \mathsf{s}(k)) &\to \mathrm{COM}_2(\mathsf{if}^\sharp(\mathsf{gt}(n, k), \mathsf{s}(n), \mathsf{s}(k)), \mathsf{gt}^\sharp(\mathsf{s}(n), \mathsf{s}(k))) \\ \mathsf{if}^\sharp(\mathsf{true}, \mathsf{s}(n), y) &\to \mathrm{COM}_2(\mathsf{m}^\sharp(n, y), \mathsf{p}^\sharp(\mathsf{s}(n))) \qquad \mathsf{gt}^\sharp(\mathsf{s}(n), \mathsf{s}(k)) \to \mathrm{COM}_1(\mathsf{gt}^\sharp(n, k)) \end{aligned}$$

# Narrowing Processor

**Reduction Pair Processor:** $\langle \mathcal{D}, \mathcal{S}, \mathcal{K}, \mathcal{R} \rangle \overset{\mathcal{P}ol_m}{\leadsto} \langle \mathcal{D}, \mathcal{S} \setminus \mathcal{D}_\succ, \mathcal{K} \cup \mathcal{D}_\succ, \mathcal{R} \rangle$ where $m$ is the maximal degree of polynomials $[f^\sharp]$

**Polynomial Order**

- $[0] = [\mathsf{true}] = [\mathsf{false}] = [\mathsf{p}^\sharp](x) = 0, \quad [\mathsf{s}](x) = x + 2$
- $[\mathsf{gt}](x, y) = [\mathsf{gt}^\sharp](x, y) = x$
- $[\mathsf{m}^\sharp](x, y) = (x + 1)^2, \quad [\mathsf{if}^\sharp](x, y, z) = y^2$

$$\langle \mathcal{D}, \mathcal{D}, \varnothing, \mathcal{R} \rangle \overset{\mathcal{P}ol_0}{\leadsto}{}^{*} \langle \mathcal{D}_1, \mathcal{D}_1, \varnothing, \mathcal{R}_1 \rangle \overset{\mathcal{P}ol_0}{\leadsto}{}^{*} \langle \mathcal{D}_2, \mathcal{D}_2, \varnothing, \mathcal{R}_1 \rangle \overset{\mathcal{P}ol_0}{\leadsto}{}^{*} \langle \mathcal{D}_3, \mathcal{D}_3, \varnothing, \mathcal{R}_2 \rangle$$

$$\overset{\mathcal{P}ol_2}{\leadsto} \langle \mathcal{D}_3, \varnothing, \mathcal{D}_3, \mathcal{R}_2 \rangle$$

$\mathcal{R}_2$ :

$$\begin{aligned} \mathsf{gt}(0, k) &\succsim \mathsf{false} \\ \mathsf{gt}(\mathsf{s}(n), 0) &\succsim \mathsf{true} \\ \mathsf{gt}(\mathsf{s}(n), \mathsf{s}(k)) &\succsim \mathsf{gt}(n, k) \end{aligned}$$

$\mathcal{D}_3$ :

$$\begin{aligned} \mathsf{m}^\sharp(\mathsf{s}(n), 0) &\succ \textsc{Com}_2(\mathsf{if}^\sharp(\mathsf{true}, \mathsf{s}(n), 0), \mathsf{gt}^\sharp(\mathsf{s}(n), 0)) \\ \mathsf{m}^\sharp(\mathsf{s}(n), \mathsf{s}(k)) &\succ \textsc{Com}_2(\mathsf{if}^\sharp(\mathsf{gt}(n, k), \mathsf{s}(n), \mathsf{s}(k)), \mathsf{gt}^\sharp(\mathsf{s}(n), \mathsf{s}(k))) \\ \mathsf{if}^\sharp(\mathsf{true}, \mathsf{s}(n), y) &\succ \textsc{Com}_2(\mathsf{m}^\sharp(\mathsf{s}(n), y), \mathsf{p}^\sharp(\mathsf{s}(n))) \quad \mathsf{gt}^\sharp(\mathsf{s}(n), \mathsf{s}(k)) \succ \textsc{Com}_1(\mathsf{gt}^\sharp(n, k)) \end{aligned}$$

# Narrowing Processor

$$\begin{array}{lll} \langle \mathcal{D}, \mathcal{D}, \varnothing, \mathcal{R} \rangle & \overset{\mathcal{P}ol_0}{\rightsquigarrow}^* & \langle \mathcal{D}_1, \mathcal{D}_1, \varnothing, \mathcal{R}_1 \rangle \overset{\mathcal{P}ol_0}{\rightsquigarrow}^* \langle \mathcal{D}_2, \mathcal{D}_2, \varnothing, \mathcal{R}_1 \rangle \overset{\mathcal{P}ol_0}{\rightsquigarrow}^* \langle \mathcal{D}_3, \mathcal{D}_3, \varnothing, \mathcal{R}_2 \rangle \\ & \overset{\mathcal{P}ol_2}{\rightsquigarrow} & \langle \mathcal{D}_3, \varnothing, \mathcal{D}_3, \mathcal{R}_2 \rangle \end{array}$$

$$\iota_{\mathcal{R}} \leq \max(\mathcal{P}ol_0, \ldots, \mathcal{P}ol_0, \mathcal{P}ol_2) = \mathcal{P}ol_2$$

# DT Framework for Innermost Complexity Analysis

- *Direct* adaption of DP framework for termination analysis
- *Modular* combination of different techniques
- Experiments on 1323 TRSs from *Termination Problem Data Base*

AProVE: 618 examples with polynomial runtime
CaT: 447 examples with polynomial runtime
TCT: 385 examples with polynomial runtime

| | | CaT | | | | | |
|---|---|---|---|---|---|---|---|
| | | $\mathcal{P}ol_0$ | $\mathcal{P}ol_1$ | $\mathcal{P}ol_2$ | $\mathcal{P}ol_3$ | no result | $\sum$ |
| AProVE | $\mathcal{P}ol_0$ | - | 182 | - | - | 27 | 209 |
| | $\mathcal{P}ol_1$ | - | 187 | 7 | - | 76 | 270 |
| | $\mathcal{P}ol_2$ | - | 32 | 2 | - | 83 | 117 |
| | $\mathcal{P}ol_3$ | - | 6 | - | - | 16 | 22 |
| | no result | - | 27 | 3 | 1 | 674 | 705 |
| | $\sum$ | 0 | 434 | 12 | 1 | 876 | 1323 |

# DT Framework for Innermost Complexity Analysis

- *Direct* adaption of DP framework for termination analysis
- *Modular* combination of different techniques
- Experiments on 1323 TRSs from *Termination Problem Data Base*

AProVE: 618 examples with polynomial runtime
CaT: 447 examples with polynomial runtime
TCT: 385 examples with polynomial runtime

| | | TCT | | | | | |
|---|---|---|---|---|---|---|---|
| | | $\mathcal{P}ol_0$ | $\mathcal{P}ol_1$ | $\mathcal{P}ol_2$ | $\mathcal{P}ol_3$ | no result | $\sum$ |
| AProVE | $\mathcal{P}ol_0$ | 10 | 157 | - | - | 42 | 209 |
| | $\mathcal{P}ol_1$ | - | 152 | 1 | - | 117 | 270 |
| | $\mathcal{P}ol_2$ | - | 35 | - | - | 82 | 117 |
| | $\mathcal{P}ol_3$ | - | 5 | - | - | 17 | 22 |
| | no result | - | 22 | 3 | - | 680 | 705 |
| | $\sum$ | 10 | 371 | 4 | 0 | 938 | 1323 |

# Overview

## I. Termination of Term Rewriting

1. Termination of Term Rewrite Systems
2. Non-Termination of Term Rewrite Systems
3. Complexity of Term Rewrite Systems
4. Termination of Integer Term Rewrite Systems (RTA '09)

## II. Termination of Programs

1. Termination of Functional Programs (Haskell)
2. Termination of Logic Programs (Prolog)
3. Termination of Imperative Programs (Java)

# Termination of Programs

- direct approaches (e.g., Terminator for C-programs)
  - powerful for pre-defined data structures like integers
  - weak for algorithms on user-defined data structures

- transformational approaches via term rewriting (e.g., AProVE for Haskell, Prolog, Java)
  - powerful for algorithms on user-defined data structures (automatic generation of orders to compare arbitrary terms)
  - naive handling of pre-defined data structures (represent data objects by terms)

**Representing integers**

$$
\begin{aligned}
0 &\equiv \mathsf{pos}(0) \equiv \mathsf{neg}(0) \\
1 &\equiv \mathsf{pos}(\mathsf{s}(0)) \\
-1 &\equiv \mathsf{neg}(\mathsf{s}(0)) \\
1000 &\equiv \mathsf{pos}(\mathsf{s}(\mathsf{s}(\ldots \mathsf{s}(0)\ldots)))
\end{aligned}
$$

**Rules for pre-defined operations**

$$
\begin{aligned}
\mathsf{pos}(x) + \mathsf{neg}(y) &\to \mathsf{minus}(x, y) \\
\mathsf{neg}(x) + \mathsf{pos}(y) &\to \mathsf{minus}(y, x) \\
\mathsf{pos}(x) + \mathsf{pos}(y) &\to \mathsf{pos}(\mathsf{plus}(x, y)) \\
\mathsf{neg}(x) + \mathsf{neg}(y) &\to \mathsf{neg}(\mathsf{plus}(x, y)) \\
\mathsf{minus}(x, 0) &\to \mathsf{pos}(x) \\
\mathsf{minus}(0, y) &\to \mathsf{neg}(y) \\
\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{minus}(x, y)
\end{aligned}
$$

# Termination of Programs

- direct approaches (e.g., Terminator for C-programs)
  - powerful for pre-defined data structures like integers
  - weak for algorithms on user-defined data structures

- transformational approaches via term rewriting
  (e.g., AProVE for Haskell, Prolog, Java)
  - powerful for algorithms on user-defined data structures
    (automatic generation of orders to compare arbitrary terms)
  - naive handling of pre-defined data structures
    (represent data objects by terms)

Goal:

- integrate pre-defined data structures like $\mathbb{Z}$ into term rewriting
- develop method to prove termination of integer TRSs
  $\Rightarrow$ adapt DP framework to ITRSs
- for algorithms on integers: as powerful as direct techniques
- for user-defined data structures: as powerful as DP framework

# Integer Term Rewriting

- $\mathcal{F}_{int}$: pre-defined symbols
  - $\mathbb{Z} = \{0, 1, -1, 2, -2, \dots\}$
  - $\mathbb{B} = \{\mathsf{true}, \mathsf{false}\}$
  - $+, -, *, /, \%$
  - $>, \geq, <, \leq, ==, !=$
  - $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$
- $\mathcal{PD}$: pre-defined rules

  $2 * 21 \to 42$ $\quad$ $42 \geq 23 \to \mathsf{true}$

  $\mathsf{true} \wedge \mathsf{false} \to \mathsf{false}$ $\quad$ $\dots$

  $\Rightarrow$ pre-defined operations only evaluated if all arguments are from $\mathbb{Z}$ or $\mathbb{B}$

- ITRS $\mathcal{R}$: finite TRS
  - no pre-defined symbols except $\mathbb{Z}$ and $\mathbb{B}$ in lhs
  - lhs $\notin \mathbb{Z} \cup \mathbb{B}$
  - rewrite relation $\hookrightarrow_{\mathcal{R}}$ defined as $\stackrel{\mathrm{i}}{\to}_{\mathcal{R} \cup \mathcal{PD}}$

**Example ITRS computing $\sum_{i=y}^{x} i$**

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

# Integer Term Rewriting

$$
\begin{array}{llllll}
\underline{\mathsf{sum}(1,1)} & \hookrightarrow_{\mathcal{R}} & \mathsf{sif}(\underline{1 \geq 1},1,1) & \hookrightarrow_{\mathcal{R}} & \underline{\mathsf{sif}(\mathsf{true},1,1)} \\
 & \hookrightarrow_{\mathcal{R}} & 1 + \mathsf{sum}(1,\underline{1+1}) & \hookrightarrow_{\mathcal{R}} & 1 + \underline{\mathsf{sum}(1,2)} \\
 & \hookrightarrow_{\mathcal{R}} & 1 + \mathsf{sif}(\underline{1 \geq 2},1,2) & \hookrightarrow_{\mathcal{R}} & 1 + \underline{\mathsf{sif}(\mathsf{false},1,2)} \\
 & \hookrightarrow_{\mathcal{R}} & \underline{1+0} & \hookrightarrow_{\mathcal{R}} & 1
\end{array}
$$

- ITRS $\mathcal{R}$: finite TRS
  - no pre-defined symbols except $\mathbb{Z}$ and $\mathbb{B}$ in lhs
  - lhs $\notin \mathbb{Z} \cup \mathbb{B}$
  - rewrite relation $\hookrightarrow_{\mathcal{R}}$ defined as $\xrightarrow{\mathsf{i}}_{\mathcal{R} \cup \mathcal{PD}}$

**Example ITRS computing $\sum_{i=y}^{x} i$**

$$
\begin{array}{rcl}
\mathsf{sum}(x,y) & \to & \mathsf{sif}(x \geq y, x, y) \\
\mathsf{sif}(\mathsf{true},x,y) & \to & y + \mathsf{sum}(x, y+1) \\
\mathsf{sif}(\mathsf{false},x,y) & \to & 0
\end{array}
$$

# Integer Term Rewriting

- Goal: prove innermost termination of $\mathcal{R} \cup \mathcal{PD}$ automatically

  Problem: $\mathcal{PD}$ is infinite

  Solution: handle $\mathcal{PD}$ implicitly by integrating it into the processors of the DP framework

- ITRS $\mathcal{R}$: finite TRS
  - no pre-defined symbols except $\mathbb{Z}$ and $\mathbb{B}$ in lhs
  - lhs $\notin \mathbb{Z} \cup \mathbb{B}$
  - rewrite relation $\hookrightarrow_{\mathcal{R}}$ defined as $\xrightarrow{\mathsf{i}}_{\mathcal{R} \cup \mathcal{PD}}$

**Example ITRS computing $\sum_{i=y}^{x} i$**

$$
\begin{array}{rcl}
\mathsf{sum}(x, y) & \to & \mathsf{sif}(x \geq y, x, y) \\
\mathsf{sif}(\mathsf{true}, x, y) & \to & y + \mathsf{sum}(x, y + 1) \\
\mathsf{sif}(\mathsf{false}, x, y) & \to & 0
\end{array}
$$

# Integer Dependency Pair Framework

- Defined symbols of TRS $\mathcal{R}$: roots of left-hand sides of $\mathcal{R}$
- Dependency pairs of TRS $\mathcal{R}$:
  if $f(s_1, \ldots, s_n) \to \ldots g(t_1, \ldots, t_m) \ldots \in \mathcal{R}$ and $g$ is defined,
  then $F(s_1, \ldots, s_n) \to G(t_1, \ldots, t_m) \in DP(\mathcal{R})$

**Example TRS**

$$\mathsf{sum}(x, y) \to \mathsf{sif}(x \geq y, x, y)$$
$$\mathsf{sif}(\mathsf{true}, x, y) \to y + \mathsf{sum}(x, y + 1)$$
$$\mathsf{sif}(\mathsf{false}, x, y) \to 0$$

**DPs**

$$\mathsf{SUM}(x, y) \to \mathsf{SIF}(x \geq y, x, y)$$
$$\mathsf{SIF}(\mathsf{true}, x, y) \to \mathsf{SUM}(x, y + 1)$$

- $\mathcal{P}$-chain for DPs $\mathcal{P}$ and TRS $\mathcal{R}$: $s_1 \to t_1$, $s_2 \to t_2, \ldots$ where
  - $s_i \to t_i \in \mathcal{P}$
  - $t_i\sigma \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} s_{i+1}\sigma$
  - $s_i\sigma$ in normal form w.r.t. $\xrightarrow{\mathrm{i}}_{\mathcal{R}}$
- Theorem TRS $\mathcal{R}$ terminating iff there is no infinite $DP(\mathcal{R})$-chain

# Integer Dependency Pair Framework

- Defined symbols of ITRS $\mathcal{R}$: roots of left-hand sides of $\mathcal{R} \cup \mathcal{PD}$ including $+, -, >, \geq, \neg, \wedge, \vee, \ldots$
- Dependency pairs of ITRS $\mathcal{R}$:
  if $f(s_1, \ldots, s_n) \to \ldots g(t_1, \ldots, t_m) \ldots \in \mathcal{R}$ and $g \notin \mathcal{F}_{int}$ is defined,
  then $F(s_1, \ldots, s_n) \to G(t_1, \ldots, t_m) \in DP(\mathcal{R})$

**Example ITRS**

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

**DPs**

$$\begin{aligned} \mathsf{SUM}(x, y) &\to \mathsf{SIF}(x \geq y, x, y) \\ \mathsf{SIF}(\mathsf{true}, x, y) &\to \mathsf{SUM}(x, y + 1) \end{aligned}$$

- $\mathcal{P}$-chain for DPs $\mathcal{P}$ and ITRS $\mathcal{R}$: $s_1 \to t_1$, $s_2 \to t_2, \ldots$ where
  - $s_i \to t_i \in \mathcal{P}$
  - $t_i\sigma \hookrightarrow_{\mathcal{R}}^* s_{i+1}\sigma$
  - $s_i\sigma$ in normal form w.r.t. $\hookrightarrow_{\mathcal{R}}$
- Theorem ITRS $\mathcal{R}$ terminating iff there is no infinite $DP(\mathcal{R})$-chain

# Integer Dependency Pair Framework

## Chain

$\mathsf{SUM}(x, y) \to \mathsf{SIF}(x \geq y, x, y)$, $\quad \mathsf{SIF}(\mathsf{true}, x, y) \to \mathsf{SUM}(x, y+1)$ is chain for $\sigma(x) = \sigma(y) = 1$

$\mathsf{SIF}(1 \geq 1, 1, 1) \hookrightarrow^*_{\mathcal{R}} \mathsf{SIF}(\mathsf{true}, 1, 1)$

## Example ITRS

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

## DPs

$$\begin{aligned} \mathsf{SUM}(x, y) &\to \mathsf{SIF}(x \geq y, x, y) \\ \mathsf{SIF}(\mathsf{true}, x, y) &\to \mathsf{SUM}(x, y+1) \end{aligned}$$

- $\mathcal{P}$-chain for DPs $\mathcal{P}$ and ITRS $\mathcal{R}$: $s_1 \to t_1$, $s_2 \to t_2, \ldots$ where
  - $s_i \to t_i \in \mathcal{P}$
  - $t_i\sigma \hookrightarrow^*_{\mathcal{R}} s_{i+1}\sigma$
  - $s_i\sigma$ in normal form w.r.t. $\hookrightarrow_{\mathcal{R}}$
- Theorem ITRS $\mathcal{R}$ terminating iff there is no infinite $DP(\mathcal{R})$-chain

# Integer Dependency Pair Framework

- DP processors $\mathit{Proc}(\mathcal{P}) = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$
  - transform problem into simpler sub-problems
  - soundness: if there are no infinite $\mathcal{P}_1$-, ..., $\mathcal{P}_n$-chains, then there is no infinite $\mathcal{P}$-chain
  - start with initial problem $DP(\mathcal{R})$, apply processors repeatedly until all problems are solved

- numerous DP processors developed for TRSs
  - many processors only rely on DPs and on defined symbols
    $\Rightarrow$ adaption to ITRSs straightforward
  - reduction pair processor relies on DPs and on *rules*
    $\Rightarrow$ adaption to ITRSs problematic, since $\mathcal{PD}$ has infinitely many rules

Goal: adapt reduction pair processor to ITRSs

# Reduction Pair Processor

**Pre-defined rules $\mathcal{PD}$ and ITRS $\mathcal{R}$**

$$
\begin{aligned}
0 + 0 &\to 0 \\
-1 + 3 &\to 2 \\
&\dots \\
\mathsf{sum}(x, y) &\to \mathsf{sif}(x \geq y, x, y) \\
\mathsf{sif}(\mathsf{true}, x, y) &\to y + \mathsf{sum}(x, y + 1) \\
\mathsf{sif}(\mathsf{false}, x, y) &\to 0
\end{aligned}
$$

**DPs $\mathcal{P}$**

$$
\begin{aligned}
\mathsf{SUM}(x, y) &\to \mathsf{SIF}(x \geq y, x, y) \\
\mathsf{SIF}(\mathsf{true}, x, y) &\to \mathsf{SUM}(x, y + 1)
\end{aligned}
$$

- Generate constraints such that infinite chain leads to infinite decrease

$$
\begin{array}{ccccccccccccc}
s_1 & \to & t_1, & & s_2 & \to & t_2, & & s_3 & \to & t_3, & & \dots \\
s_1 & \underset{(\sim)}{\succ} & t_1 & \succsim & s_2 & \underset{(\sim)}{\succ} & t_2 & \succsim & s_3 & \underset{(\sim)}{\succ} & t_3 & \succsim & \dots
\end{array}
$$

- Reduction pair processor: $Proc(\mathcal{P}) = \{\mathcal{P} \setminus \succ\}$ if
  - $\ell \succsim r$ for all usable rules $\ell \to r$ in $\mathcal{R} \cup \mathcal{PD}$
  - $s \succ t$ or $s \succsim t$ for all $s \to t$ in $\mathcal{P}$

- Usable rules: rules that can reduce terms of $\mathcal{P}$'s right-hand sides when instantiating their variables with normal forms

# Reduction Pair Processor

**Usable rules in $\mathcal{R} \cup \mathcal{PD}$**

$$\begin{aligned} 0 + 0 &\succsim 0 \\ -1 + 3 &\succsim 2 \\ &\cdots \end{aligned}$$

**DPs $\mathcal{P}$**

$$\begin{aligned} \mathsf{SUM}(x, y) &\overset{(\succ)}{\succsim} \mathsf{SIF}(x \geq y, x, y) \\ \mathsf{SIF}(\mathsf{true}, x, y) &\overset{(\succ)}{\succsim} \mathsf{SUM}(x, y + 1) \\ \mathsf{SUM}(x, y) &\overset{(\succ)}{\succsim} \mathsf{c} \\ \mathsf{SIF}(\mathsf{true}, x, y) &\overset{(\succ)}{\succsim} \mathsf{c} \end{aligned}$$

- Problem: infinitely many constraints $\ell \succsim r$ since $\mathcal{PD}$ is infinite
  Solution: consider $\mathcal{PD}$ implicitly $\Rightarrow$ fix interpretation for $\mathcal{F}_{int}$

- Reduction pair processor: $Proc(\mathcal{P}) = \{\mathcal{P} \setminus \succ, \mathcal{P} \setminus \mathcal{P}_{bound}\}$ if
  - $\ell \succsim r$ for all usable rules $\ell \to r$ in $\mathcal{R} \cup \mathcal{PD}$
  - $s \succ t$ or $s \succsim t$ for all $s \to t$ in $\mathcal{P}$

  where $\mathcal{P}_{bound} = \{s \to t \in \mathcal{P} \mid s \succsim \mathsf{c}\}$

- search for integer max-polynomial interpretation *Pol* satisfying constraints

# Reduction Pair Processor

**Usable rules in $\mathcal{R}$**

**DPs $\mathcal{P}$**

$$\begin{array}{rcl} \mathsf{SUM}(x,y) & \overset{\succ}{(\succsim)} & \mathsf{SIF}(x \geq y, x, y) \\ \mathsf{SIF}(\mathsf{true}, x, y) & \overset{\succ}{(\succsim)} & \mathsf{SUM}(x, y+1) \\ \mathsf{SUM}(x,y) & \overset{\succ}{(\succsim)} & \mathsf{c} \\ \mathsf{SIF}(\mathsf{true}, x, y) & \overset{\succ}{(\succsim)} & \mathsf{c} \end{array}$$

- choose $\mathsf{SUM}_{Pol}(x,y) = x - y \qquad \mathsf{SIF}_{Pol}(x,y,z) = y - z$

  But: $\mathsf{SUM}(x,y) \succsim \mathsf{c}$, $\mathsf{SIF}(\mathsf{true},x,y) \succsim \mathsf{c}$ not satisfied for any $\mathsf{c}_{Pol}$

- Reduction pair processor: $Proc(\mathcal{P}) = \{\mathcal{P} \setminus \succ, \mathcal{P} \setminus \mathcal{P}_{bound}\}$ if
  - $\ell \succsim r$ for all usable rules $\ell \to r$ in $\mathcal{R}$
  - $s \succ t$ or $s \succsim t$ for all $s \to t$ in $\mathcal{P}$

  where $\mathcal{P}_{bound} = \{s \to t \in \mathcal{P} \mid s \succsim \mathsf{c}\}$

- I-interpretation: $0_{Pol} = 0, \quad 1_{Pol} = 1, \quad -1_{Pol} = -1, \; \ldots,$
  $x +_{Pol} y = x + y, \quad x -_{Pol} y = x - y, \quad x *_{Pol} y = x * y, \; \ldots$

# Reduction Pair Processor with Conditional Constraints

- weaken constraints: do not require them for *all* $x$ and $y$, but only for those instantiations $\sigma$ used in chains
- require $\mathsf{SUM}(x, y) \succsim \mathsf{c}$ only for $\sigma$ where $\mathsf{SIF}(x \geq y, x, y)\sigma$ reduces to $\mathsf{SUM}(x', y')\sigma$ or $\mathsf{SIF}(\mathsf{true}, x', y')\sigma$

**Constraints**

$$\mathsf{SIF}(x \geq y, x, y) = \mathsf{SUM}(x', y') \Rightarrow \mathsf{SUM}(x, y) \succsim \mathsf{c}$$
$$\mathsf{SIF}(x \geq y, x, y) = \mathsf{SIF}(\mathsf{true}, x', y') \Rightarrow \mathsf{SUM}(x, y) \succsim \mathsf{c}$$

**DPs**

$$\mathsf{SUM}(x, y) \to \mathsf{SIF}(x \geq y, x, y)$$
$$\mathsf{SIF}(\mathsf{true}, x, y) \to \mathsf{SUM}(x, y + 1)$$

# Reduction Pair Processor with Conditional Constraints

**Rules to simplify conditional constraints**

1. **Constructor and Different Function Symbol**

$$\frac{f(s_1, \ldots, s_n) = g(t_1, \ldots, t_m) \land \varphi \;\Rightarrow\; \psi}{TRUE}$$

if $f$ is a constructor and $f \neq g$

**Constraints**

$$\mathsf{SIF}(x \geq y, x, y) = \mathsf{SIF}(\mathsf{true}, x', y') \;\Rightarrow\; \mathsf{SUM}(x, y) \;\succsim\; \mathsf{c}$$

# Reduction Pair Processor with Conditional Constraints

**Rules to simplify conditional constraints**

1. **Constructor and Different Function Symbol**

$$\frac{f(s_1, \ldots, s_n) = g(t_1, \ldots, t_m) \wedge \varphi \;\Rightarrow\; \psi}{TRUE}$$

if $f$ is a constructor and $f \neq g$

2. **Same Constructors on Both Sides**

$$\frac{f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n) \wedge \varphi \;\Rightarrow\; \psi}{s_1 = t_1 \wedge \ldots \wedge s_n = t_n \wedge \varphi \;\Rightarrow\; \psi}$$

if $f$ is a constructor

**Constraints**

$$x \geq y = \mathsf{true} \;\wedge\; x = x' \;\wedge\; y = y' \;\Rightarrow\; \mathsf{SUM}(x, y) \;\succsim\; \mathsf{c}$$

# Reduction Pair Processor with Conditional Constraints

**Rules to simplify conditional constraints**

1. **Constructor and Different Function Symbol**

$$\frac{f(s_1, \ldots, s_n) = g(t_1, \ldots, t_m) \wedge \varphi \;\Rightarrow\; \psi}{\mathit{TRUE}}$$

if $f$ is a constructor and $f \neq g$

2. **Same Constructors on Both Sides**

$$\frac{f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n) \wedge \varphi \;\Rightarrow\; \psi}{s_1 = t_1 \wedge \ldots \wedge s_n = t_n \wedge \varphi \;\Rightarrow\; \psi}$$

if $f$ is a constructor

3. **Lift Pre-Defined Symbols**

$$\frac{s \geq t = \mathrm{true} \wedge \varphi \;\Rightarrow\; \psi}{(s \succsim t \qquad \wedge \varphi \;\Rightarrow\; \psi) \wedge \ell \succsim r \text{ for all usable rules } \ell \to r}$$

**Constraints**

$x \succsim y \qquad \Rightarrow \mathrm{SUM}(x, y) \succsim \mathrm{c}$

# Reduction Pair Processor with Conditional Constraints

## Constraints

$$
\begin{array}{rcrcl}
 & & \mathsf{SUM}(x,y) & \underset{(\sim)}{\succ} & \mathsf{SIF}(x \geq y, x, y) \\
 & & \mathsf{SIF}(\mathsf{true}, x, y) & \underset{(\sim)}{\succ} & \mathsf{SUM}(x, y+1) \\
\mathsf{SIF}(x \geq y, x, y) = \mathsf{SUM}(x', y') & \Rightarrow & \mathsf{SUM}(x,y) & \succsim & \mathsf{c} \\
\mathsf{SIF}(x \geq y, x, y) = \mathsf{SIF}(\mathsf{true}, x', y') & \Rightarrow & \mathsf{SUM}(x,y) & \succsim & \mathsf{c}
\end{array}
$$

- Reduction pair processor: $Proc(\mathcal{P}) = \{\mathcal{P} \setminus \succ, \mathcal{P} \setminus \mathcal{P}_{bound}\}$ if
  - $\ell \succsim r$ for all usable rules $\ell \to r$ in $\mathcal{R}$
  - $s \succ t$ or $s \succsim t$ for all $s \to t$ in $\mathcal{P}$

  where $\mathcal{P}_{bound} = \{s \to t \in \mathcal{P} \mid s \succsim \mathsf{c}\}$

# Reduction Pair Processor with Conditional Constraints

## Constraints

$$\mathsf{SUM}(x,y) \underset{(\sim)}{\succsim} \mathsf{SIF}(x \geq y, x, y)$$
$$\mathsf{SIF}(\mathsf{true}, x, y) \underset{(\sim)}{\succsim} \mathsf{SUM}(x, y+1)$$

$$x \succsim y \;\Rightarrow\; \mathsf{SUM}(x,y) \succsim \mathsf{c}$$

- Reduction pair processor: $Proc(\mathcal{P}) = \{\mathcal{P} \setminus \succ, \mathcal{P} \setminus \mathcal{P}_{bound}\}$ if
  - $\ell \succsim r$ for all usable rules $\ell \to r$ in $\mathcal{R}$
  - $s \succ t$ or $s \succsim t$ for all $s \to t$ in $\mathcal{P}$

  where $\mathcal{P}_{bound} = \{s \to t \in \mathcal{P} \mid s \succsim \mathsf{c}\}$

- I-interpretation: $\mathsf{SUM}_{Pol}(x,y) = x - y$ $\quad \mathsf{SIF}_{Pol}(x,y,z) = y - z$ $\quad \mathsf{c}_{Pol} = 0$

# Reduction Pair Processor with Conditional Constraints

## Constraints

$$\begin{aligned}
\mathsf{SUM}(x,y) &\underset{(\sim)}{\succ} \mathsf{SIF}(x \geq y, x, y)\\
\mathsf{SIF}(\mathsf{true}, x, y) &\succ \mathsf{SUM}(x, y+1)
\end{aligned}$$

$$x \succsim y \;\Rightarrow\; \mathsf{SUM}(x,y) \succsim \mathsf{c}$$

- Reduction pair processor: $Proc(\mathcal{P}) = \{\mathcal{P}\setminus \succ, \mathcal{P} \setminus \mathcal{P}_{bound}\}$ if
  - $\ell \succsim r$ for all usable rules $\ell \to r$ in $\mathcal{R}$
  - $s \succ t$ or $s \succsim t$ for all $s \to t$ in $\mathcal{P}$

  where $\mathcal{P}_{bound} = \{s \to t \in \mathcal{P} \mid s \succsim \mathsf{c}\}$

- I-interpretation: $\mathsf{SUM}_{Pol}(x,y) = x - y \quad \mathsf{SIF}_{Pol}(x,y,z) = y - z \quad \mathsf{c}_{Pol} = 0$

- *Proc* transforms initial problem into two separate problems

$$\begin{aligned}
\mathcal{P}\setminus \succ: &\quad \mathsf{SUM}(x,y) \to \mathsf{SIF}(x \geq y, x, y)\\
\mathcal{P} \setminus \mathcal{P}_{bound}: &\quad \mathsf{SIF}(\mathsf{true}, x, y) \to \mathsf{SUM}(x, y+1)
\end{aligned}$$

$\Rightarrow$ both can easily be solved separately

$\Rightarrow$ termination easy if one can generate I-interpretation automatically

# Generating I-Interpretations

- *abstract* I-interpretation: $\mathsf{SUM}_{Pol}(x, y) = a_0 + a_1\, x + a_2\, y \qquad \mathsf{c}_{Pol} = c_0$

**Conditional constraint (without =)**

$$x \succsim y \quad \Rightarrow \quad \mathsf{SUM}(x, y) \succsim \mathsf{c}$$

# Generating I-Interpretations

- *abstract* I-interpretation: $\mathrm{SUM}_{Pol}(x, y) = a_0 + a_1\, x + a_2\, y \qquad \mathsf{c}_{Pol} = c_0$

**Inequality constraint**

$$\forall x \in \mathbb{Z},\ y \in \mathbb{Z} \quad (\ x \ \geq \ y \quad \Rightarrow \quad a_0 + a_1 \quad x \quad + a_2\, y \ \geq \ c_0\ )$$

# Generating I-Interpretations

## Rules to simplify inequality constraints

Goal: remove $\forall$, $\mathbb{Z}$, conditions $\Rightarrow$ Diophantine constraints $\Rightarrow$ SAT solving

1. **Eliminate Conditions**

$$\frac{\forall x \in \mathbb{Z}, \ldots \quad (x \geq p \wedge \varphi \Rightarrow \psi)}{\forall z \in \mathbb{N}, \ldots \quad (\varphi[x/p+z] \Rightarrow \psi[x/p+z])}$$

if $x$ does not occur in the polynomial $p$

### Inequality constraint

$\forall y \in \mathbb{Z},\ z \in \mathbb{N}$ $\quad a_0 + a_1 (\ y + z) + a_2\, y \ \geq\ c_0$

# Generating I-Interpretations

**Rules to simplify inequality constraints**

Goal: remove $\forall$, $\mathbb{Z}$, conditions $\Rightarrow$ Diophantine constraints $\Rightarrow$ SAT solving

1. **Eliminate Conditions**

$$\frac{\forall x \in \mathbb{Z}, \ldots \quad (x \geq p \wedge \varphi \Rightarrow \psi)}{\forall z \in \mathbb{N}, \ldots \quad (\varphi[x/p+z] \Rightarrow \psi[x/p+z])}$$

if $x$ does not occur in the polynomial $p$

2. **Split**

$$\frac{\forall y \in \mathbb{Z} \quad \varphi}{\forall y \in \mathbb{N} \quad \varphi \qquad \wedge \qquad \forall y \in \mathbb{N} \quad \varphi[y/-y]}$$

**Inequality constraint**

$$\forall y \in \mathbb{N},\ z \in \mathbb{N} \qquad a_0 + a_1\,(\ \ y + z) + a_2\, y \ \geq\ c_0$$
$$\forall y \in \mathbb{N},\ z \in \mathbb{N} \qquad a_0 + a_1\,(-y + z) - a_2\, y \ \geq\ c_0$$

# Generating I-Interpretations

**Rules to simplify inequality constraints**

Goal: remove $\forall$, $\mathbb{Z}$, conditions $\Rightarrow$ Diophantine constraints $\Rightarrow$ SAT solving

1. **Eliminate Conditions**

$$\frac{\forall x \in \mathbb{Z}, \ldots \quad (x \geq p \wedge \varphi \Rightarrow \psi)}{\forall z \in \mathbb{N}, \ldots \quad (\varphi[x/p+z] \Rightarrow \psi[x/p+z])}$$

if $x$ does not occur in the polynomial $p$

2. **Split**

$$\frac{\forall y \in \mathbb{Z} \quad \varphi}{\forall y \in \mathbb{N} \quad \varphi \qquad \wedge \qquad \forall y \in \mathbb{N} \quad \varphi[y/-y]}$$

**Inequality constraint**

$$\forall y \in \mathbb{N},\ z \in \mathbb{N} \qquad (a_1 + a_2)\, y + a_1\, z + (a_0 - c_0)\ \geq\ 0$$

# Generating I-Interpretations

**Rules to simplify inequality constraints**

Goal: remove $\forall$, $\mathbb{Z}$, conditions $\Rightarrow$ Diophantine constraints $\Rightarrow$ SAT solving

1. **Eliminate Conditions**

$$\frac{\forall x \in \mathbb{Z}, \dots \quad (x \geq p \land \varphi \Rightarrow \psi)}{\forall z \in \mathbb{N}, \dots \quad (\varphi[x/p+z] \Rightarrow \psi[x/p+z])}$$

if $x$ does not occur in the polynomial $p$

2. **Split**

$$\frac{\forall y \in \mathbb{Z} \quad \varphi}{\forall y \in \mathbb{N} \quad \varphi \qquad \land \qquad \forall y \in \mathbb{N} \quad \varphi[y/-y]}$$

3. **Eliminate Universally Quantified Variables**

$$\frac{\forall x_i \in \mathbb{N} \quad p_1\, x_1^{e_{11}} \dots x_m^{e_{m1}} + \dots + p_k\, x_1^{e_{1k}} \dots x_m^{e_{mk}} \geq 0}{p_1 \geq 0 \ \land \dots \land \ p_k \geq 0}$$

if the $p_j$ do not contain variables

**Inequality constraint**

$$a_1 + a_2 \geq 0 \quad \land \quad a_1 \geq 0 \quad \land \quad a_0 - c_0 \geq 0$$

Solution: $a_0 = 0 \quad a_1 = 1 \quad a_2 = -1 \quad c_0 = 0$

# Generating I-Interpretations

*abstract* I-interpretation: $\mathsf{SUM}_{Pol}(x,y) = a_0 + a_1\,x + a_2\,y$ $\quad \mathsf{c}_{Pol} = c_0$

*actual* I-interpretation: $\mathsf{SUM}_{Pol}(x,y) = x - y$ $\quad \mathsf{c}_{Pol} = 0$

**Inequality constraint**

$$a_1 + a_2 \geq 0 \quad \wedge \quad a_1 \geq 0 \quad \wedge \quad a_0 - c_0 \geq 0$$

Solution: $a_0 = 0 \quad a_1 = 1 \quad a_2 = -1 \quad c_0 = 0$

# Proving Termination of Integer Term Rewriting

- ITRSs: TRSs with built-in integers, adapted DP framework to ITRSs
- also suitable for ITRSs with large numbers
  $\mathsf{f}(\mathsf{true},\ x) \ \to\ \mathsf{f}(\mathsf{ack}(10,10) \geq x,\ x+1)$
- implemented in AProVE and evaluated on collection of 117 ITRSs
  - examples from TPDB and rewriting papers adapted to integers
  - examples from termination of imperative programming

| | YES | MAYBE | TIMEOUT |
|---|---|---|---|
| AProVE Integer | 104 (avg. 4.9 s) | 0 | 13 |
| AProVE old | 24 (avg. 7.2 s) | 6 (avg. 0.9 s) | 87 |
| $\mathsf{T_TT_2}$ | 6 (avg. 3.6 s) | 110 (avg. 4.9 s) | 1 |

$\Rightarrow$ enormous benefit of built-in integers

# Automated Termination Analysis

Jürgen Giesl

LuFG Informatik 2, RWTH Aachen University, Germany

VTSA '12, Saarbrücken, Germany

# Overview

## I. Termination of Term Rewriting

1. Termination of Term Rewrite Systems
2. Non-Termination of Term Rewrite Systems
3. Complexity of Term Rewrite Systems
4. Termination of Integer Term Rewrite Systems

## II. Termination of Programs

1. Termination of Functional Programs (Haskell) (ACM TOPLAS '11)
2. Termination of Logic Programs (Prolog)
3. Termination of Imperative Programs (Java)

# Automated Termination Tools for TRSs

- AProVE *(Aachen)*
- CARIBOO *(Nancy)*
- CiME *(Orsay)*
- Jambox *(Amsterdam)*
- Matchbox *(Leipzig)*
- MU-TERM *(Valencia)*
- MultumNonMulta *(Kassel)*
- TEPARLA *(Eindhoven)*
- Termptation *(Barcelona)*
- TORPA *(Eindhoven)*
- TPA *(Eindhoven)*
- TTT *(Innsbruck)*
- VMTL *(Vienna)*

- Annual *International Competition of Termination Tools*
- well-developed field
- active research
- powerful techniques & tools
- **But:** What about application in practice?
- **Goal:** TRS-techniques for programming languages

# Termination of Functional Programs

- first-order languages with strict evaluation strategy
  *(Walther, 94), (Giesl, 95), (Lee, Jones, Ben-Amram, 01)*
- ensuring termination (e.g., by typing)
  *(Telford & Turner, 00), (Xi, 02), (Abel, 04), (Barthe et al, 04) etc.*
- outermost termination of untyped first-order rewriting
  *(Fissore, Gnaedig, Kirchner, 02)*
- automated technique for small HASKELL-like language
  *(Panitz & Schmidt-Schauss, 97)*

- do not work on full existing languages
- no use of TRS-techniques (stand-alone methods)

# Termination of Functional Programs

- first-order languages with strict evaluation strategy
  *(Walther, 94), (Giesl, 95), (Lee, Jones, Ben-Amram, 01)*
- ensuring termination (e.g., by typing)
  *(Telford & Turner, 00), (Xi, 02), (Abel, 04), (Barthe et al, 04) etc.*
- outermost termination of untyped first-order rewriting
  *(Fissore, Gnaedig, Kirchner, 02)*
- automated technique for small HASKELL-like language
  *(Panitz & Schmidt-Schauss, 97)*

- **new approach to use TRS-techniques for termination of HASKELL**
- based on *(Panitz & Schmidt-Schauss, 97)*, but:
  - works on full HASKELL-language
  - allows to integrate modern TRS-techniques and TRS-tools

# HASKELL

- one of the most popular functional languages
- using TRS-techniques for HASKELL is challenging:
  - HASKELL has a lazy evaluation strategy.
    For TRSs, one proves termination of *all* 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.
  - HASKELL is a higher-order language.
    Most automatic TRS-methods only handle *first-order* rewriting.

# Syntax of HASKELL

## Data Structures

- data Nats = Z | S Nats — type constructor: Nats of arity 0
  data constructors: Z :: Nats
  S :: Nats $\to$ Nats
- data List $a$ = Nil | Cons $a$ (List $a$) — type constructor: List of arity 1
  data constructors: Nil :: List $a$
  Cons :: $a$ $\to$ (List $a$) $\to$ (List $a$)

## Terms (well-typed)

- Variables: $x, y, \ldots$
- Function Symbols: constructors (Z, S, Nil, Cons) & defined (from, take)
- Applications $(t_1 \ t_2)$
  - S Z represents number 1
  - Cons $x$ Nil $\equiv$ (Cons $x$) Nil represents $[x]$

# Syntax of HASKELL

## Data Structures

- data Nats = Z | S Nats — type constructor: Nats of arity 0
  data constructors: Z :: Nats
  S :: Nats $\rightarrow$ Nats
- data List $a$ = Nil | Cons $a$ (List $a$) — type constructor: List of arity 1
  data constructors: Nil :: List $a$
  Cons :: $a \rightarrow$ (List $a$) $\rightarrow$ (List $a$)

## Types

- Type Variables: $a, b, \ldots$
- Applications of type constructors to types: List Nats, $a \rightarrow$ (List $a$), ...

| | | |
|---|---|---|
| S Z | has type | Nats |
| Cons $x$ Nil | has type | List $a$ |

# Syntax of HASKELL

## Function Declarations (general)

$$f\ \ell_1 \dots \ell_n = r$$

- $f$ is *defined* function symbol
- $n$ is *arity* of $f$
- $r$ is arbitrary term
- $\ell_1 \dots \ell_n$ are linear *patterns* (terms from constructors and variables)

## Function Declarations (example)

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

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

$$\mathsf{from} \ :: \ \mathsf{Nats} \to \mathsf{List\ Nats}$$

$$\mathsf{take} \ :: \ \mathsf{Nats} \to (\mathsf{List}\ a) \to (\mathsf{List}\ a)$$

$$\mathsf{from}\ x \ \equiv \ [x, x+1, x+2, \dots]$$

$$\mathsf{take}\ n\ [x_1, \dots, x_n, \dots] \ \equiv \ [x_1, \dots, x_n]$$

# Syntax of HASKELL

**Extension of our approach for**

- type classes
- built-in data structures

**All other HASKELL-constructs:** eliminated by automatic transformation

- Lambda Abstractions

replace $\backslash\ \ m \to \mathsf{take}\, u\, (\mathsf{from}\, m)$
by $\mathsf{f}\ u$
where $\mathsf{f}\ u\ m = \mathsf{take}\, u\, (\mathsf{from}\, m)$

# Syntax of HASKELL

**Extension of our approach for**

- type classes
- built-in data structures

**All other HASKELL-constructs:** eliminated by automatic transformation

- Lambda Abstractions

  replace $\backslash\, t_1 \ldots t_n \rightarrow t$ with free variables $x_1, \ldots, x_m$
  by $\mathsf{f}\, x_1 \ldots x_m$
  where $\mathsf{f}\, x_1 \ldots x_m\, t_1 \ldots t_n = t$
- Conditions
- Local Declarations
- ...

# Semantics and Termination of HASKELL

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

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

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

$$\begin{array}{ll} & \mathsf{from}\ \mathsf{Z} \\ \to_{\mathsf{H}} & \mathsf{Cons}\ \mathsf{Z}\ (\mathsf{from}\ (\mathsf{S}\ \mathsf{Z})) \\ \to_{\mathsf{H}} & \mathsf{Cons}\ \mathsf{Z}\ (\mathsf{Cons}\ (\mathsf{S}\ \mathsf{Z})\ (\mathsf{from}\ (\mathsf{S}\ (\mathsf{S}\ \mathsf{Z})))) \\ \to_{\mathsf{H}} & \ldots \end{array}$$

*evaluation position*

# Semantics and Termination of HASKELL

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

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

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

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

# Semantics and Termination of HASKELL

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

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

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

$$\begin{array}{ll} & \mathsf{take}\, (\mathsf{S}\, \mathsf{Z})\, (\mathsf{from}\, m) \\ \to_{\mathsf{H}} & \mathsf{take}\, (\mathsf{S}\, \mathsf{Z})\, (\mathsf{Cons}\, m\, (\mathsf{from}\, (\mathsf{S}\, m))) \\ \to_{\mathsf{H}} & \mathsf{Cons}\, m\, (\mathsf{take}\, \mathsf{Z}\, (\mathsf{from}\, (\mathsf{S}\, m))) \\ \to_{\mathsf{H}} & \mathsf{Cons}\, m\, \mathsf{Nil} \end{array}$$

*evaluation position*

# Semantics and Termination of HASKELL

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

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

- **Evaluation Relation** $\rightarrow_{\mathsf{H}}$
- **H-Termination** of ground term $t$ if
  - $t$ does not start infinite evaluation $t \rightarrow_{\mathsf{H}} \ldots$
  - if $t \rightarrow_{\mathsf{H}}^{*} (f\ t_1 \ldots t_n)$, $f$ defined, $n < \mathrm{arity}(f)$,
    then $(f\ t_1 \ldots t_n\ t')$ is also H-terminating if $t'$ is H-terminating
  - if $t \rightarrow_{\mathsf{H}}^{*} (c\ t_1 \ldots t_n)$, $c$ 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

# Proving Termination of HASKELL

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

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

- **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 $\Rightarrow$ termination graph
  - do not transform HASKELL into TRS directly,
    but transform termination graph into TRS

# From HASKELL to Termination Graphs

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

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

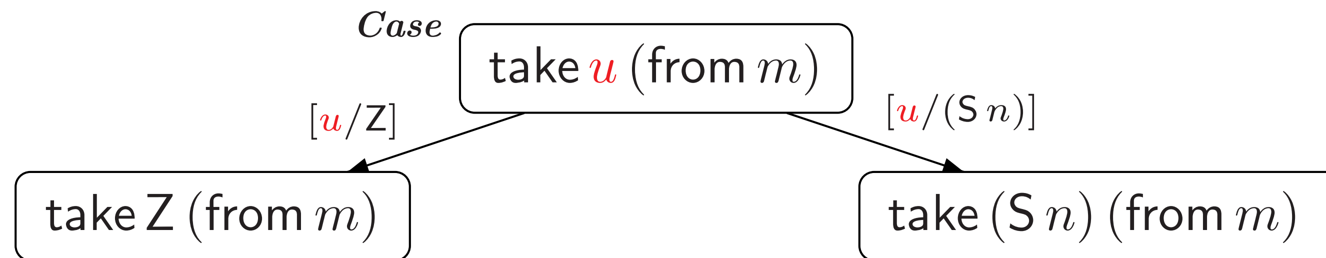
$\mathsf{take}\ u\ (\mathsf{from}\ m)$

- begin with node marked with start term
- 5 expansion rules to add children to leaves
- expansion rules try to *evaluate* terms

# From HASKELL to Termination Graphs

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

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

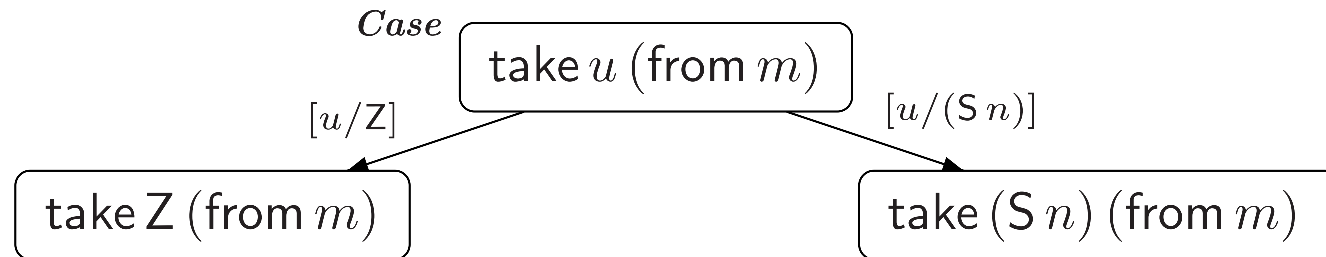



- ***Case*** rule:
  - evaluation has to continue with variable $u$
  - instantiate $u$ by all possible constructor terms of correct type

# From HASKELL to Termination Graphs

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

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



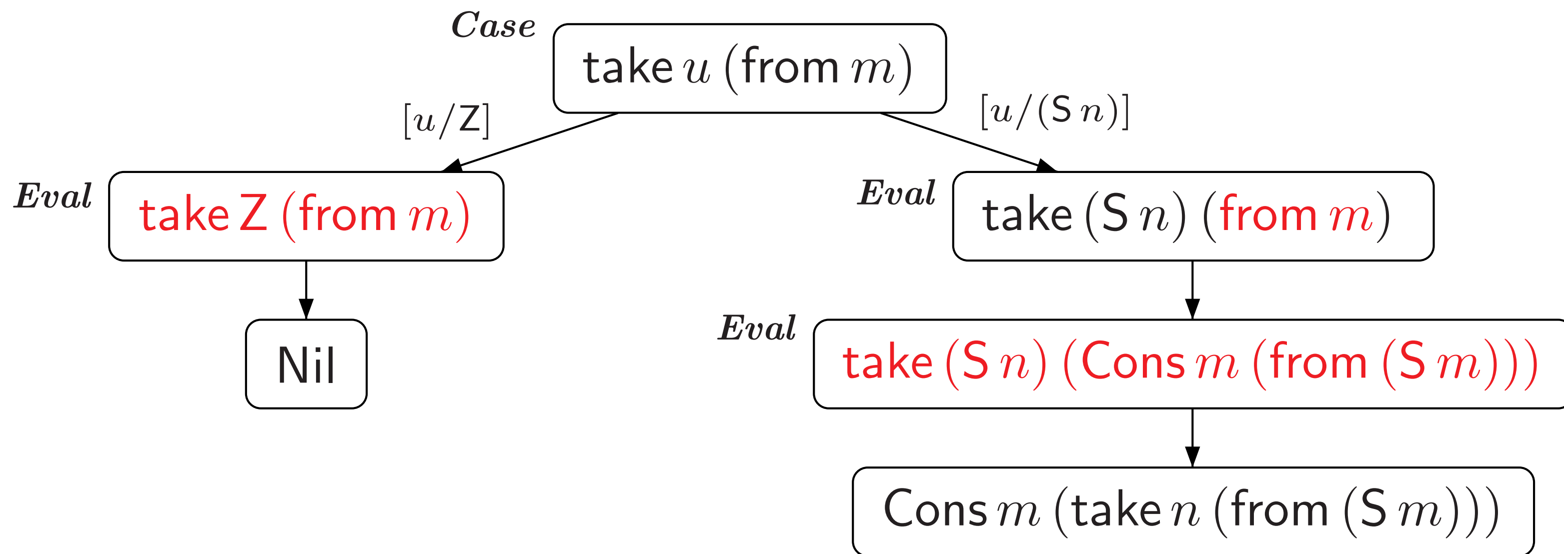

- **Main Property of Termination Graphs:**

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

# From HASKELL to Termination Graphs

$$\text{from}\; x = \text{Cons}\; x\; (\text{from}\; (\text{S}\; x))$$

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



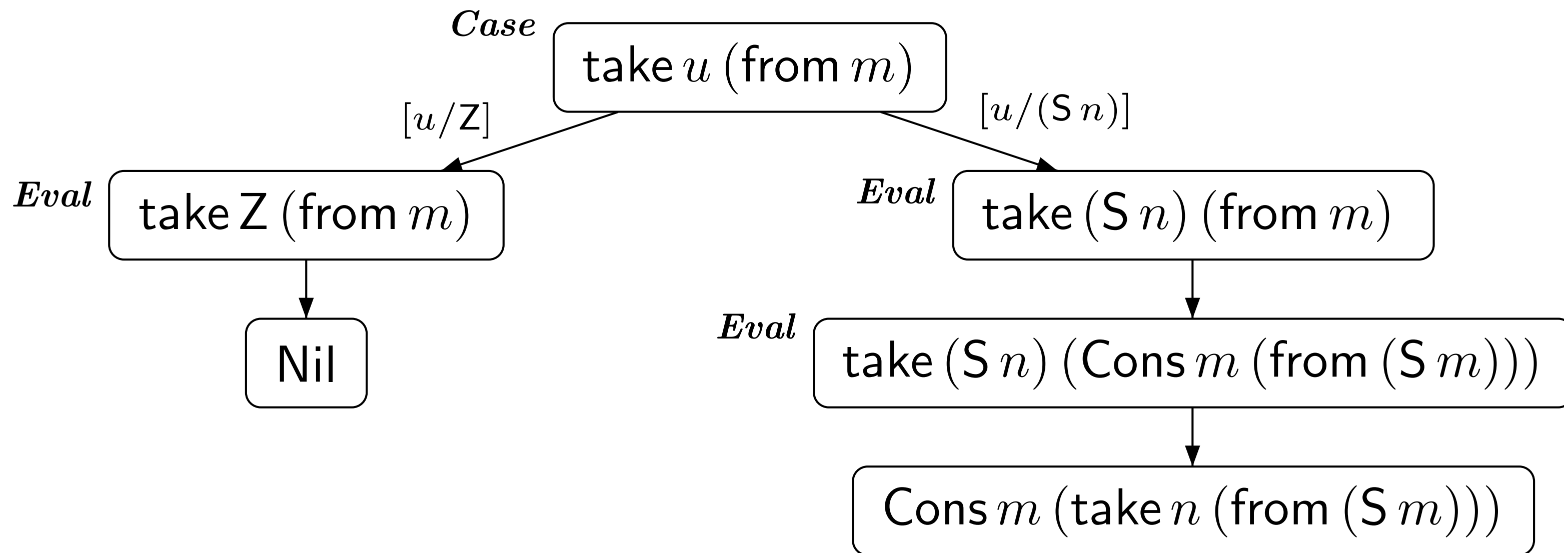

- ***Eval*** rule:

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

# From HASKELL to Termination Graphs

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

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

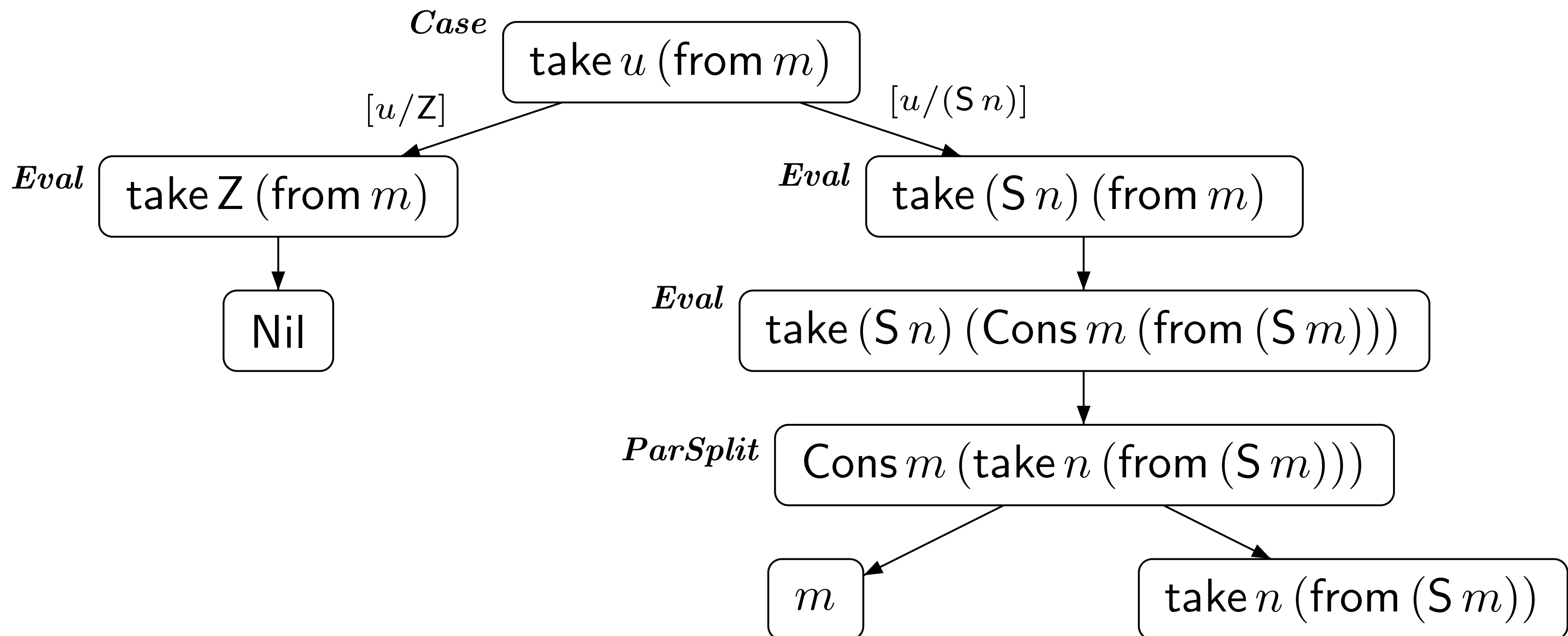



***Case*** and ***Eval*** rule perform *narrowing*
w.r.t. HASKELL's evaluation strategy and types

# From HASKELL to Termination Graphs

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

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



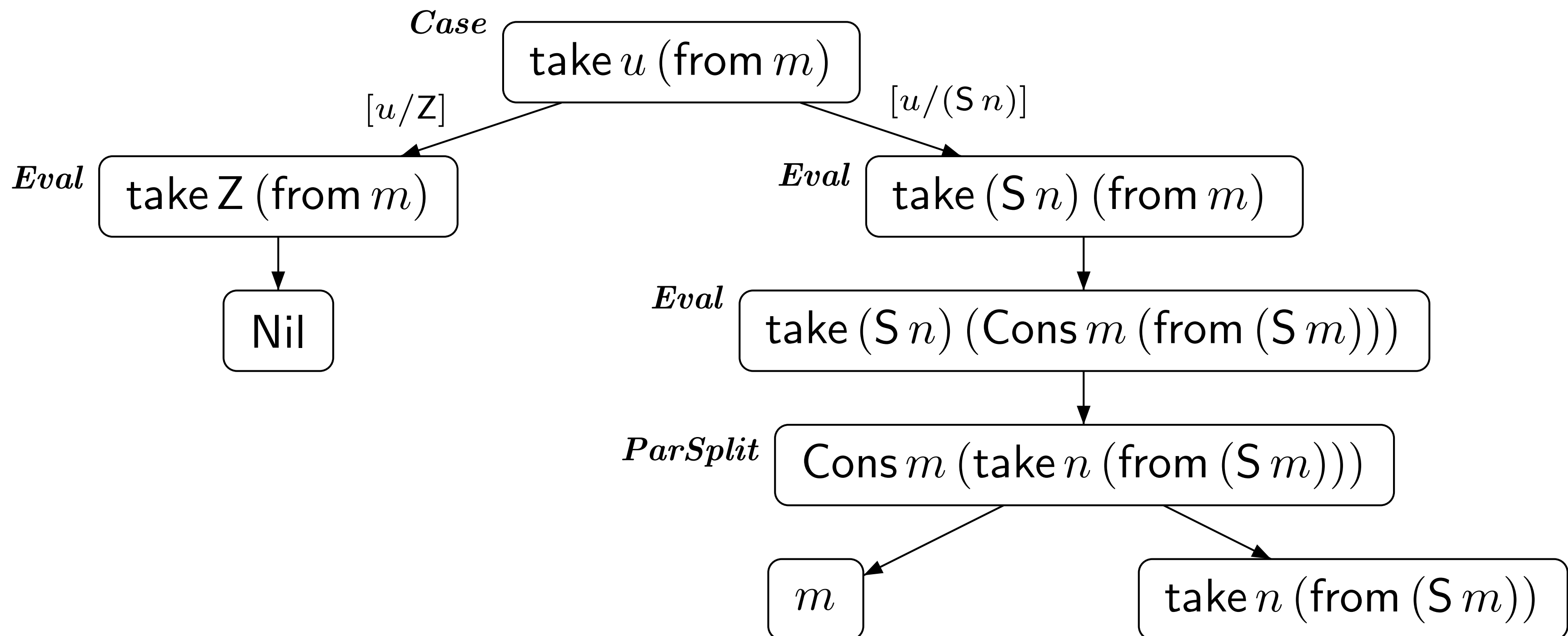

- ***ParSplit*** rule:

  if head of term is a constructor like $\mathsf{Cons}$ or a variable, then continue with the parameters

# From HASKELL to Termination Graphs

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

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

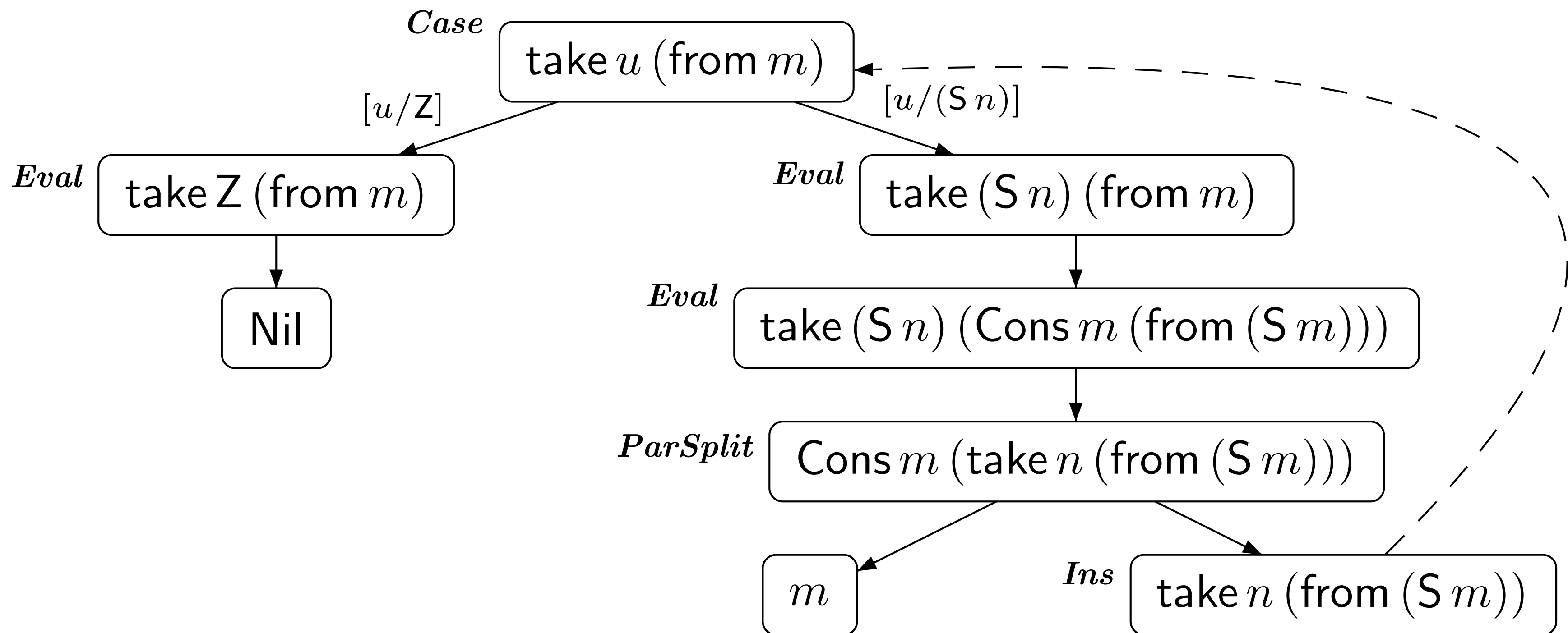



- one could continue with ***Case***, ***Eval***, ***ParSplit***
  $\Rightarrow$ infinite tree
- Instead: ***Ins*** rule to obtain finite graphs

# From HASKELL to Termination Graphs

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

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

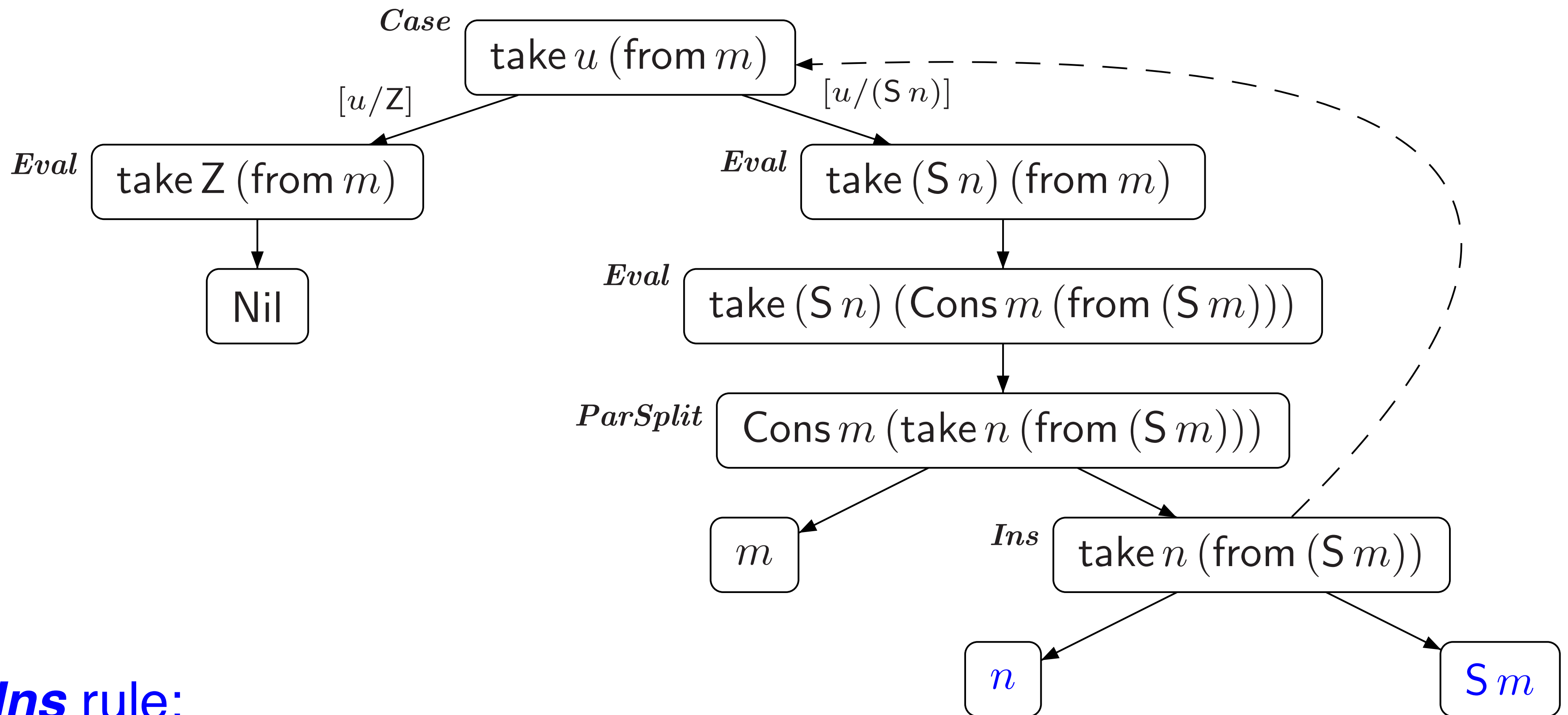



- ***Ins*** **rule:**
  - if leaf $t$ is instance of $t'$, then add instantiation edge from $t$ to $t'$
  - one may re-use an existing node for $t'$, if possible

# From HASKELL to Termination Graphs

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

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

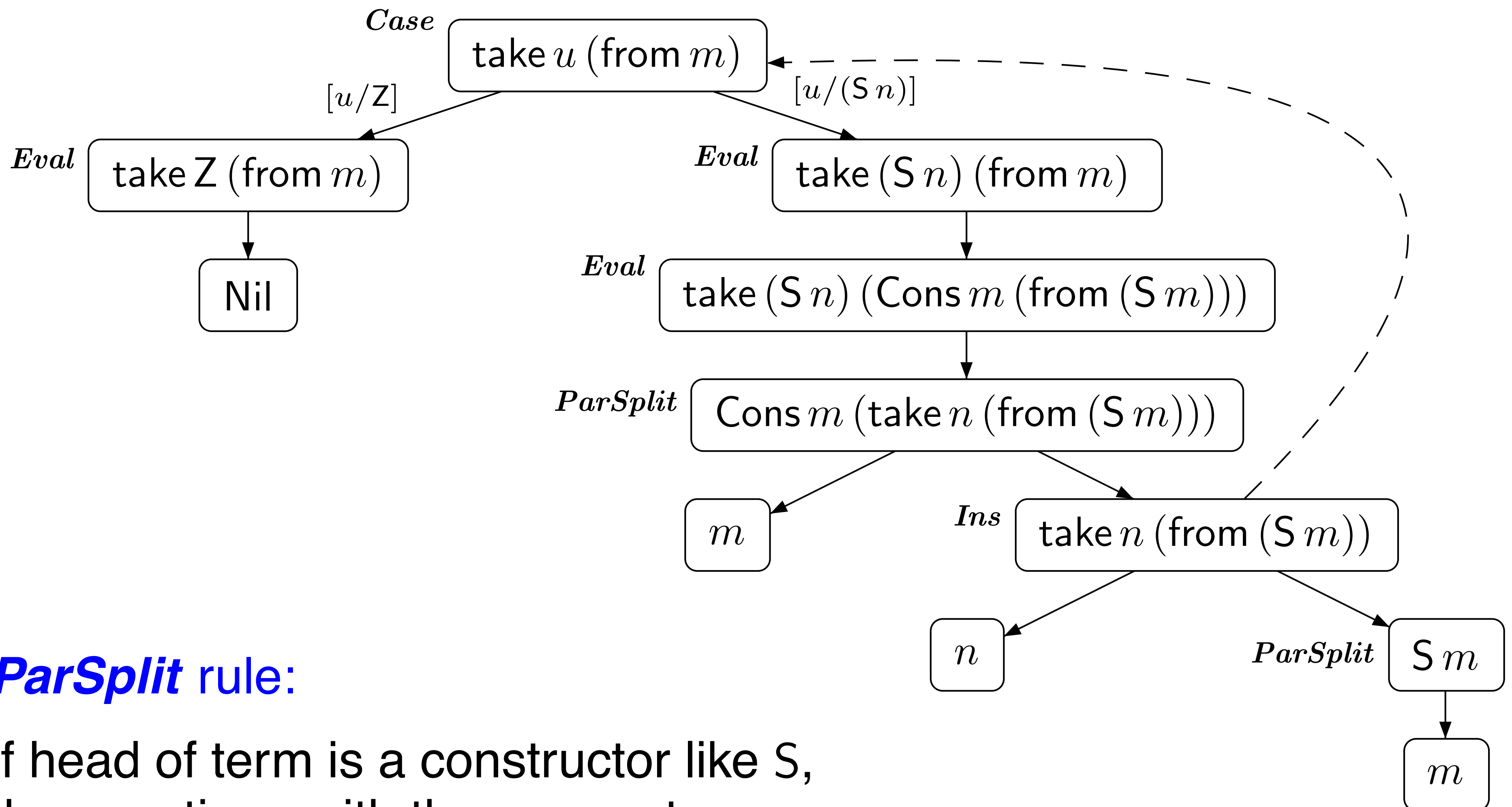



- ***Ins*** **rule:**
  - if leaf $t$ is instance of $t'$, then add instantiation edge from $t$ to $t'$
  - since instantiation is $[u/n,\ m/(\mathsf{S}\ m)]$, add child nodes $n$ and $(\mathsf{S}\ m)$

# From HASKELL to Termination Graphs

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

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



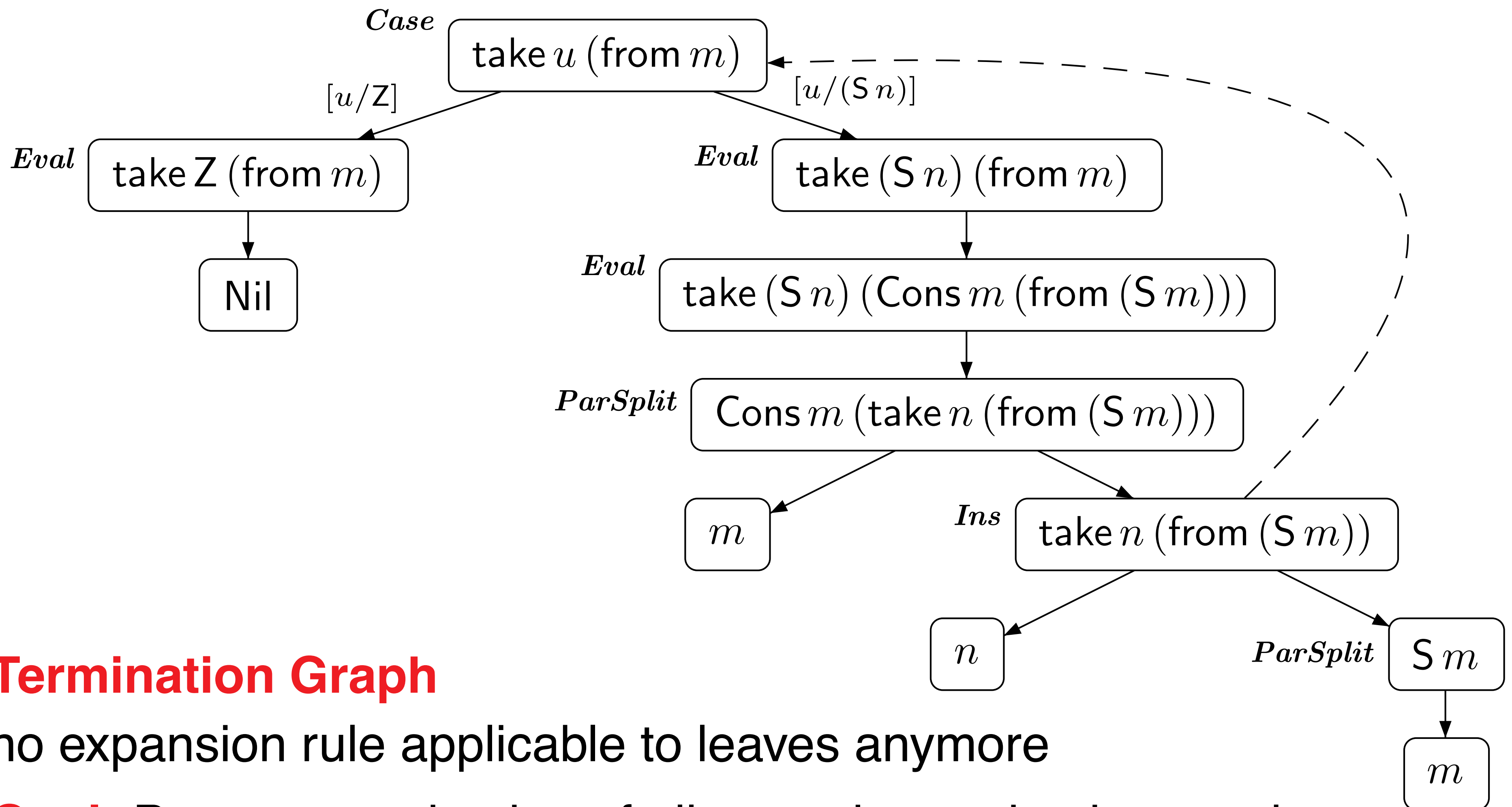

- ***ParSplit*** rule:

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

# From HASKELL to Termination Graphs

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

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

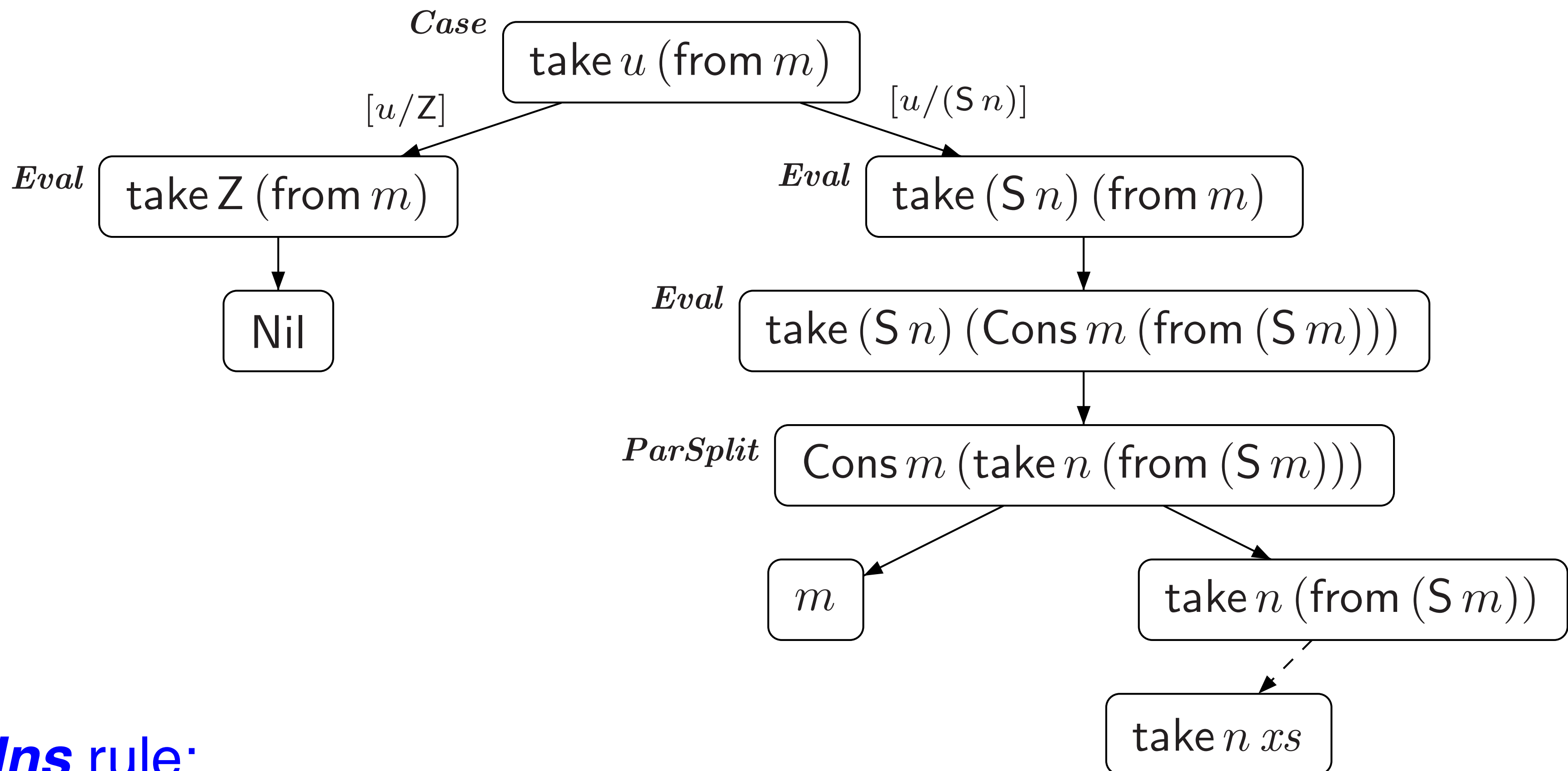



- **Termination Graph**
  no expansion rule applicable to leaves anymore
- **Goal:** Prove H-termination of all terms in termination graph

# From HASKELL to Termination Graphs

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

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

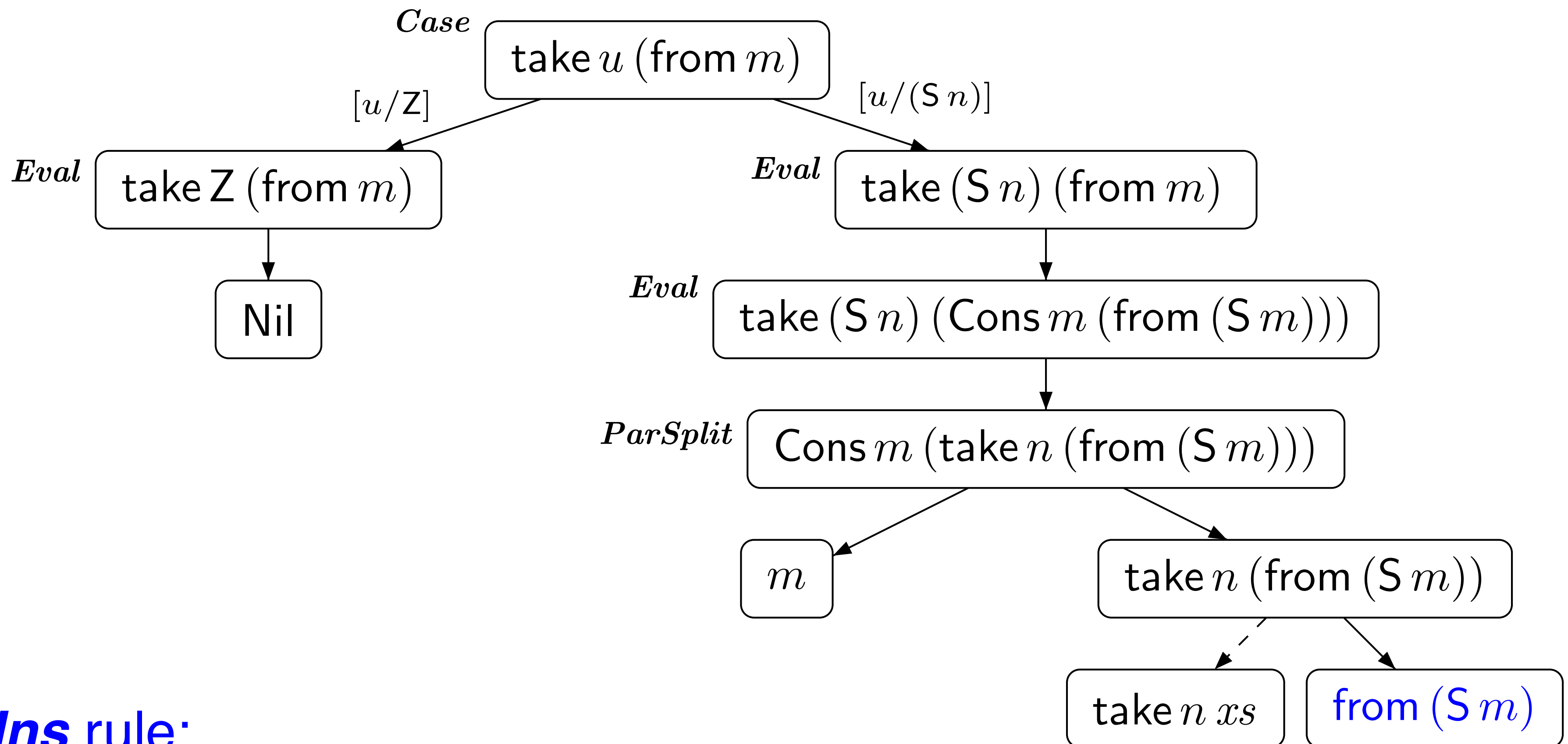



- ***Ins*** **rule:**
  - if leaf $t$ is instance of $t'$, then add instantiation edge from $t$ to $t'$
  - introduces *indeterminism*

# From HASKELL to Termination Graphs

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

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

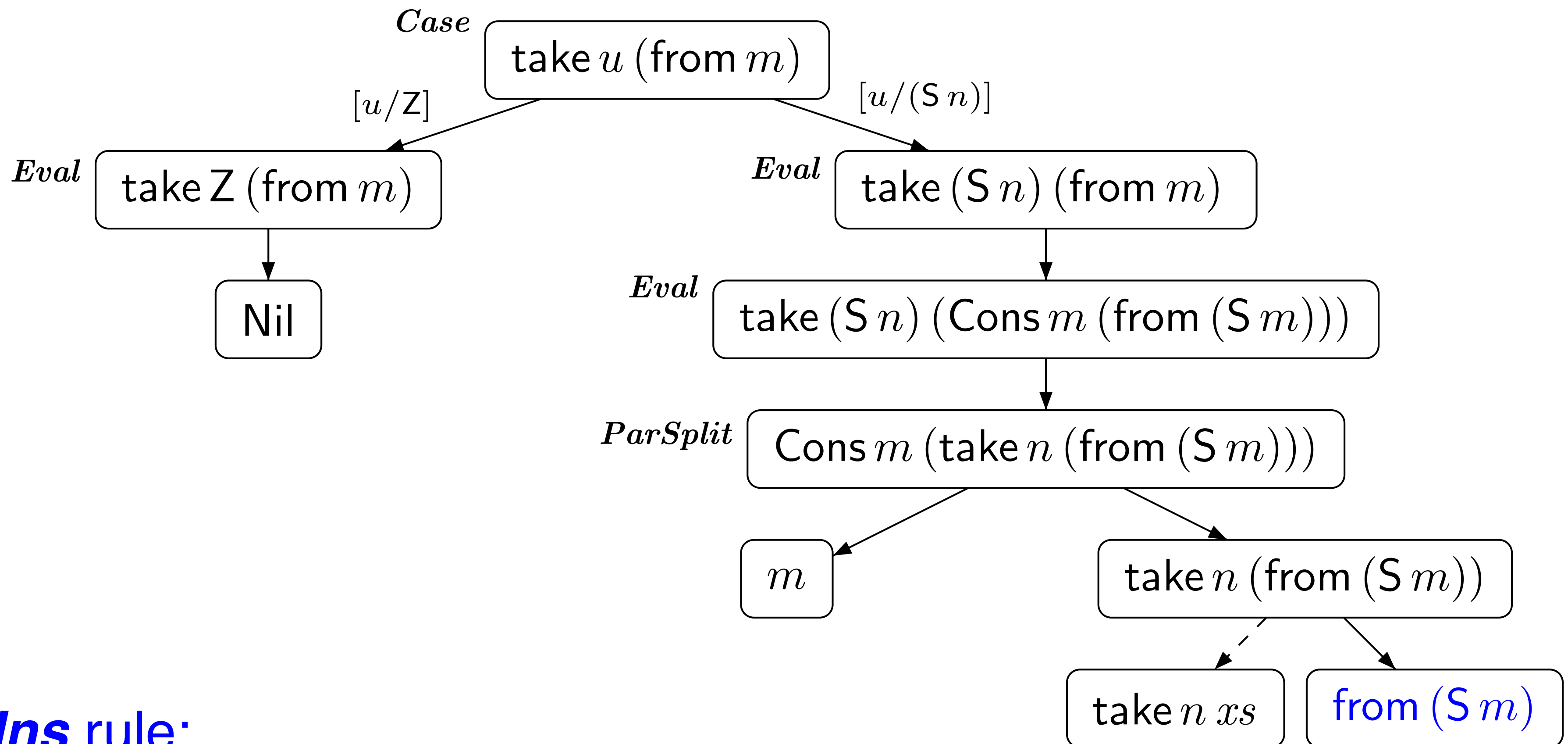



- ***Ins*** **rule:**
  - if leaf $t$ is instance of $t'$, then add instantiation edge from $t$ to $t'$
  - since instantiation is $[xs/\mathsf{from}\ (\mathsf{S}\ m)]$, add child node $\mathsf{from}\ (\mathsf{S}\ m)$

# From HASKELL to Termination Graphs

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

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



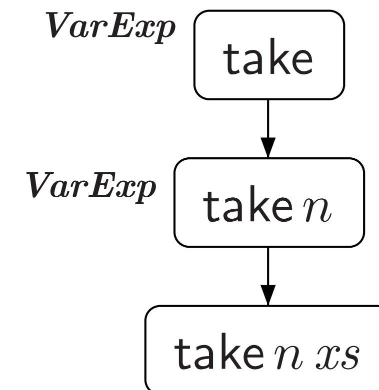

- ***Ins*** **rule:**
  - if leaf $t$ is instance of $t'$, then add instantiation edge from $t$ to $t'$
  - proving H-termination of all terms in termination graph fails!

# From HASKELL to Termination Graphs

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

## Expansion Rules

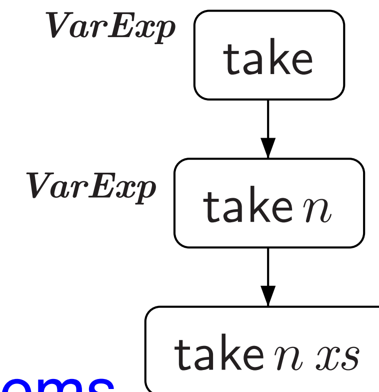
- ***Case***
- ***Eval***
- ***ParSplit***
- ***Ins***
- ***VarExp***




- ***VarExp*** rule:
  - if function is applied to too few arguments,
    then add fresh variable as additional argument

# From Termination Graphs to TRSs

- Termination graphs can be obtained for any start term
- **Goal:** Prove H-termination of all terms in termination graph
- **First Approach:**
  Transform termination graph into TRS
  $\Rightarrow$ disadvantageous




- **Better Approach:**
  Transform termination graph into DP problems

- **Dependency Pairs**
  - powerful & popular termination technique for TRSs
  - DP framework allows integration & combination of *any* TRS-termination technique

# 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*"?
  - DP problems $(\mathcal{P}, \mathcal{R})$

    $\mathcal{P}$ *dependency pairs*

    $\mathcal{R}$ *rules*
- 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

    $s_1\sigma_1 \;\to_{\mathcal{P}}\; t_1\sigma_1 \;\to_{\mathcal{R}}^*\; s_2\sigma_2 \;\to_{\mathcal{P}}\; t_2\sigma_2 \;\to_{\mathcal{R}}^*\; \ldots$ where $s_i \to t_i \in \mathcal{P}$

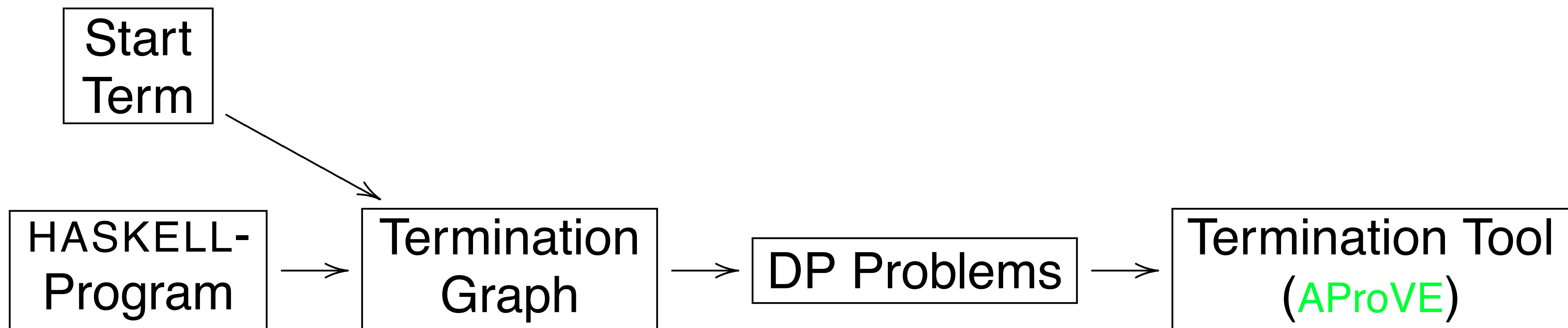
# Dependency Pair Framework

**Termination of TRS $\mathcal{R}$**

- construct initial DP problem $(DP(\mathcal{R}), \mathcal{R})$
- TRS $\mathcal{R}$ is terminating iff initial DP problem is finite
- use DP framework to prove that initial DP problem is finite

**Termination of HASKELL**

- generate termination graph for start term
- construct initial DP problems from termination graph
- start term is H-terminating if initial DP problems are finite
- use DP framework to prove that initial DP problems are finite

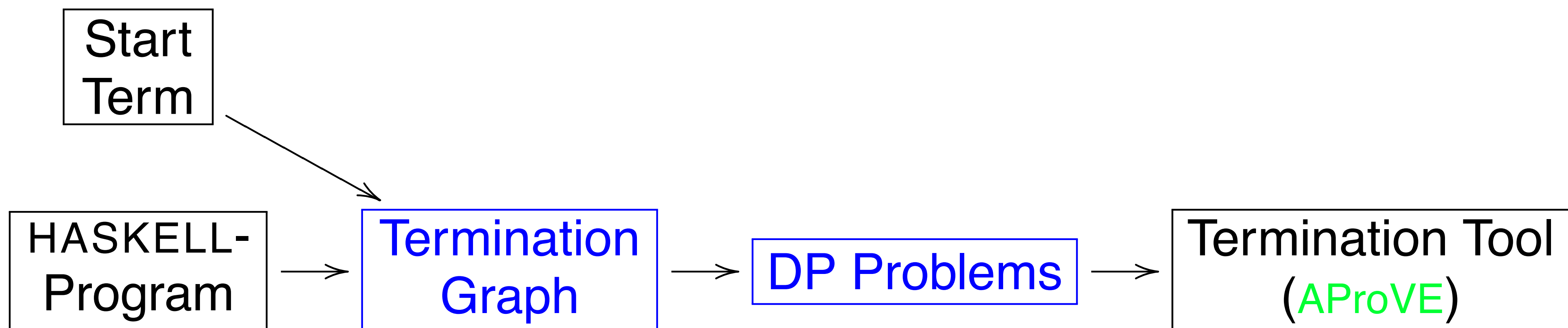


# Dependency Pair Framework

How to construct DP problems from termination graph?

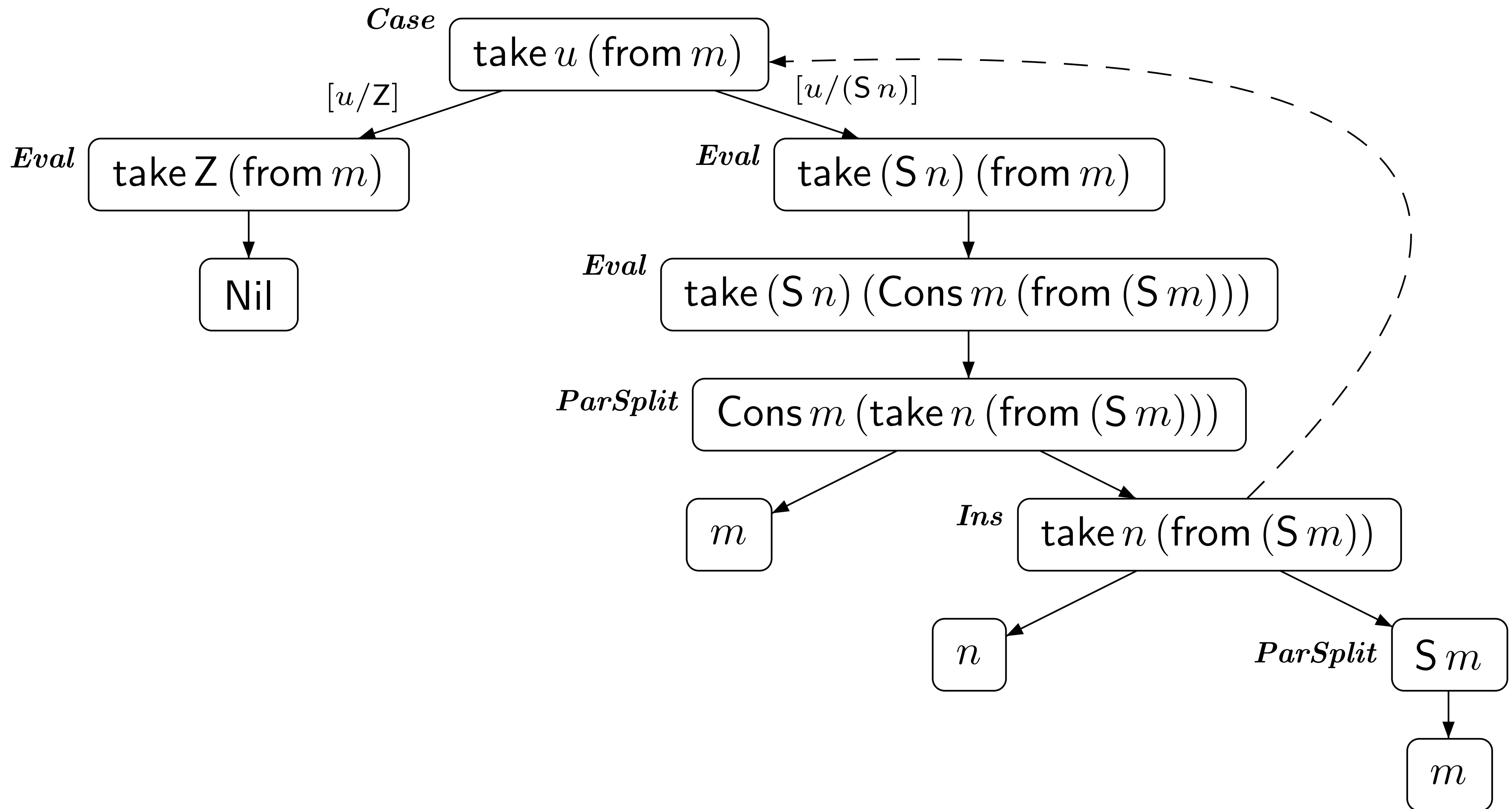
**Termination of** HASKELL

- generate termination graph for start term
- construct initial DP problems from termination graph
- start term is H-terminating if initial DP problems are finite
- use DP framework to prove that initial DP problems are finite



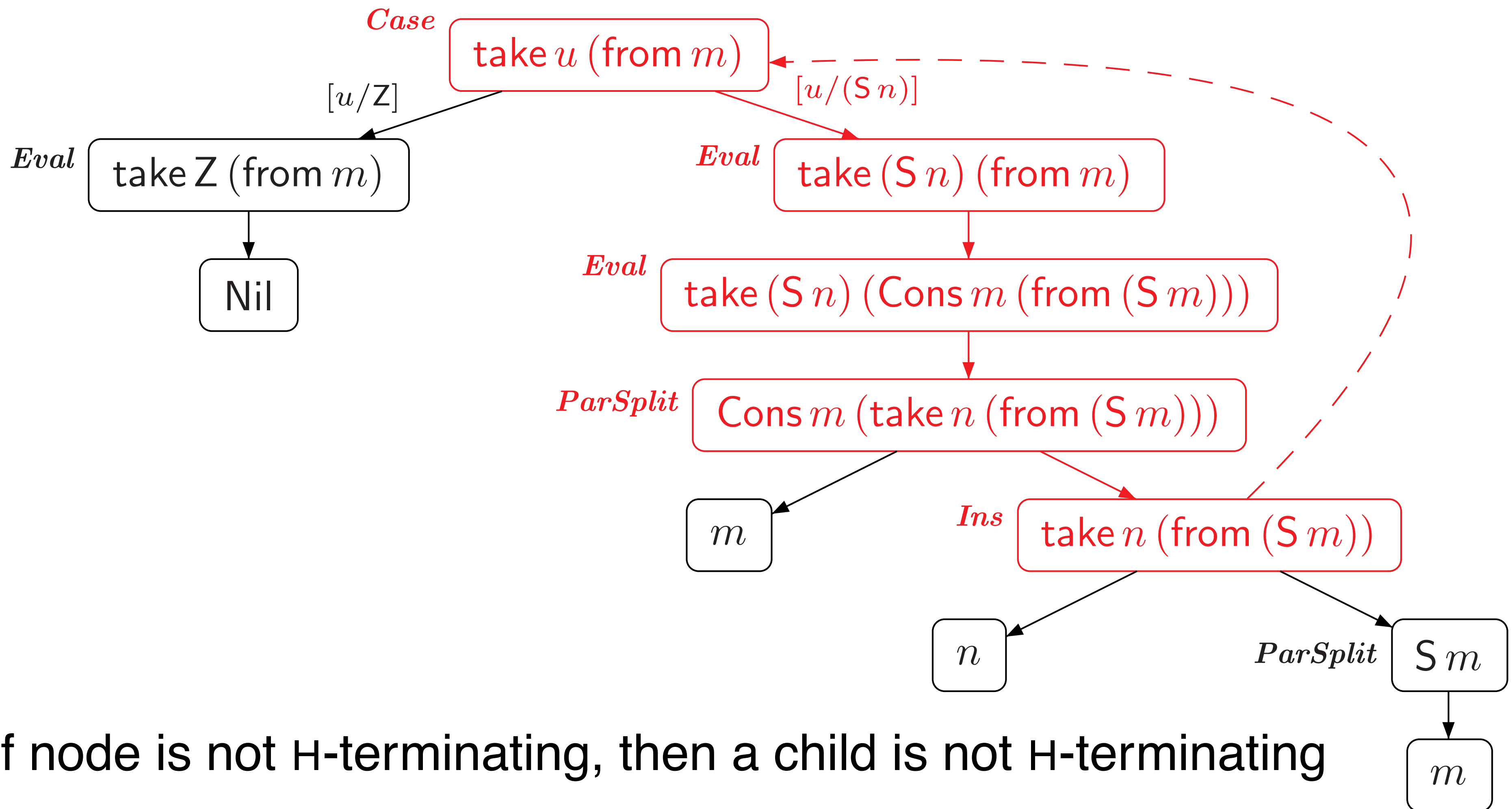
# From Termination Graphs to DP Problems

- 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

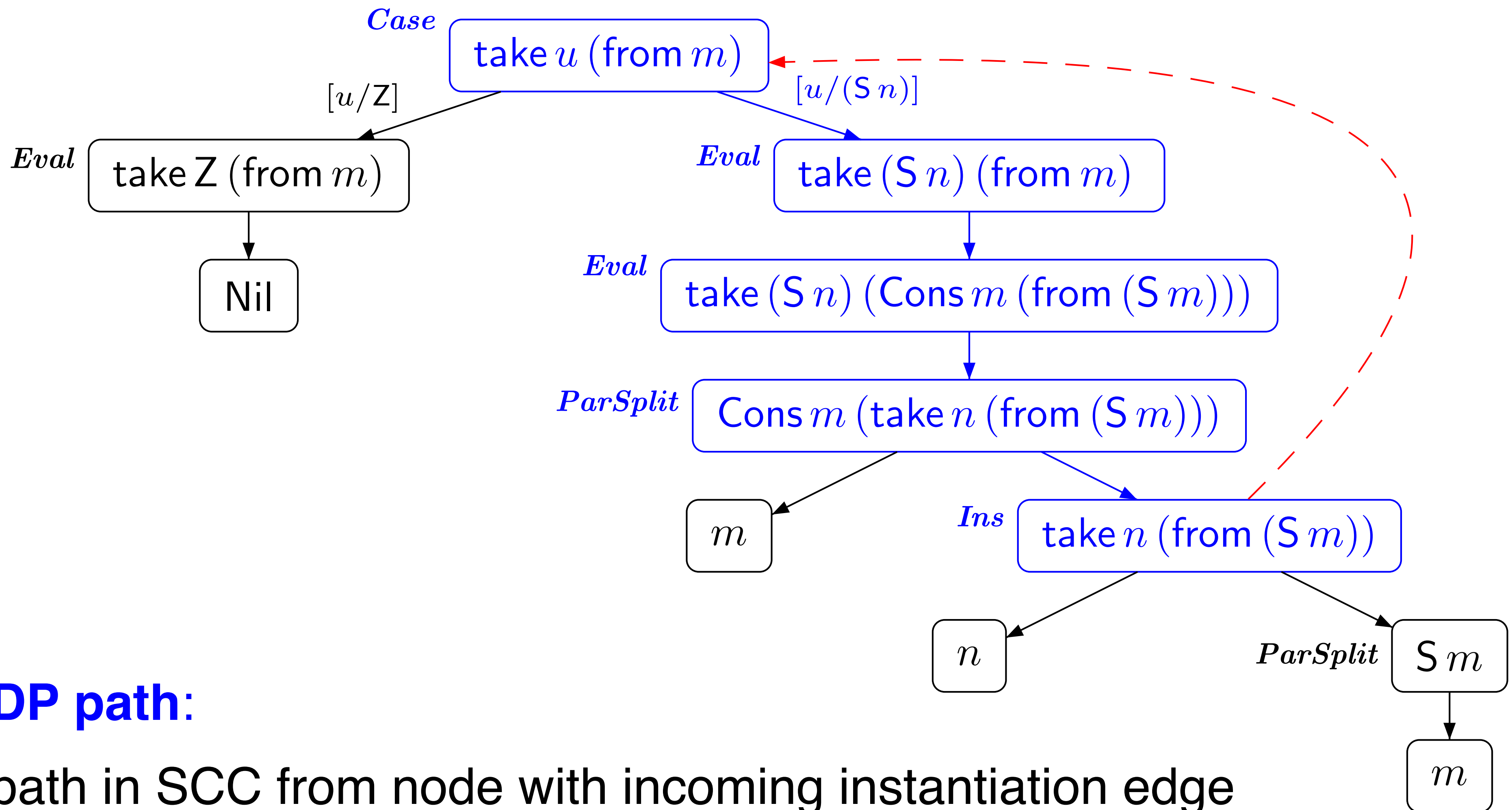



- if node is not H-terminating, then a child is not H-terminating
- not H-terminating node corresponds to SCC

# From Termination Graphs to DP Problems

- every infinite path traverses a DP path infinitely often

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



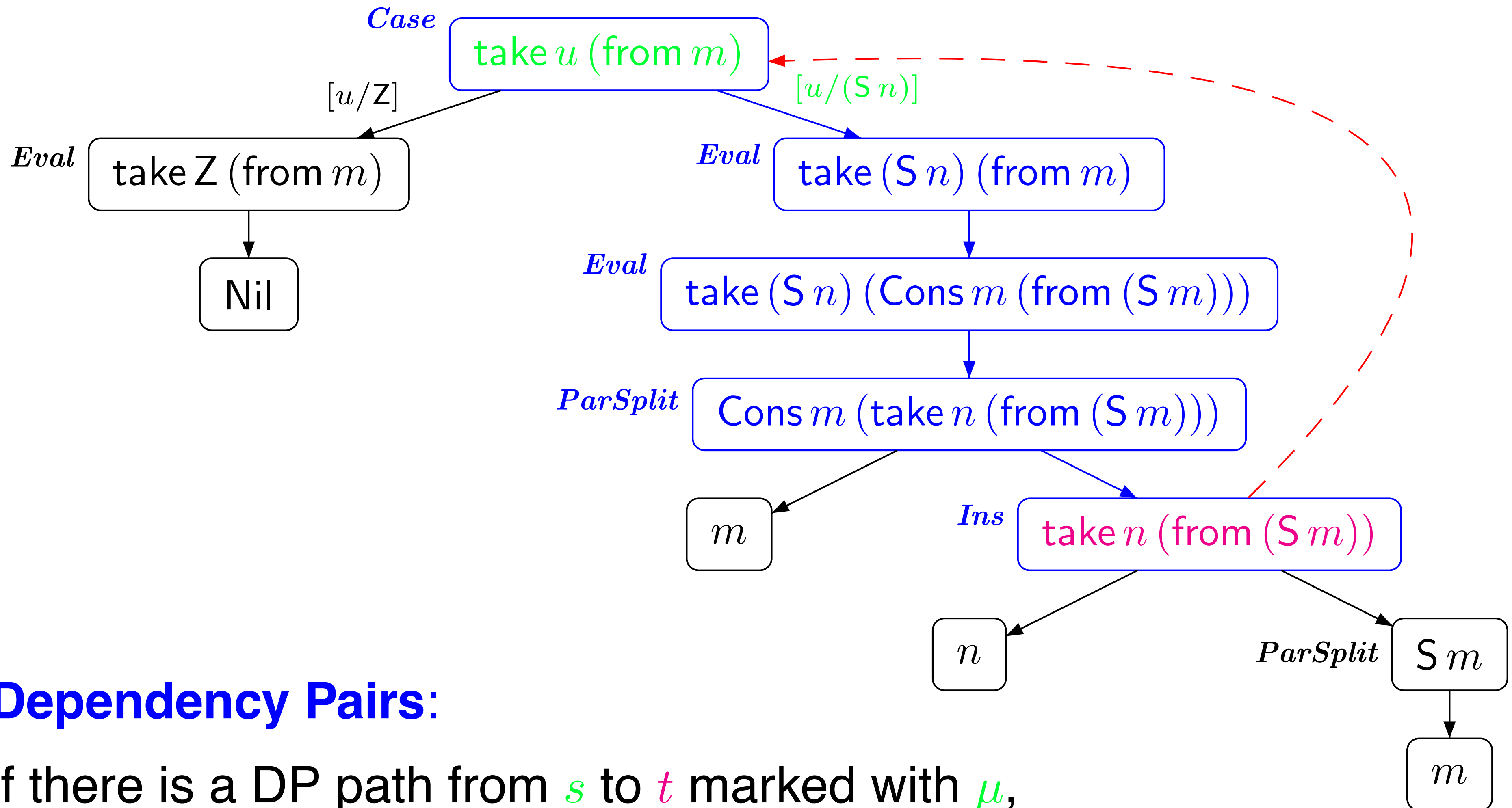

- **DP path**:

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

# From Termination Graphs to DP Problems

- **Dependency Pair $\mathcal{P}$:** take (S $n$) (from $m$) $\to$ take $n$ (from (S $m$))

  **Rules $\mathcal{R}$:** $\varnothing$ termination is easy to prove




- **Dependency Pairs:**

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

# Generating infinite $(\mathcal{P}, \mathcal{R})$-chains

Term in graph not terminating

$\curvearrowright$ $s_1$ not terminating

DP path from $s_1$ to $t_1$ marked with $\mu_1$

$\curvearrowright$ $s_1 \ \tau_1$ not terminating

$\curvearrowright$ $s_1 \ (\tau_1 {\downarrow_\mathsf{H}})$ not terminating

$\curvearrowright$ $s_1 \ \mu_1 \ \sigma_1$ not terminating

$\curvearrowright$ $t_1 \ \sigma_1$ not terminating

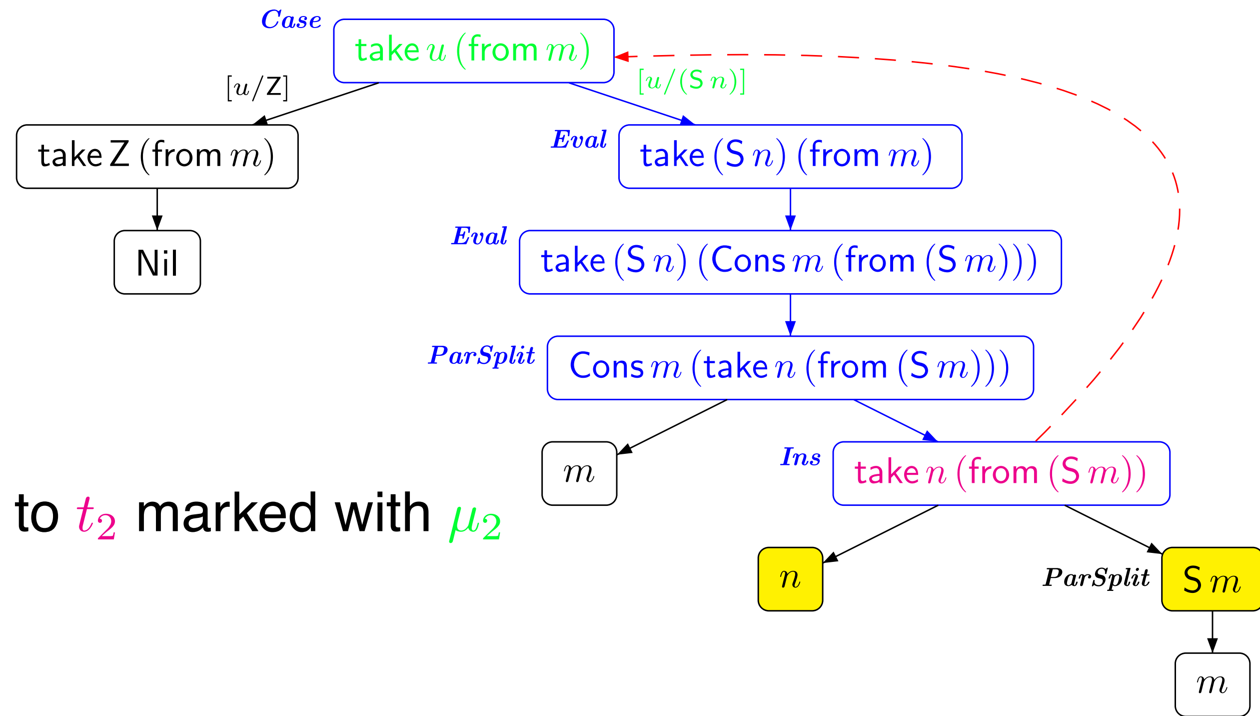
$\curvearrowright$ $s_2 \ \tau_2$ not terminating

DP path from $s_2$ to $t_2$ marked with $\mu_2$

$\curvearrowright$ $s_2 \ (\tau_2 {\downarrow_\mathsf{H}})$ not terminating

$\curvearrowright$ $s_2 \ \mu_2 \ \sigma_2$ not terminating

$\curvearrowright$ $t_2 \ \sigma_2$ not terminating




$$s_1 \, \mu_1 \, \sigma_1 \to_{\mathcal{P}} \underbrace{t_1 \sigma_1}_{s_2 \ \tau_2} \quad \to^*_{\mathcal{R}} \quad \underbrace{s_2 \, \mu_2 \, \sigma_2}_{s_2 \ (\tau_2 \downarrow_\mathsf{H})} \to_{\mathcal{P}} t_2 \sigma_2$$

$\mathcal{R}$: rules for terms in matcher

$\mathcal{R}=\varnothing$ if no defined symbol in matcher

# From Termination Graphs to DP Problems

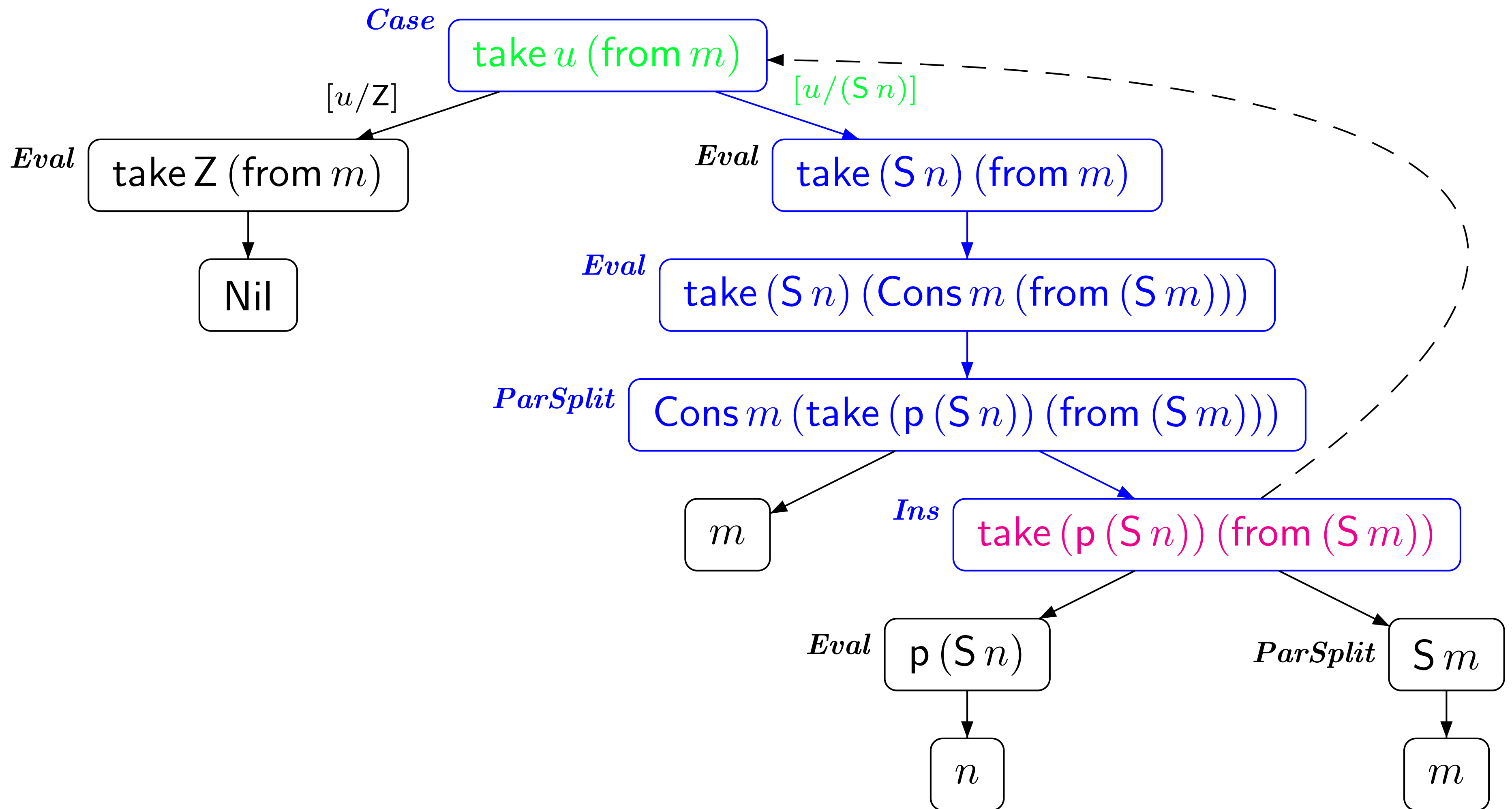
$\mathsf{from}\ x = \mathsf{Cons}\ x\ (\mathsf{from}\ (\mathsf{S}\ x))$

$\mathsf{p}\ (\mathsf{S}\ x) = x$

$\mathsf{take}\ \mathsf{Z}\ xs = \mathsf{Nil}$

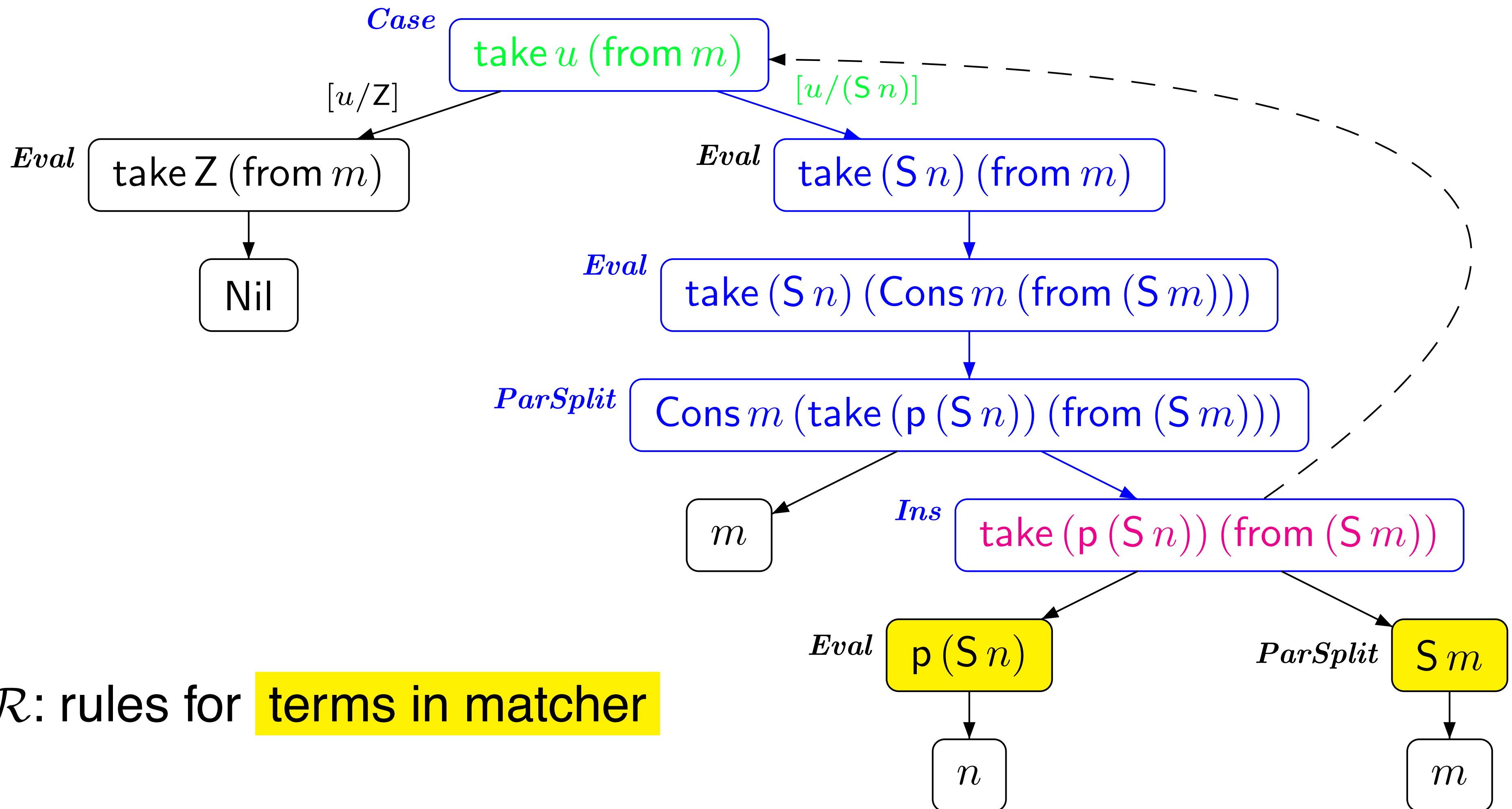
$\mathsf{take}\ n\ \mathsf{Nil} = \mathsf{Nil}$

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



# From Termination Graphs to DP Problems

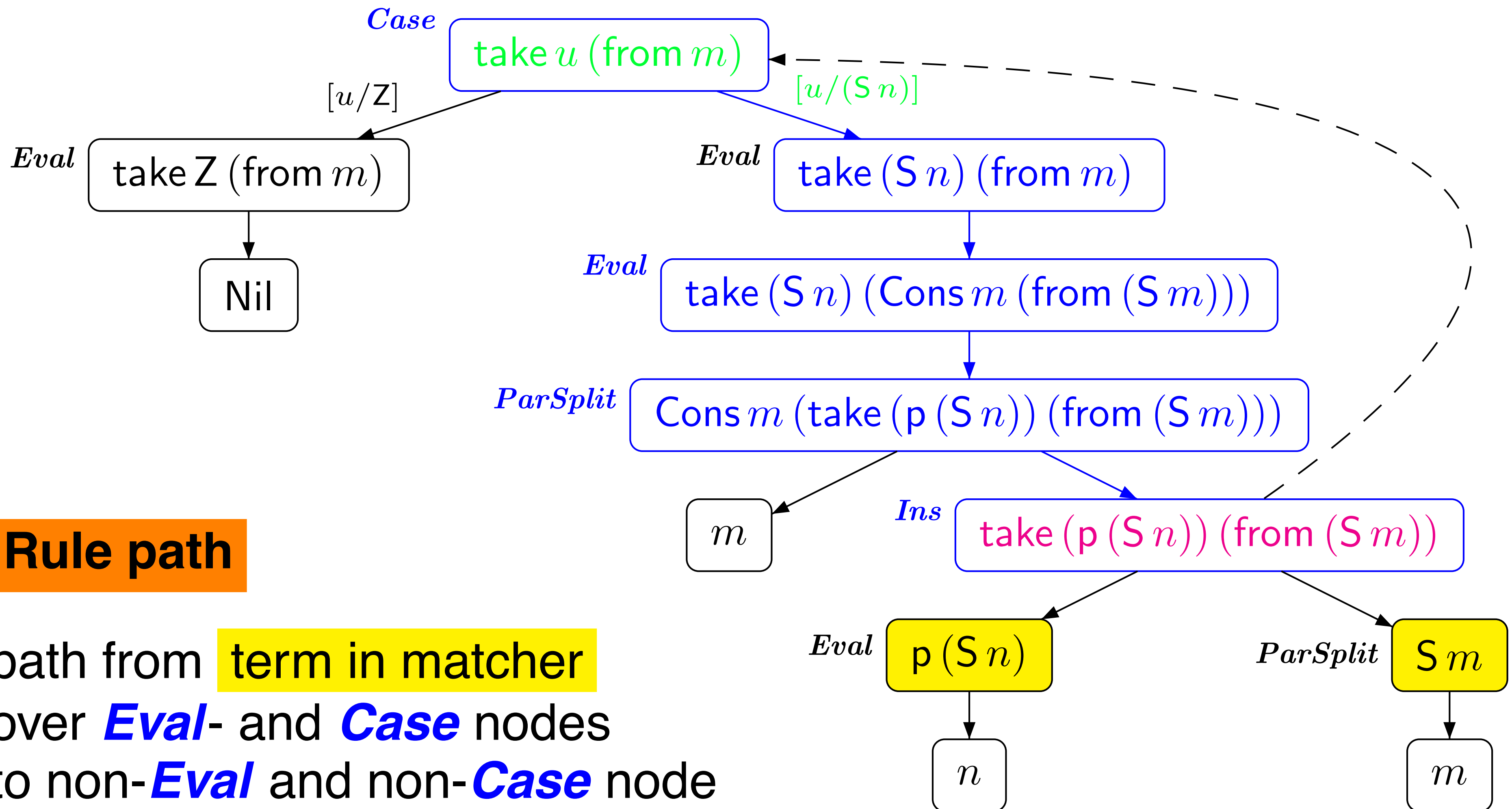
- **Dependency Pair $\mathcal{P}$:** take (S $n$) (from $m$) $\to$ take (p (S $n$)) (from (S $m$))




- $\mathcal{R}$: rules for terms in matcher

# From Termination Graphs to DP Problems

- **Dependency Pair $\mathcal{P}$:** take (S $n$) (from $m$) $\rightarrow$ take (p (S $n$)) (from (S $m$))



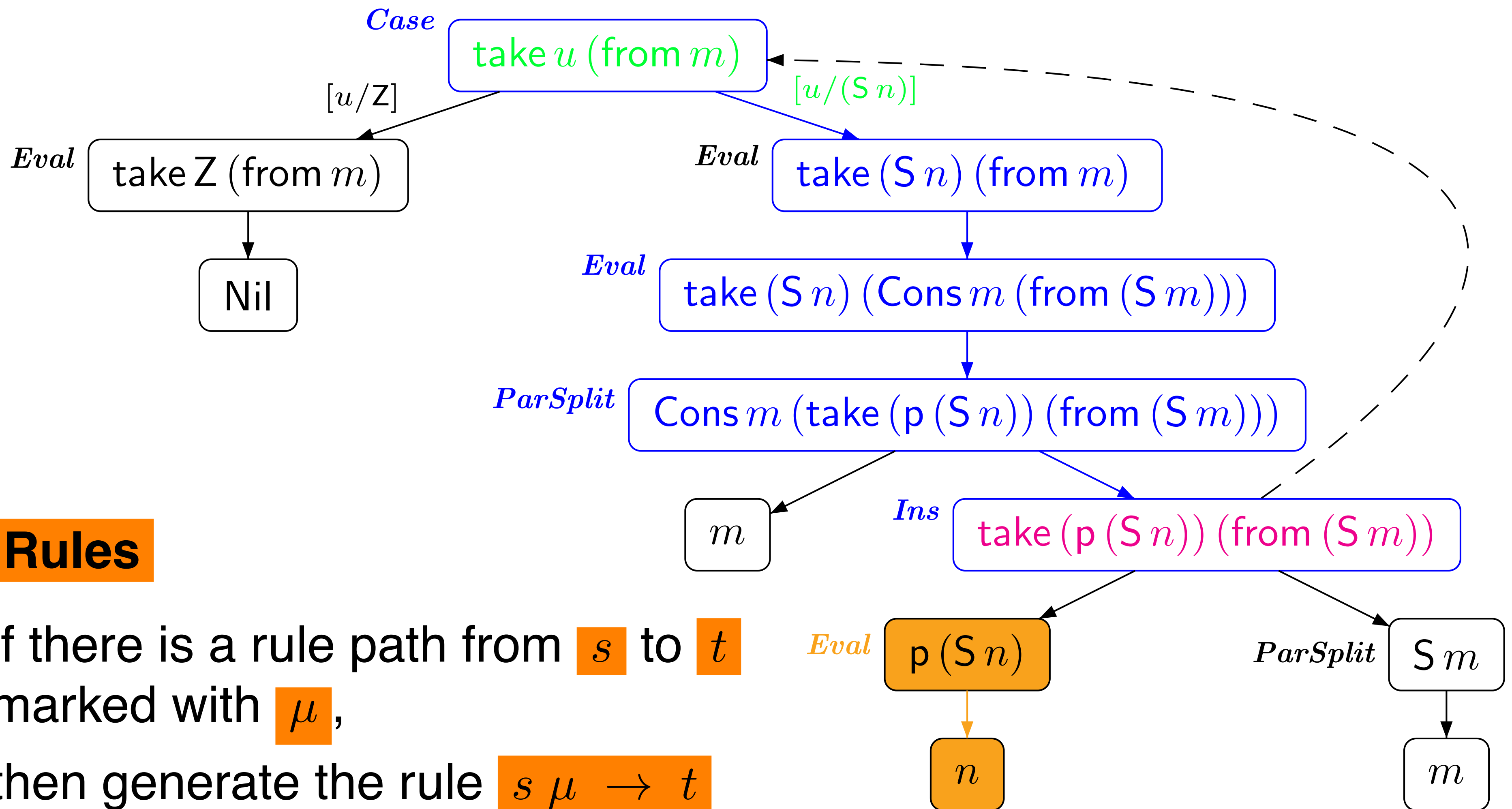

- **Rule path**

path from term in matcher
over ***Eval***- and ***Case*** nodes
to non-***Eval*** and non-***Case*** node

# From Termination Graphs to DP Problems

- **Dependency Pair $\mathcal{P}$:** $\mathsf{take}\,(\mathsf{S}\,n)\,(\mathsf{from}\,m) \to \mathsf{take}\,(\mathsf{p}\,(\mathsf{S}\,n))\,(\mathsf{from}\,(\mathsf{S}\,m))$

  **Rule $\mathcal{R}$:** $\mathsf{p}\,(\mathsf{S}\,n) \;\to\; n$ termination easy to prove




- **Rules**

  if there is a rule path from $s$ to $t$ marked with $\mu$,

  then generate the rule $s\,\mu \;\to\; t$

# From Termination Graphs to DP Problems

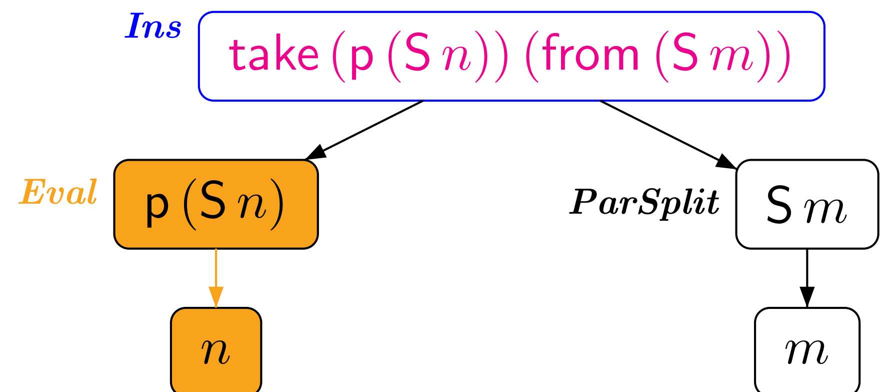
- **Dependency Pair $\mathcal{P}$:** $\mathsf{take}\,(\mathsf{S}\,n)\,(\mathsf{from}\,m) \to \mathsf{take}\,(\mathsf{p}\,(\mathsf{S}\,n))\,(\mathsf{from}\,(\mathsf{S}\,m))$

  **Rule $\mathcal{R}$:** $\mathsf{p}\,(\mathsf{S}\,n) \;\to\; n$

- **Improvement:** evaluate rhs of DP as much as possible

$$
\begin{aligned}
& \mathbf{ev}(\,\mathsf{take}\;(\,\mathsf{p}\,(\mathsf{S}\,n)\,)\;(\mathsf{from}\;(\,\mathsf{S}\,m\,))\,)\\
= \;& \mathsf{take}\;\mathbf{ev}(\,\mathsf{p}\,(\mathsf{S}\,n)\,)\;(\mathsf{from}\;\mathbf{ev}(\,\mathsf{S}\,m\,))\\
= \;& \mathsf{take}\;n\;(\mathsf{from}\;(\,\mathsf{S}\,m\,))
\end{aligned}
$$

- $\mathbf{ev}(\,t\,)$ : term reachable from $t$ by traversing ***Eval***-nodes
  traverses subterms of ***ParSplit***- and ***Ins***-nodes



# From Termination Graphs to DP Problems

- **Dependency Pair $\mathcal{P}$:** $\mathsf{take}\,(\mathsf{S}\,n)\,(\mathsf{from}\,m) \to \mathsf{take}\;\;n\;\;(\mathsf{from}\,(\mathsf{S}\,m))$
  **Rule $\mathcal{R}$:** $\varnothing$

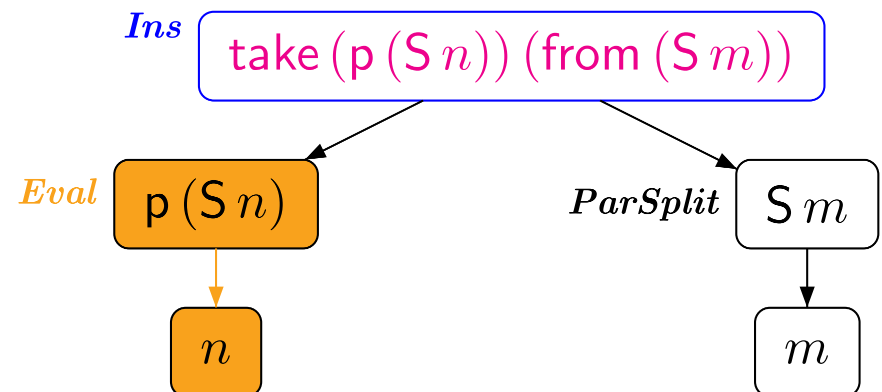
- **Improvement:** evaluate rhs of DP as much as possible

$$\begin{aligned} & \mathbf{ev}(\,\mathsf{take}\;(\,\mathsf{p}\,(\mathsf{S}\,n)\,)\;(\mathsf{from}\;(\,\mathsf{S}\,m\,))\,) \\ = \;& \mathsf{take}\;\mathbf{ev}(\,\mathsf{p}\,(\mathsf{S}\,n)\,)\;(\mathsf{from}\;\mathbf{ev}(\,\mathsf{S}\,m\,)) \\ = \;& \mathsf{take}\;\;n\;\;(\mathsf{from}\;(\,\mathsf{S}\,m\,)) \end{aligned}$$

- $\mathbf{ev}(\,t\,)$ : term reachable from $t$ by traversing ***Eval***-nodes
  traverses subterms of ***ParSplit***- and ***Ins***-nodes

- **Rules**

  only needed for terms
  where computation of $\mathbf{ev}$ stopped



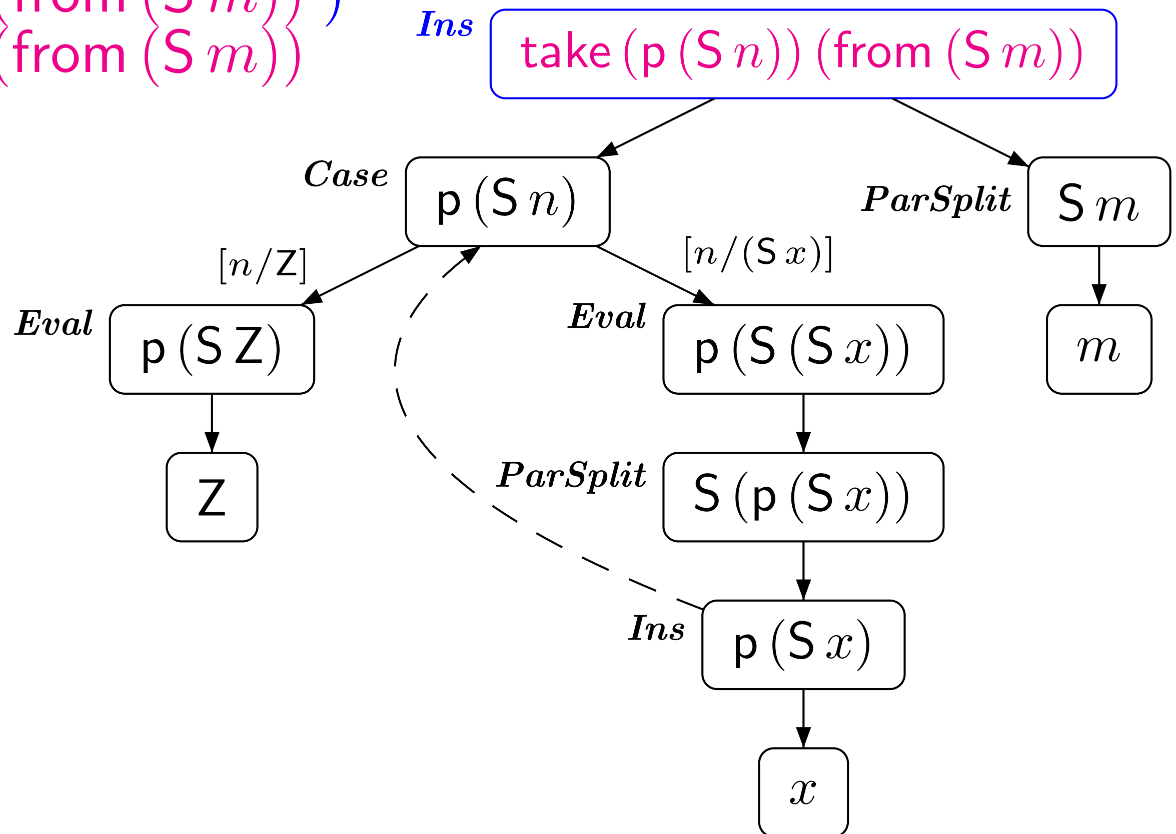
# From Termination Graphs to DP Problems

- **Dependency Pair $\mathcal{P}$:** $\mathsf{take}\,(\mathsf{S}\,n)\,(\mathsf{from}\,m) \to \mathsf{take}\,(\mathsf{p}\,(\mathsf{S}\,n))\,(\mathsf{from}\,(\mathsf{S}\,m))$

- **Improvement:** evaluate rhs of DP as much as possible

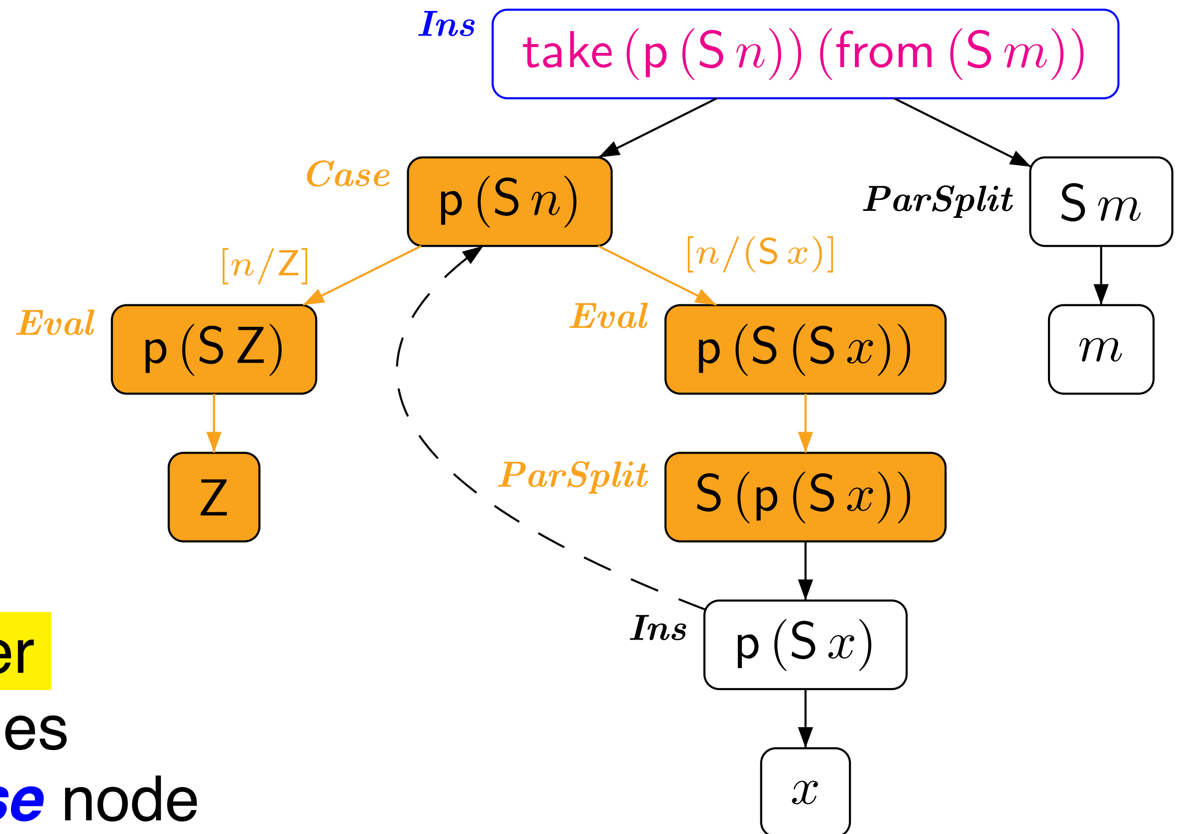
$$\begin{aligned} & \mathbf{ev}(\ \mathsf{take}\,(\mathsf{p}\,(\mathsf{S}\,n))\,(\mathsf{from}\,(\mathsf{S}\,m))\ ) \\ = \ & \mathsf{take}\,(\mathsf{p}\,(\mathsf{S}\,n))\,(\mathsf{from}\,(\mathsf{S}\,m)) \end{aligned}$$

$$\begin{aligned} \mathsf{p}\,(\mathsf{S}\,\mathsf{Z}) &= \mathsf{Z} \\ \mathsf{p}\,(\mathsf{S}\,x) &= \mathsf{S}\,(\mathsf{p}\,x) \end{aligned}$$



# From Termination Graphs to DP Problems

- **Dependency Pair $\mathcal{P}$:** take (S $n$) (from $m$) $\to$ take (p (S $n$)) (from (S $m$))




- **Rule path**

  path from term in matcher
  over ***Eval***- and ***Case*** nodes
  to non-***Eval*** and non-***Case*** node

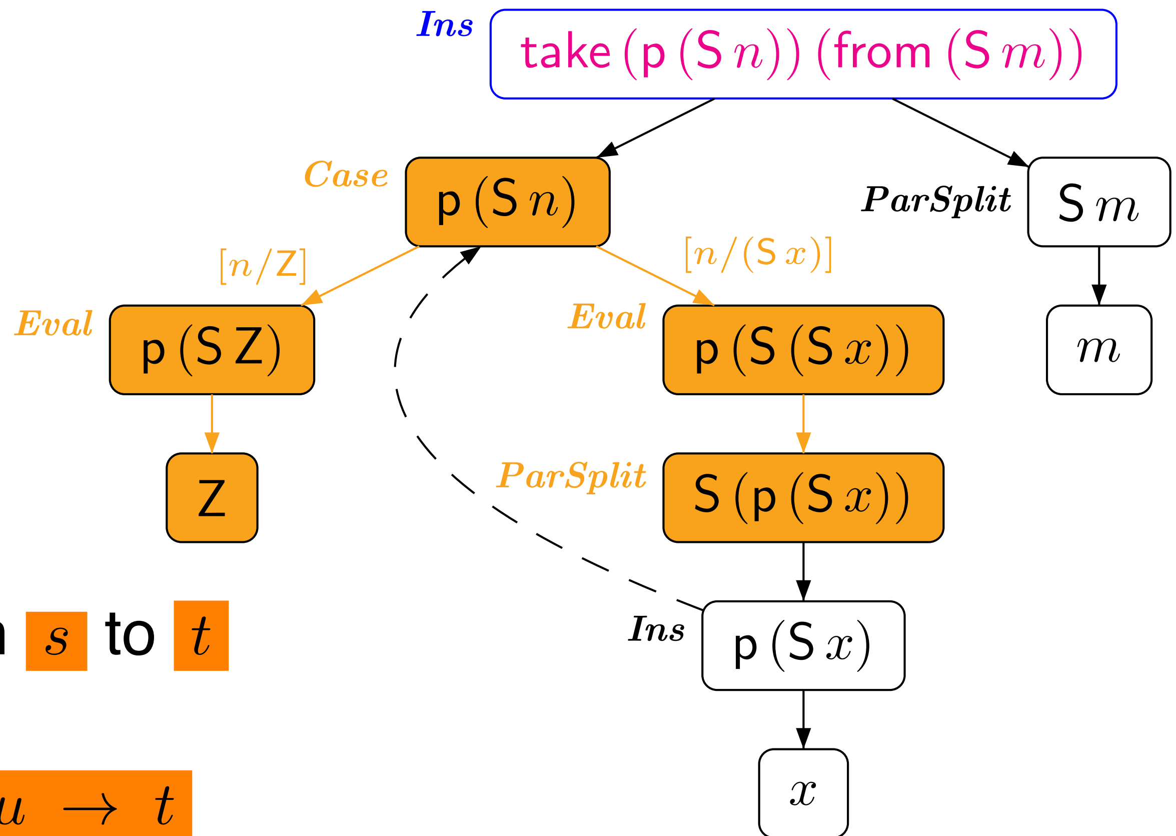
# From Termination Graphs to DP Problems

- **Dependency Pair $\mathcal{P}$:** take (S $n$) (from $m$) $\rightarrow$ take (p (S $n$)) (from (S $m$))

  **Rules $\mathcal{R}$:**

  p (S Z) $\rightarrow$ Z

  p (S (S $x$)) $\rightarrow$ S (p (S $x$))



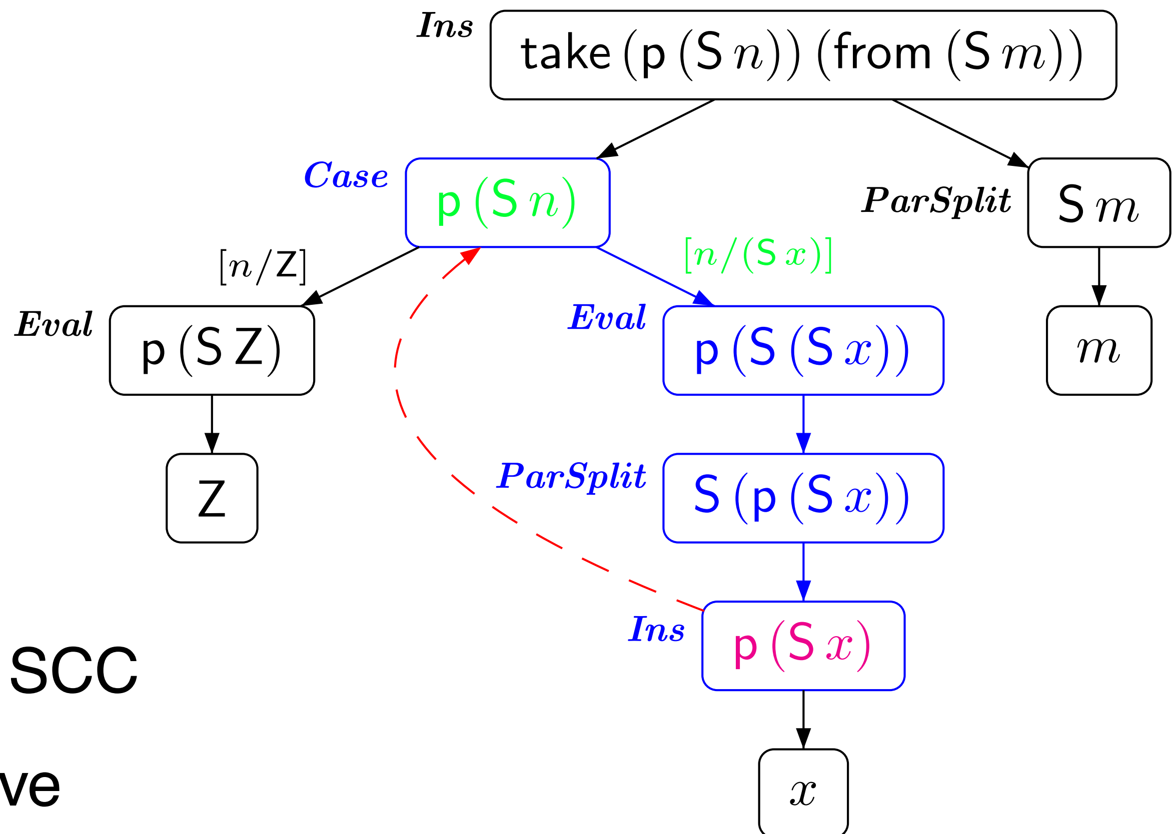

- **Rules**

  if there is a rule path from $s$ to $t$ marked with $\mu$,

  then generate the rule $s\,\mu \rightarrow t$

# From Termination Graphs to DP Problems

- **Dependency Pair $\mathcal{P}$:** $\mathsf{take}\,(\mathsf{S}\,n)\,(\mathsf{from}\,m) \;\to\; \mathsf{take}\,(\mathsf{p}\,(\mathsf{S}\,n))\,(\mathsf{from}\,(\mathsf{S}\,m))$
  **Rules $\mathcal{R}$:** $\mathsf{p}\,(\mathsf{S}\,\mathsf{Z}) \;\to\; \mathsf{Z}$
  $\mathsf{p}\,(\mathsf{S}\,(\mathsf{S}\,x)) \;\to\; \mathsf{S}\,(\mathsf{p}\,(\mathsf{S}\,x))$
- **Dependency Pair $\mathcal{P}$:** $\mathsf{p}\,(\mathsf{S}\,(\mathsf{S}\,x)) \to \mathsf{p}\,(\mathsf{S}\,x)$
  **Rules $\mathcal{R}$:** $\varnothing$




- one DP problem for each SCC
- termination is easy to prove

# Termination of HASKELL-Programs

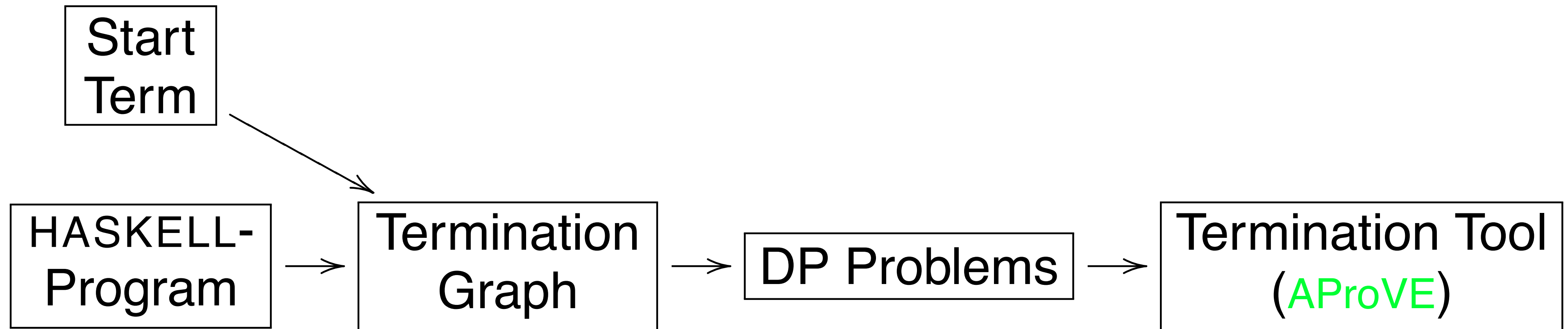
- **New approach in order to use TRS-techniques for HASKELL**
  - generate termination graph for given start term
  - extract DP problems from termination graph
  - prove finiteness of DP problems by existing TRS-techniques

- **Implemented in AProVE**
  - accepts full HASKELL 98 language
  - successfully evaluated with standard HASKELL-libraries

| | FiniteMap | List | Maybe | Monad | Prelude | Queue | Total |
|---|---|---|---|---|---|---|---|
| YES | 256 | 166 | 9 | 69 | 489 | 5 | 994 |
| TOTAL | 321 | 174 | 9 | 80 | 692 | 5 | 1281 |

# Termination of HASKELL-Programs

- **New approach in order to use TRS-techniques for HASKELL**




- **Implemented in AProVE**
  - accepts full HASKELL 98 language
  - successfully evaluated with standard HASKELL-libraries

| | FiniteMap | List | Maybe | Monad | Prelude | Queue | Total |
|---|---|---|---|---|---|---|---|
| YES | 256 | 166 | 9 | 69 | 489 | 5 | 994 |
| TOTAL | 321 | 174 | 9 | 80 | 692 | 5 | 1281 |

# Overview

## I. Termination of Term Rewriting

1. Termination of Term Rewrite Systems
2. Non-Termination of Term Rewrite Systems
3. Complexity of Term Rewrite Systems
4. Termination of Integer Term Rewrite Systems

## II. Termination of Programs

1. Termination of Functional Programs (Haskell)
2. Termination of Logic Programs (Prolog) (PPDP '12)
3. Termination of Imperative Programs (Java)

# Termination of Logic Programming Languages

- well-developed field *(De Schreye & Decorte, 94) etc.*
- **direct approaches:** work directly on the logic program
  - cTI *(Mesnard et al)*
  - TerminWeb *(Codish et al)*
  - TermiLog *(Lindenstrauss et al)*
  - Polytool *(Nguyen, De Schreye, Giesl, Schneider-Kamp)*

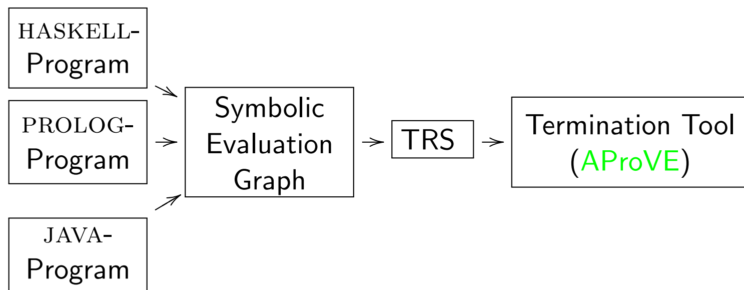
  TRS-techniques can be adapted to work *directly* on the LP
- **transformational approaches:** transform LP to TRS
  - TALP *(Ohlebusch et al)*
  - AProVE *(Giesl et al)*
- only for *definite* LP (without cut)
- not for real PROLOG

# Termination of Logic Programming Languages

- analyzing PROLOG is challenging due to cuts etc.
- **New approach**
  - Frontend
    - evaluate PROLOG a few steps $\Rightarrow$ symbolic evaluation graph
      graph captures evaluation strategy due to cuts etc.
    - transform symbolic evaluation graph $\Rightarrow$ TRS
  - Backend
    - prove termination of the resulting TRS
      (using existing techniques & tools)
- implemented in **AProVE**
  - successfully evaluated on PROLOG-collections with cuts
  - most powerful termination tool for PROLOG
    (winner of *termination competition* for PROLOG)

# Termination of Logic Programming Languages

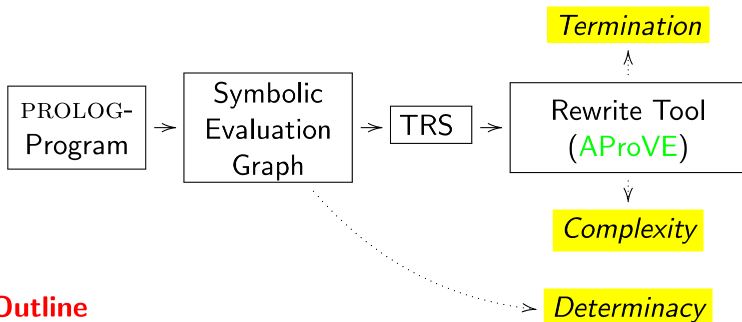
- analyzing PROLOG is challenging due to cuts etc.




- implemented in **AProVE**
  - successfully evaluated on PROLOG-collections with cuts
  - most powerful termination tool for PROLOG (winner of *termination competition* for PROLOG)

# Symbolic Evaluation Graphs and Term Rewriting

**General methodology for analyzing PROLOG programs**




**Outline**

- linear operational semantics of PROLOG
- from PROLOG to symbolic evaluation graphs
- from symbolic evaluation graphs to TRSs for termination analysis
- from symbolic evaluation graphs to TRSs for complexity analysis
- determinacy analysis

$$\begin{array}{rcll} \mathsf{star}(XS, []) & \text{:-} & !. & (1)\\ \mathsf{star}([], ZS) & \text{:-} & !, \mathsf{eq}(ZS, []). & (2)\\ \mathsf{star}(XS, ZS) & \text{:-} & \mathsf{app}(XS, YS, ZS), \mathsf{star}(XS, YS). & (3)\\ \mathsf{app}([], YS, YS). & & & (4)\\ \mathsf{app}([X \mid XS], YS, [X \mid ZS]) & \text{:-} & \mathsf{app}(XS, YS, ZS). & (5)\\ \mathsf{eq}(X, X). & & & (6) \end{array}$$

- $\mathsf{star}(t_1, t_2)$ holds iff $t_2$ results from concatenation of $t_1$ $\quad (t_2 \in (t_1)^*)$
  - $\mathsf{star}([1, 2], [])$ holds
  - $\mathsf{star}([1, 2], [1, 2])$ holds, since $\mathsf{app}([1, 2], [], [1, 2]), \mathsf{star}([1, 2], [])$ hold
  - $\mathsf{star}([1, 2], [1, 2, 1, 2])$ holds, etc.

- cut in clause (2) needed for *termination*. Otherwise:

  $\mathsf{star}([], t)$ would lead to

  $\mathsf{app}([], YS, t), \mathsf{star}([], YS)$ would lead to

  $\mathsf{star}([], t)$

$$
\begin{aligned}
\mathsf{star}(XS, [\,]) &\;\text{:-}\; !. & (1)\\
\mathsf{star}([\,], ZS) &\;\text{:-}\; !, \mathsf{eq}(ZS, [\,]). & (2)\\
\mathsf{star}(XS, ZS) &\;\text{:-}\; \mathsf{app}(XS, YS, ZS), \mathsf{star}(XS, YS). & (3)\\
\mathsf{app}([\,], YS, YS). && (4)\\
\mathsf{app}([X \mid XS], YS, [X \mid ZS]) &\;\text{:-}\; \mathsf{app}(XS, YS, ZS). & (5)\\
\mathsf{eq}(X, X). && (6)
\end{aligned}
$$

- **state:** $(G_1 \mid \ldots \mid G_n)$ with current goal $G_1$ and next goals $G_2, \ldots, G_n$
- **goal:** $(t_1, \ldots, t_k)$ query or
  $(t_1, \ldots, t_k)^c$ query labeled by clause $c$ used for next resolution
- **inference rules:**
  - Case
  - Eval
  - Back
  - Cut
  - Suc

$$
\begin{array}{rl}
\mathsf{star}([1,2],[\,]) & \vdash_{\text{Case}}\\
\mathsf{star}([1,2],[\,])^{(1)} \mid \mathsf{star}([1,2],[\,])^{(2)} \mid \mathsf{star}([1,2],[\,])^{(3)} & \vdash_{\text{Eval}}\\
!_1 \mid \mathsf{star}([1,2],[\,])^{(2)} \mid \mathsf{star}([1,2],[\,])^{(3)} & \vdash_{\text{Cut}}\\
\Box & \vdash_{\text{Suc}}\\
\varepsilon &
\end{array}
$$

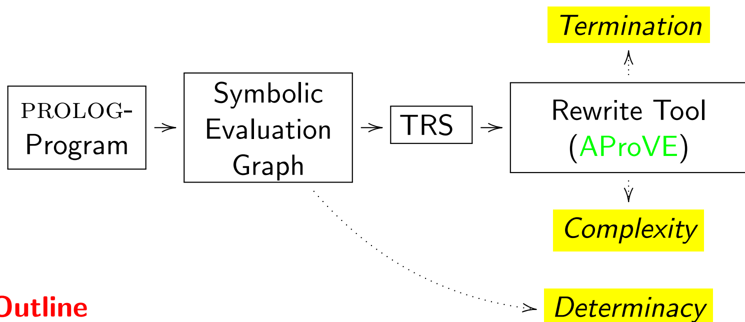
$$
\begin{aligned}
\mathsf{star}(XS, [\,]) &\;\text{:-}\; !. & (1)\\
\mathsf{star}([\,], ZS) &\;\text{:-}\; !, \mathsf{eq}(ZS, [\,]). & (2)\\
\mathsf{star}(XS, ZS) &\;\text{:-}\; \mathsf{app}(XS, YS, ZS), \mathsf{star}(XS, YS). & (3)\\
\mathsf{app}([\,], YS, YS). & & (4)\\
\mathsf{app}([X \mid XS], YS, [X \mid ZS]) &\;\text{:-}\; \mathsf{app}(XS, YS, ZS). & (5)\\
\mathsf{eq}(X, X). & & (6)
\end{aligned}
$$

- **state:** $(G_1 \mid \ldots \mid G_n)$ with current goal $G_1$ and next goals $G_2, \ldots, G_n$
- *linear* semantics, since state contains all backtracking information
  $\Rightarrow$ evaluation is a sequence of states, not a search tree
- suitable for extension to abstract states

$$
\begin{array}{rl}
\mathsf{star}([1,2],[\,]) & \vdash_{\text{CASE}}\\
\mathsf{star}([1,2],[\,])^{(1)} \mid \mathsf{star}([1,2],[\,])^{(2)} \mid \mathsf{star}([1,2],[\,])^{(3)} & \vdash_{\text{EVAL}}\\
!_1 \mid \mathsf{star}([1,2],[\,])^{(2)} \mid \mathsf{star}([1,2],[\,])^{(3)} & \vdash_{\text{CUT}}\\
\square & \vdash_{\text{SUC}}\\
\varepsilon &
\end{array}
$$

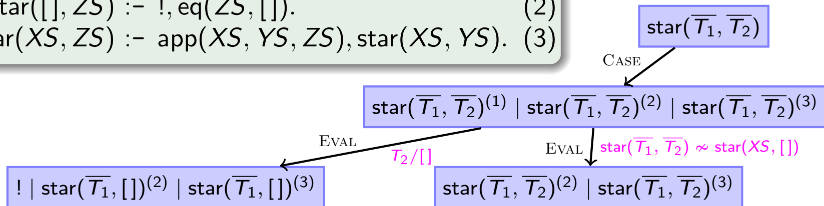
# Symbolic Evaluation Graphs and Term Rewriting

**General methodology for analyzing PROLOG programs**



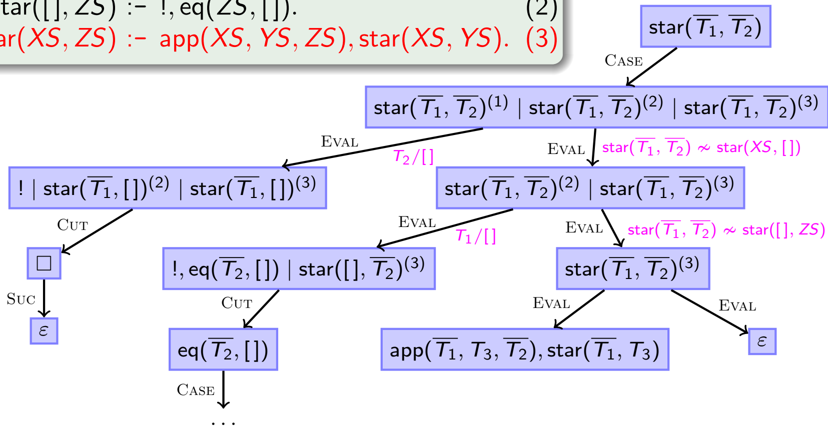

**Outline**

- linear operational semantics of PROLOG
- from PROLOG to symbolic evaluation graphs
- from symbolic evaluation graphs to TRSs for termination analysis
- from symbolic evaluation graphs to TRSs for complexity analysis
- determinacy analysis

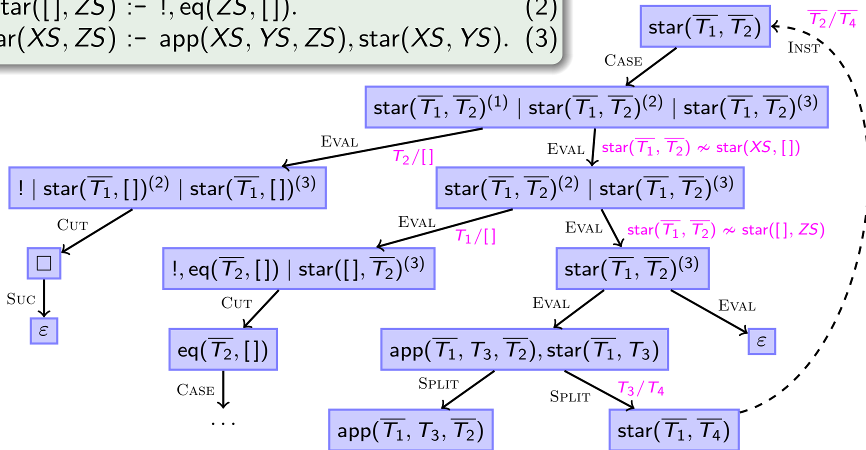



- **symbolic evaluation graph:** all evaluations for a *class* of queries
- **class of queries** $\mathcal{Q}_m^p$ described by *predicate p* and *moding m*

  Example: $\mathcal{Q}_m^{\mathsf{star}} = \{\mathsf{star}(t_1, t_2) \,|\, t_1, t_2 \text{ are ground}\}$.
- **abstract state:** stands for *set* of concrete states
  - state with *abstract* variables $T_1, T_2, \ldots$ representing arbitrary terms
  - constraints on the terms represented by $T_1, T_2, \ldots$
    - groundness constraints: $\overline{T_1}$, $\overline{T_2}$
    - unification constraints: $\mathsf{star}(\overline{T_1}, \overline{T_2}) \not\sim \mathsf{star}(XS, [\,])$



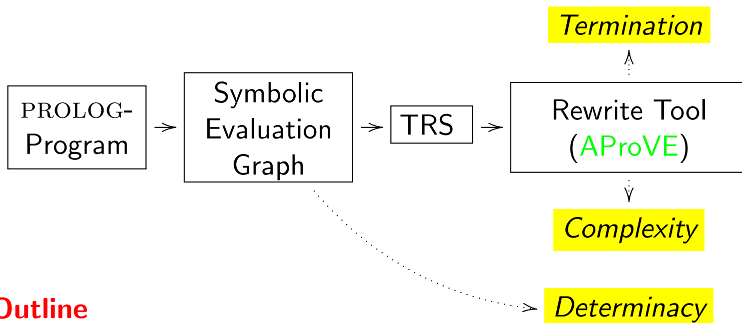

- **abstract state:** stands for *set* of concrete states
  - state with *abstract* variables $T_1, T_2, \ldots$ representing arbitrary terms
  - constraints on the terms represented by $T_1, T_2, \ldots$
    - groundness constraints: $\overline{T_1}, \ \overline{T_2}$
    - unification constraints: $\mathsf{star}(\overline{T_1}, \overline{T_2}) \not\sim \mathsf{star}(XS, [\,])$




- INST: connection to previous state if current state is an *instance*
- SPLIT: split away first atom from a query
  - fresh variables in SPLIT's second successor
  - approximate first atom's answer substitution by *groundness analysis*

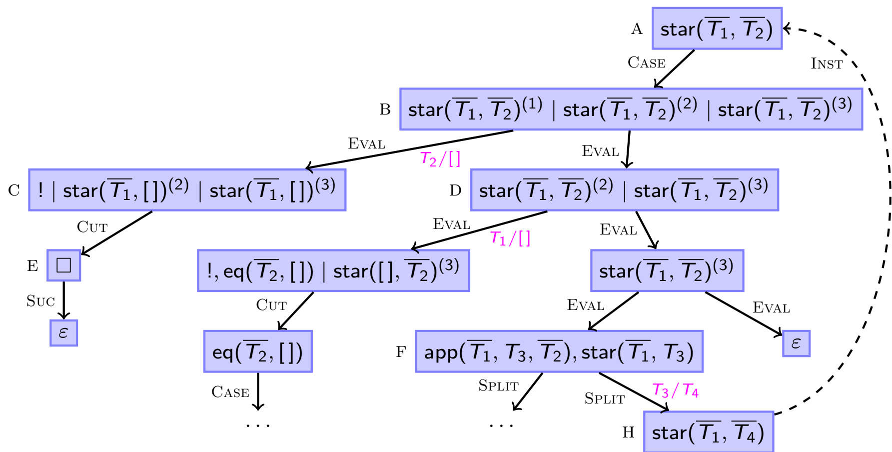
# Symbolic Evaluation Graphs and Term Rewriting

**General methodology for analyzing PROLOG programs**



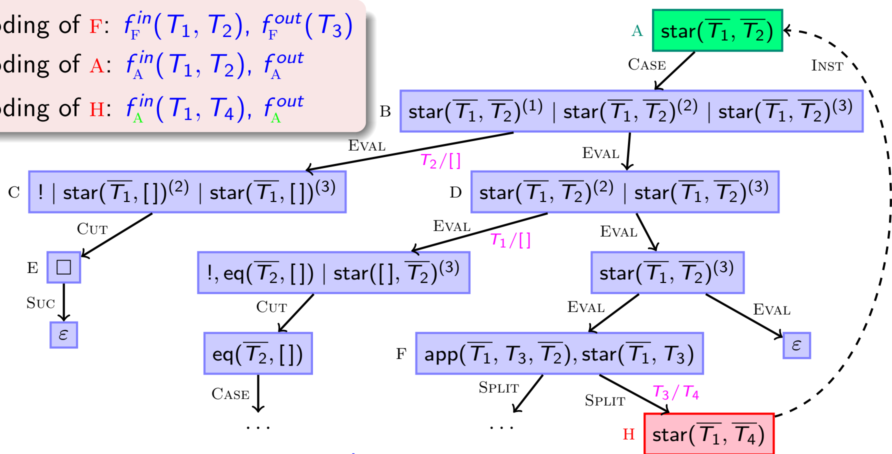

**Outline**

- linear operational semantics of PROLOG
- from PROLOG to symbolic evaluation graphs
- from symbolic evaluation graphs to TRSs for termination analysis
- from symbolic evaluation graphs to TRSs for complexity analysis
- determinacy analysis

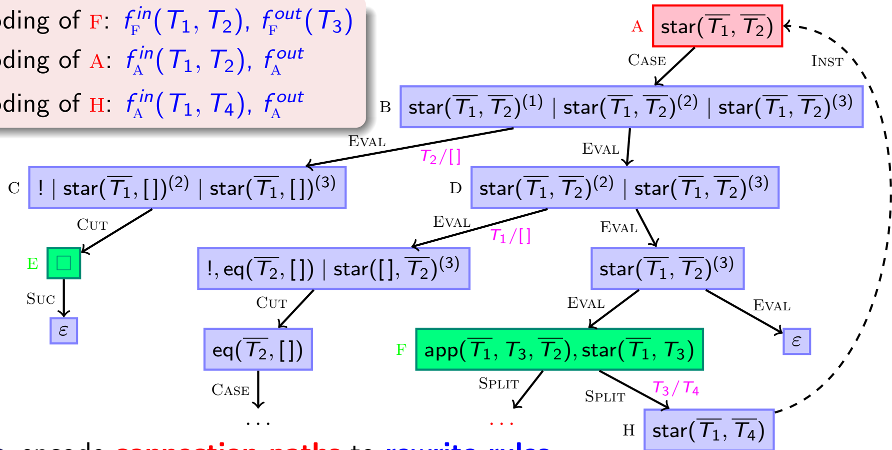



- **Aim:** show termination of concrete states represented by graph
- **Solution:** synthesize TRS from the graph
  - TRS captures all evaluations that are crucial for termination behavior
  - existing rewrite tools can show termination of TRS

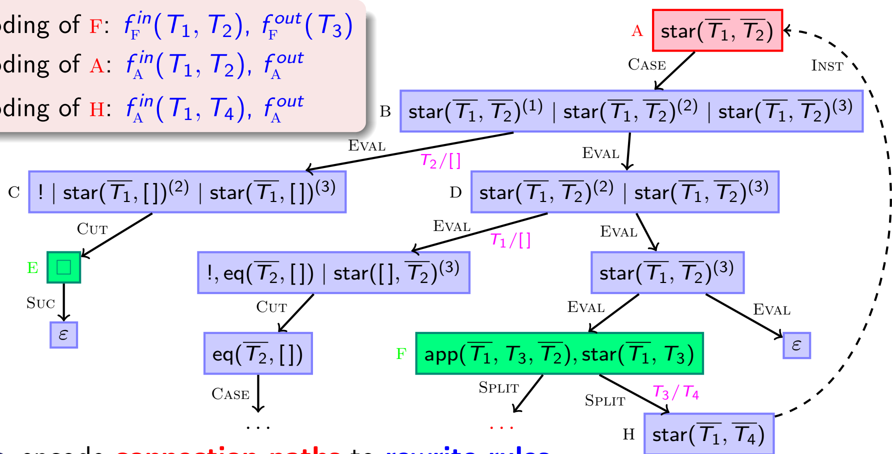
  ⇒ prove termination of original PROLOG program



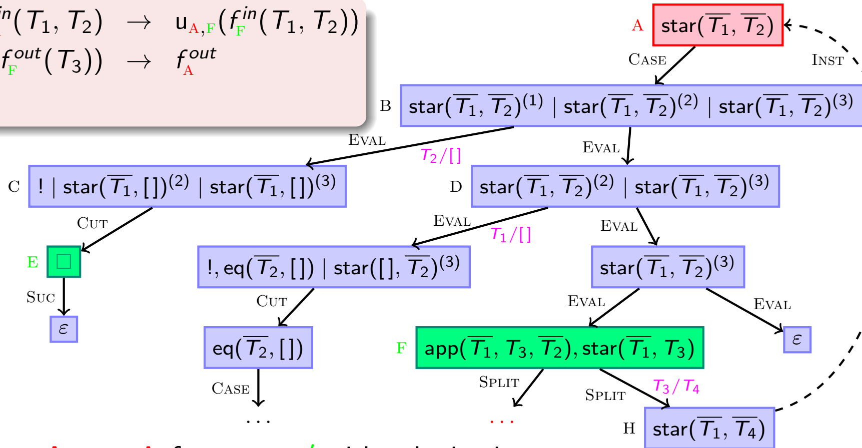

- encode **state** $s$ to **terms** $f_s^{in}(\ldots)$, $f_s^{out}(\ldots)$
  - arguments of $f_s^{in}$: abstract ground variables of $s$ $(\overline{T_1}, \overline{T_2}, \ldots)$
  - arguments of $f_s^{out}$: remaining abstract variables of $s$ which are made ground by every answer substitution of $s$ (*groundness analysis*)
- for state $s$ with Inst edge to $s'$: use $f_{s'}^{in}$, $f_{s'}^{out}$ instead of $f_s^{in}$, $f_s^{out}$




- encode **connection paths** to **rewrite rules**
- **connection path**:
  - start state = root, successor of Inst, or successor of Split but no Inst or Split node itself
  - end state = Inst, Split, Suc node, or successor of Inst node
  - connection path may not traverse end nodes except Suc nodes




- encode **connection paths** to **rewrite rules**
- **connection path**: cover all ways through graph except
  - Inst edges (are covered by the encoding of terms)
  - Split edges (will be covered by extra Split rules later)
  - parts without cycles or Suc nodes (irrelevant for termination behavior)




**connection path** from $s$ to $s'$ with substitution $\sigma$:

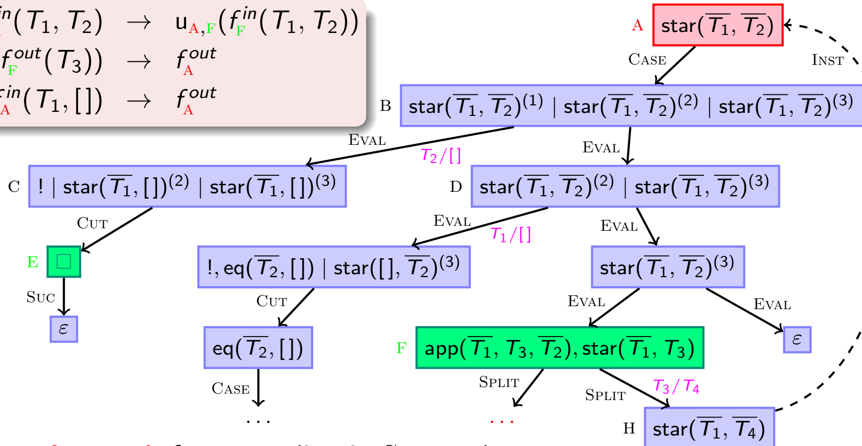
$f_s^{in}(\ldots)\sigma$ evaluates to $f_s^{out}(\ldots)\sigma$ if
$f_{s'}^{in}(\ldots)$ evaluates to $f_{s'}^{out}(\ldots)$

$f_A^{in}(T_1, T_2)$ evaluates to $f_A^{out}$ if
$f_F^{in}(T_1, T_2)$ evaluates to $f_F^{out}(T_3)$

**rewrite rules:**

$$f_s^{in}(\ldots)\sigma \rightarrow u_{s,s'}(\; f_{s'}^{in}(\ldots) \;)$$
$$u_{s,s'}(\; f_{s'}^{out}(\ldots)\;) \rightarrow f_s^{out}(\ldots)\sigma$$

$$f_A^{in}(T_1, T_2) \rightarrow u_{A,F}(\; f_F^{in}(T_1, T_2)\;)$$
$$u_{A,F}(\; f_F^{out}(T_3) \;) \rightarrow f_A^{out}$$




**connection path** from $s$ ending in SUC node:

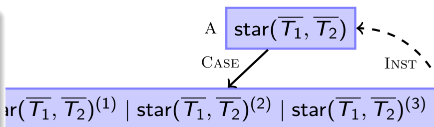
$f_s^{in}(\ldots)\sigma$ evaluates to $f_s^{out}(\ldots)\sigma$ $\qquad$ $f_A^{in}(T_1, [])$ evaluates to $f_A^{out}$

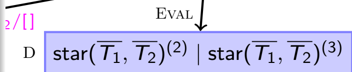
**intuition:**

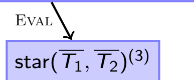
$f_A^{in}(T_1, T_2)$ evaluates to $f_A^{out}$ $\qquad$ if $T_2 \in (T_1)^*$

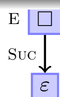
$f_F^{in}(T_1, T_2)$ evaluates to $f_F^{out}(T_3)$ $\qquad$ if $T_1 \neq []$, $T_2 \neq []$, $T_3$ is $T_2$ without prefix $T_1$, $T_3 \in (T_1)^*$

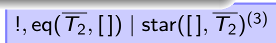
$$f^{in}_{A}(T_1, T_2) \rightarrow u_{A,F}(f^{in}_{F}(T_1, T_2))$$
$$u_{A,F}(f^{out}_{F}(T_3)) \rightarrow f^{out}_{A}$$
$$f^{in}_{A}(T_1, []) \rightarrow f^{out}_{A}$$
$$f^{in}_{F}(T_1, T_2) \rightarrow u_{F,G}(f^{in}_{G}(T_1, T_2))$$
$$u_{F,G}(f^{out}_{G}(T_4)) \rightarrow u_{G,H}(f^{in}_{A}(T_1, T_4), T_4)$$
$$u_{G,H}(f^{out}_{A}, T_4) \rightarrow f^{out}_{F}(T_4)$$

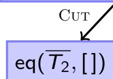


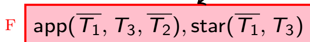



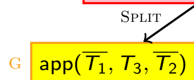



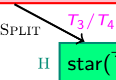





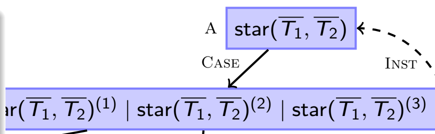



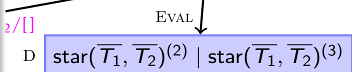

SPLIT node $s$ with successors $s_1$ and $s_1$:

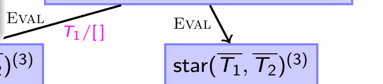
$f^{in}_{s}(\ldots)\sigma$ evaluates to $f^{out}_{s}(\ldots)\sigma$ if
$f^{in}_{s_1}(\ldots)\sigma$ evaluates to $f^{out}_{s_1}(\ldots)\sigma$ and
$f^{in}_{s_2}(\ldots)$ evaluates to $f^{out}_{s_2}(\ldots)$

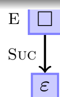
$f^{in}_{F}(T_1, T_2)$ evaluates to $f^{out}_{F}(T_4)$ if
$f^{in}_{G}(T_1, T_2)$ evaluates to $f^{out}_{G}(T_4)$ and
$f^{in}_{A}(T_1, T_4)$ evaluates to $f^{out}_{A}$

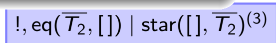
$$
\begin{array}{rcl}
f^{in}_{A}(T_1, T_2) & \to & u_{A,F}(f^{in}_{F}(T_1, T_2)) \\
u_{A,F}(f^{out}_{F}(T_3)) & \to & f^{out}_{A} \\
f^{in}_{A}(T_1, []) & \to & f^{out}_{A} \\
f^{in}_{F}(T_1, T_2) & \to & u_{F,G}(f^{in}_{G}(T_1, T_2)) \\
u_{F,G}(f^{out}_{G}(T_4)) & \to & u_{G,H}(f^{in}_{A}(T_1, T_4), T_4) \\
u_{G,H}(f^{out}_{A}, T_4) & \to & f^{out}_{F}(T_4)
\end{array}
$$

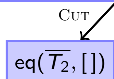


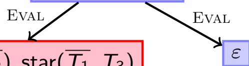



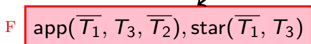



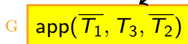



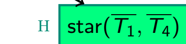











**intuition:**

$f^{in}_{F}(T_1, T_2)$ evaluates to $f^{out}_{F}(T_4)$ if $T_1 \neq []$, $T_2 \neq []$, $T_4$ is $T_2$ without prefix $T_1$, $T_4 \in (T_1)^*$

$f^{in}_{G}(T_1, T_2)$ evaluates to $f^{out}_{G}(T_4)$ if $T_1 \neq []$, $T_2 \neq []$, $T_4$ is $T_2$ without prefix $T_1$

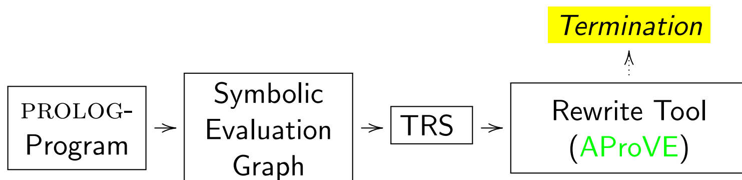
$f^{in}_{A}(T_1, T_4)$ evaluates to $f^{out}_{A}$ if $T_4 \in (T_1)^*$

```
star(XS, []) :- !.
star([], ZS) :- !, eq(ZS, []).
star(XS, ZS) :- app(XS, YS, ZS), star(XS, YS).
app([], YS, YS).
app([X | XS], YS, [X | ZS]) :- app(XS, YS, ZS).
eq(X, X).
```

$$
\begin{aligned}
f^{in}_{A}(T_1, T_2) &\rightarrow u_{A,F}(f^{in}_{F}(T_1, T_2)) \\
u_{A,F}(f^{out}_{F}(T_3)) &\rightarrow f^{out}_{A} \\
f^{in}_{A}(T_1, []) &\rightarrow f^{out}_{A} \\
f^{in}_{F}(T_1, T_2) &\rightarrow u_{F,G}(f^{in}_{G}(T_1, T_2)) \\
u_{F,G}(f^{out}_{G}(T_4)) &\rightarrow u_{G,H}(f^{in}_{A}(T_1, T_4), T_4) \\
u_{G,H}(f^{out}_{A}, T_4) &\rightarrow f^{out}_{F}(T_4) \\
f^{in}_{G}([T_5 \mid T_6], [T_5 \mid T_7]) &\rightarrow u_{G,I}(f^{in}_{I}(T_6, T_7)) \\
u_{G,I}(f^{out}_{I}(T_3)) &\rightarrow f^{out}_{G}(T_3) \\
f^{in}_{I}([T_8 \mid T_9], [T_8 \mid T_{10}]) &\rightarrow u_{I,K}(f^{in}_{I}(T_9, T_{10})) \\
u_{I,K}(f^{out}_{I}(T_3)) &\rightarrow f^{out}_{I}(T_3) \\
f^{in}_{I}([], T_3) &\rightarrow f^{out}_{I}(T_3)
\end{aligned}
$$

- existing TRS tools prove termination automatically
- original PROLOG program terminates

# Symbolic Evaluation Graphs and Term Rewriting

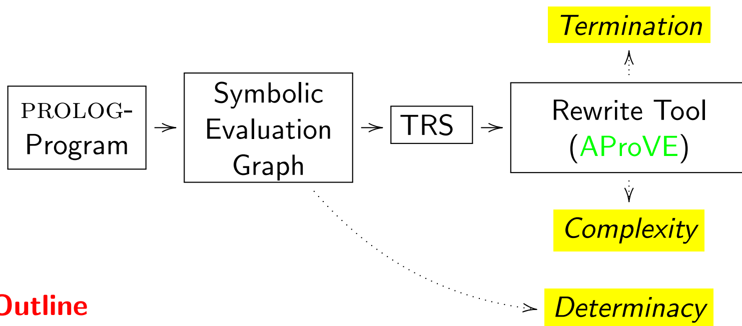



implemented in tool **AProVE**

- most powerful tool for termination of definite logic programs
- only tool for termination of non-definite PROLOG programs
- winner of *termination competition* for PROLOG
  (proves 342 of 477 examples, average runtime 6.5 s per example)

# Symbolic Evaluation Graphs and Term Rewriting

**General methodology for analyzing PROLOG programs**



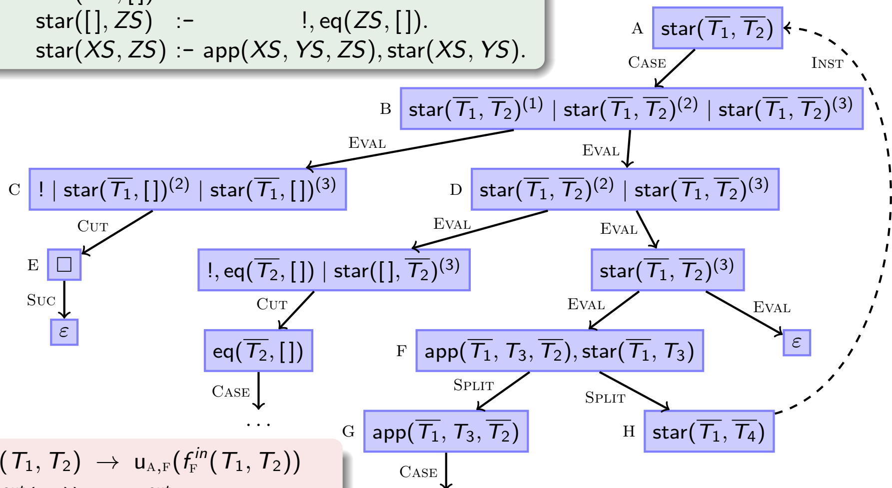

**Outline**

- linear operational semantics of PROLOG
- from PROLOG to symbolic evaluation graphs
- from symbolic evaluation graphs to TRSs for termination analysis
- from symbolic evaluation graphs to TRSs for complexity analysis
- determinacy analysis

# Complexity for Logic Programs

**Program $\mathcal{P}$, Class of queries $\mathcal{Q}_m^{\mathsf{p}}$**

- $prc_{\mathcal{P},\mathcal{Q}_m^{\mathsf{p}}}$ maps $n \in \mathbb{N}$ to longest evaluation starting with $Q \in \mathcal{Q}_m^{\mathsf{p}}$, where $|Q|_m \leq n$
- $|Q|_m$: number of variables and function symbols on *input positions*
- corresponds to number of unification attempts
- $\mathcal{R}$ has linear complexity for class $\mathcal{Q}_m^{\mathsf{p}}$ if $prc_{\mathcal{P},\mathcal{Q}_m^{\mathsf{p}}}(n) \in \mathcal{O}(n)$
  $\mathcal{R}$ has quadratic complexity for class $\mathcal{Q}_m^{\mathsf{p}}$ if $prc_{\mathcal{P},\mathcal{Q}_m^{\mathsf{p}}}(n) \in \mathcal{O}(n^2)$ etc.
- Example (star-program): has linear complexity
- Goal: Re-use existing methodology for termination analysis
  to analyze complexity as well




$$
\begin{aligned}
f^{in}_{\mathrm{A}}(T_1, T_2) &\rightarrow \mathsf{u}_{\mathrm{A,F}}(f^{in}_{\mathrm{F}}(T_1, T_2)) \\
\mathsf{u}_{\mathrm{A,F}}(f^{out}_{\mathrm{F}}(T_3)) &\rightarrow f^{out}_{\mathrm{A}} \\
f^{in}_{\mathrm{A}}(T_1, []) &\rightarrow f^{out}_{\mathrm{A}} \\
f^{in}_{\mathrm{F}}(T_1, T_2) &\rightarrow \mathsf{u}_{\mathrm{F,G}}(f^{in}_{\mathrm{G}}(T_1, T_2)) \\
\mathsf{u}_{\mathrm{F,G}}(f^{out}_{\mathrm{G}}(T_4)) &\rightarrow \mathsf{u}_{\mathrm{G,H}}(f^{in}_{\mathrm{A}}(T_1, T_4), T_4) \\
\mathsf{u}_{\mathrm{G,H}}(f^{out}_{\mathrm{A}}, T_4) &\rightarrow f^{out}_{\mathrm{F}}(T_4)
\end{aligned}
$$

- generate symbolic evaluation graph
- generate TRS from graph
- determine complexity of TRS by existing tool

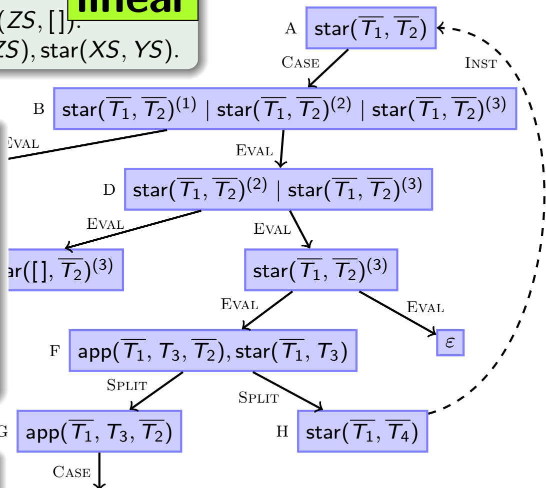
$\mathcal{P}$ :

```
star(XS, [])  :-                     !.
star([], ZS)  :-              !, eq(ZS, []).
star(XS, ZS)  :- app(XS, YS, ZS), star(XS, YS).
```

**linear**

## Correct!

- depends on SPLIT's successor G
- in $\mathcal{P}$: repeat evaluation of H for every answer of G (*backtracking*)
- in TRS: evaluate H once (choose G's answer *non-deterministically*)
- Here: G is *deterministic* (has only one answer)




$$
\begin{aligned}
f_{\mathrm{A}}^{in}(T_1, T_2) &\to \mathrm{u}_{\mathrm{A,F}}(f_{\mathrm{F}}^{in}\ldots \\
\mathrm{u}_{\mathrm{A,F}}(f_{\mathrm{F}}^{out}(T_3)) &\to f_{\mathrm{A}}^{out} \\
f_{\mathrm{A}}^{in}(T_1, []) &\to f_{\mathrm{A}}^{out} \\
f_{\mathrm{F}}^{in}(T_1, T_2) &\to \mathrm{u}_{\mathrm{F,G}}(f_{\mathrm{G}}^{in}(T_1, T_2)) \\
\mathrm{u}_{\mathrm{F,G}}(f_{\mathrm{G}}^{out}(T_4)) &\to \mathrm{u}_{\mathrm{G,H}}(f_{\mathrm{A}}^{in}(T_1, T_4), T_4) \\
\mathrm{u}_{\mathrm{G,H}}(f_{\mathrm{A}}^{out}, T_4) &\to f_{\mathrm{F}}^{out}(T_4)
\end{aligned}
$$

**linear**

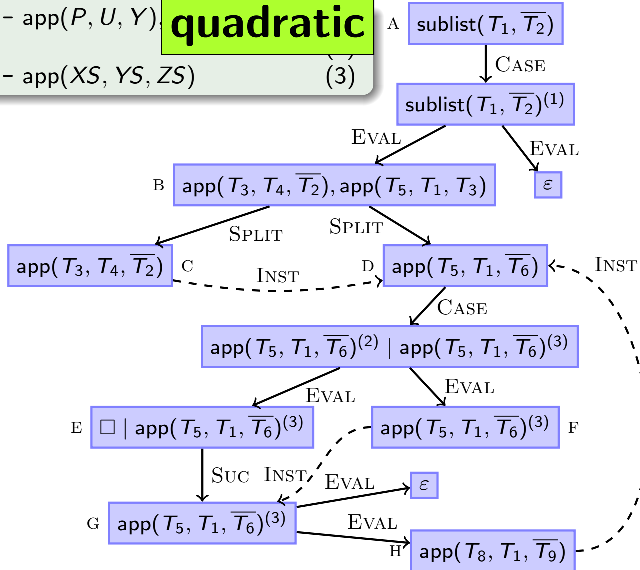
- generate symbolic evaluation graph
- generate TRS from graph
- determine complexity of TRS by existing tool
- infer that $\mathcal{P}$ has the same complexity

$$
\begin{aligned}
\mathcal{P}: \quad \text{sublist}(X, Y) &\text{ :- } \text{app}(P, U, Y),\ \text{app}(V, X, P). && (1)\\
\text{app}([\,], YS, YS). & && (2)\\
\text{app}([X \mid XS], YS, [X \mid ZS]) &\text{ :- } \text{app}(XS, YS, ZS) && (3)
\end{aligned}
$$

## Evaluation of sublist

- $\mathcal{Q}_m^{\text{sublist}} = \{\text{sublist}(t_1, t_2) \mid t_2 \text{ ground}\}$
- computes all sublists of $Y$ (by *backtracking*)
- $\mathcal{P}$:
  - linear many possibilities to split $Y$ into $P$ and $U$
  - for each possible $P$, linear evaluation of $\text{app}(V, X, P)$

$\mathcal{P}$ : sublist($X$, $Y$) :- app($P$, $U$, $Y$), **quadratic**
app([ ], $YS$, $YS$).
app([$X$ | $XS$], $YS$, [$X$ | $ZS$]) :- app($XS$, $YS$, $ZS$) (3)




$f_{\mathrm{B}}^{in}(T_2) \rightarrow$ u **linear**

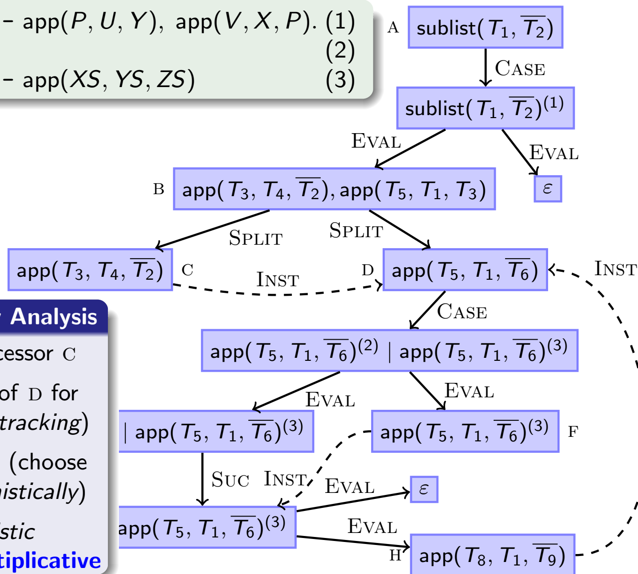
$\mathrm{u}_{\mathrm{B,C}}(f_{\mathrm{D}}^{out}(\dots)) \rightarrow \mathrm{u}_{\mathrm{C,D}}(f_{\mathrm{D}}^{in}(\dots))$

$\mathrm{u}_{\mathrm{C,D}}(f_{\mathrm{D}}^{out}(\dots)) \rightarrow f_{\mathrm{B}}^{out}(\dots)$

- generate symbolic evaluation graph and TRS
- determine complexity of TRS by existing tool
- ~~infer that $\mathcal{P}$ has the same complexity~~

$\mathcal{P}$ :

$$\begin{array}{lr} \text{sublist}(X, Y) \text{ :- } \text{app}(P, U, Y),\ \text{app}(V, X, P). & (1) \\ \text{app}([\,], YS, YS). & (2) \\ \text{app}([X \,|\, XS], YS, [X \,|\, ZS]) \text{ :- } \text{app}(XS, YS, ZS) & (3) \end{array}$$




## Correctness of Complexity Analysis

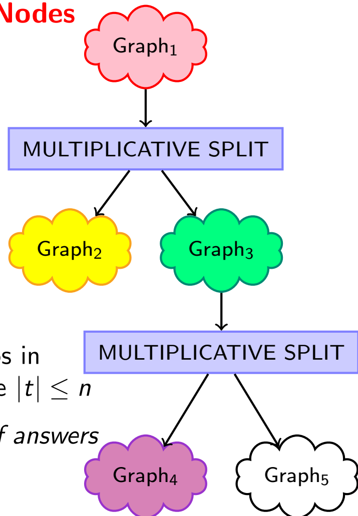
- depends on Split's successor C
- in $\mathcal{P}$: repeat evaluation of D for every answer of C (*backtracking*)
- in TRS: evaluate D once (choose C's answer *non-deterministically*)
- Here: C is *not deterministic* $\Rightarrow$ Split node B is **multiplicative**

$$\begin{array}{rcl} f^{in}_{B}(T_2) & \rightarrow & u_{B,C}(f^{in}_{D}(T_2)) \\ u_{B,C}(f^{out}_{D}(\ldots)) & \rightarrow & u_{C,D}(f^{in}_{D}(\ldots)) \\ u_{C,D}(f^{out}_{D}(\ldots)) & \rightarrow & f^{out}_{B}(\ldots) \end{array}$$

- generate symbolic evaluation graph and TRS
- determine complexity of TRS by existing tool
- ~~infer that $\mathcal{P}$ has the same complexity~~

# Decompose Graph by Multiplicative Split Nodes

- generate symbolic evaluation graph
- generate separate TRSs $\mathcal{R}_1, \ldots, \mathcal{R}_5$ for parts up to multiplicative SPLIT nodes (no multiplicative SPLIT node may reach itself)
- determine $irc_{\mathcal{R}_1,\mathcal{R}}$, ..., $irc_{\mathcal{R}_5,\mathcal{R}}$ separately
  - maps $n \in \mathbb{N}$ to maximal number of $\mathcal{R}_i$-steps in evaluation starting with basic term $t$, where $|t| \leq n$
  - upper bound for *runtime* and for *number of answers*
- combine complexities
  - multiply complexities for children of multiplicative SPLITs
  - add complexities of parents of multiplicative SPLITs
  - $irc_{\mathcal{R}_1,\mathcal{R}} + irc_{\mathcal{R}_2,\mathcal{R}} \cdot (irc_{\mathcal{R}_3,\mathcal{R}} + irc_{\mathcal{R}_4,\mathcal{R}} \cdot irc_{\mathcal{R}_5,\mathcal{R}})$



- $irc_{\mathcal{R}_1,\mathcal{R}}$: constant
- $irc_{\mathcal{R}_2,\mathcal{R}}$: linear
- $irc_{\mathcal{R}_3,\mathcal{R}}$: linear
- complexity of $\mathcal{P}$: quadratic
  $irc_{\mathcal{R}_1,\mathcal{R}} + irc_{\mathcal{R}_2,\mathcal{R}} \cdot irc_{\mathcal{R}_3,\mathcal{R}}$

$$
\begin{array}{rcl}
f^{in}_{\mathrm{A}}(T_2) & \to & u_{\mathrm{A,B}}(f^{in}_{\mathrm{B}}(T_2)) \\
u_{\mathrm{A,B}}(f^{out}_{\mathrm{B}}(\ldots)) & \to & f^{out}_{\mathrm{A}}(T_1) \\
f^{in}_{\mathrm{B}}(T_2) & \to & u_{\mathrm{B,C}}(f^{in}_{\mathrm{D}}(T_2)) \\
u_{\mathrm{B,C}}(f^{out}_{\mathrm{D}}(\ldots)) & \to & u_{\mathrm{C,D}}(f^{in}_{\mathrm{D}}(\ldots)) \\
u_{\mathrm{C,D}}(f^{out}_{\mathrm{D}}(\ldots)) & \to & f^{out}_{\mathrm{B}}(\ldots) \\
f^{in}_{\mathrm{D}}(T_6) & \to & f^{out}_{\mathrm{D}}([\,], T_6) \\
f^{in}_{\mathrm{D}}(T_6) & \to & u_{\mathrm{D,G}}(f^{in}_{\mathrm{G}}(T_6)) \\
u_{\mathrm{D,G}}(f^{out}_{\mathrm{G}}(\ldots)) & \to & f^{out}_{\mathrm{D}}(T_5, T_1) \\
f^{in}_{\mathrm{D}}(T_6) & \to & u_{\mathrm{D,F}}(f^{in}_{\mathrm{G}}(T_6)) \\
u_{\mathrm{D,F}}(\ldots) & \to & f^{out}_{\mathrm{D}}(T_5, T_1) \\
f^{in}_{\mathrm{G}}([T_7 \mid T_9]) & \to & u_{\mathrm{G,H}}(\ldots) \\
u_{\mathrm{G,H}}(\ldots) & \to & f^{out}_{\mathrm{G}}([T_7 \mid T_8])
\end{array}
$$

A: sublist$(T_1, \overline{T_2})$

Case

sublist$(T_1, \overline{T_2})^{(1)}$

Eval, Eval

$\varepsilon$

B: app$(T_3, T_4, \overline{T_2})$, app$(T_5, T_1, T_3)$

Split, Split

C: app$(T_3, T_4, \overline{T_2})$

Inst

D: app$(T_5, T_1, \overline{T_6})$

Inst

Case

app$(T_5, T_1, \overline{T_6})^{(2)}$ | app$(T_5, T_1, \overline{T_6})^{(3)}$

Eval, Eval

E: $\square$ | app$(T_5, T_1, \overline{T_6})^{(3)}$

F: app$(T_5, T_1, \overline{T_6})^{(3)}$

Suc, Inst, Eval

$\varepsilon$

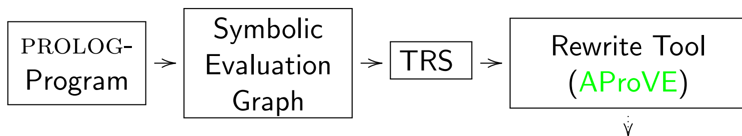
G: app$(T_5, T_1, \overline{T_6})^{(3)}$

Eval

H: app$(T_8, T_1, \overline{T_9})$

- generate graph and TRSs $\mathcal{R}_1$, $\mathcal{R}_2$, $\mathcal{R}_3$
- determine $irc_{\mathcal{R}_1,\mathcal{R}}$, $irc_{\mathcal{R}_2,\mathcal{R}}$, $irc_{\mathcal{R}_3,\mathcal{R}}$
- infer complexity of $\mathcal{P}$

# Symbolic Evaluation Graphs and Term Rewriting



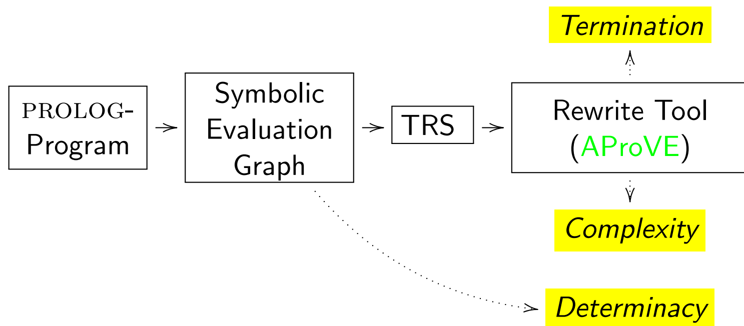

implemented in tool **AProVE**

- only tool for complexity of non-well-moded or non-definite programs
- experiments on all 477 programs of *TPDB*

| | $\mathcal{O}(1)$ | $\mathcal{O}(n)$ | $\mathcal{O}(n^2)$ | $\mathcal{O}(n \cdot 2^n)$ | **bounds** | **time** |
|---|---|---|---|---|---|---|
| CASLOG | 1 | 21 | 4 | **3** | 29 | 14.8 |
| CiaoPP | 3 | 19 | 4 | **3** | 29 | 11.7 |
| AProVE | **54** | **117** | **37** | 0 | **208** | **10.6** |

# Symbolic Evaluation Graphs and Term Rewriting

## General methodology for analyzing PROLOG programs




## Outline

- linear operational semantics of PROLOG
- from PROLOG to symbolic evaluation graphs
- from symbolic evaluation graphs to TRSs for termination analysis
- from symbolic evaluation graphs to TRSs for complexity analysis
- determinacy analysis

**Criterion for determinacy of $s$**

If $s$ reaches Suc node $s'$, then there is no path from $s'$ to a Suc node.

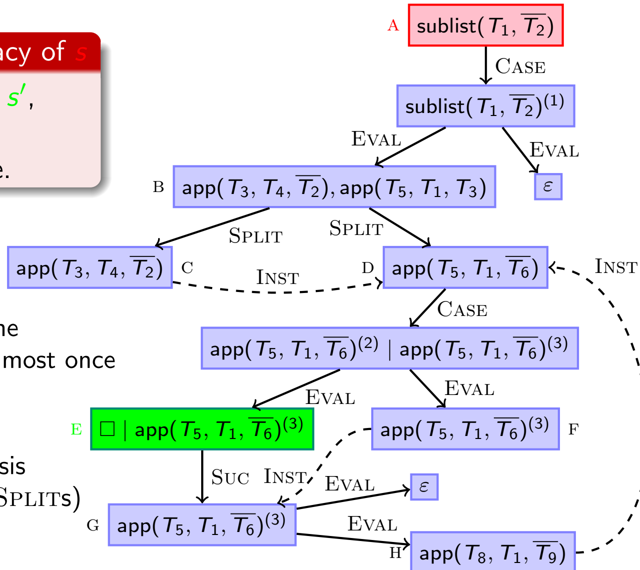
- query deterministic iff

  it generates at most one answer substitution at most once

  - for program analysis
  - for complexity analysis (non-multiplicative Splits)

- successful evaluation ⇒

  path to Suc node in symbolic evaluation graph



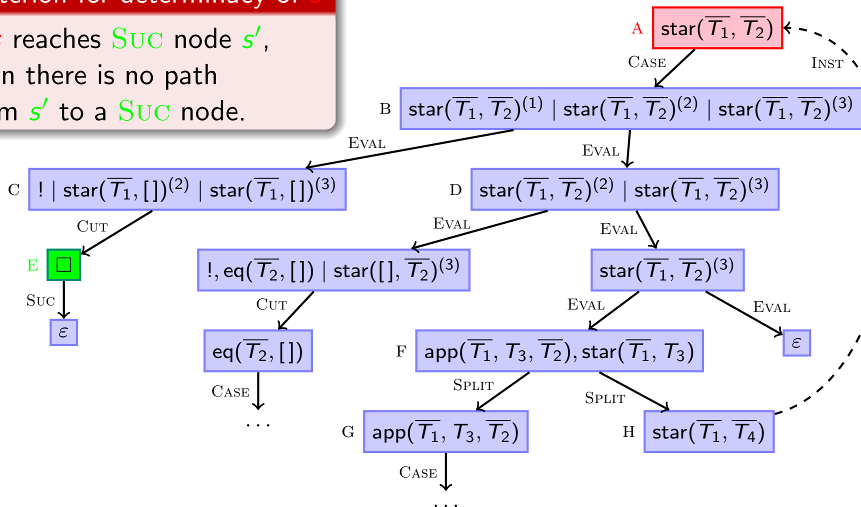

- C not deterministic

  ⇒ Split node B multiplicative

- A not deterministic

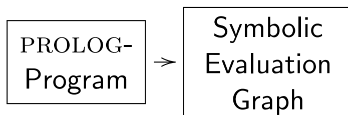
## Criterion for determinacy of $s$

If $s$ reaches Suc node $s'$, then there is no path from $s'$ to a Suc node.




- G is deterministic
  $\Rightarrow$ Split node F not multiplicative
- A is deterministic

# Symbolic Evaluation Graphs and Term Rewriting




*Determinacy*

implemented in tool **AProVE**

- experiments on 300 definite programs:
  CiaoPP: 132, AProVE: 69
- experiments on 177 non-definite programs:
  CiaoPP: 61, AProVE: 92
- only first step, but substantial addition to existing determinacy analyses
  (AProVE succeeds on 78 examples where CiaoPP fails)
- strong enough for complexity analysis

# Overview

## I. Termination of Term Rewriting

1. Termination of Term Rewrite Systems
2. Non-Termination of Term Rewrite Systems
3. Complexity of Term Rewrite Systems
4. Termination of Integer Term Rewrite Systems

## II. Termination of Programs

1. Termination of Functional Programs (Haskell)
2. Termination of Logic Programs (Prolog)
3. Termination of Imperative Programs (Java) (RTA '10 & '11, CAV '12)

# Termination of Imperative Programs

**Direct Approaches**

- Synthesis of Linear Ranking Functions
  *(Colon & Sipma, 01), (Podelski & Rybalchenko, 04), . . .*
- Terminator: Termination Analysis by Abstraction & Model Checking
  *(Cook, Podelski, Rybalchenko et al., since 05)*
- Julia & COSTA: Termination Analysis of JAVA BYTECODE
  *(Spoto, Mesnard, Payet, 10),*
  *(Albert, Arenas, Codish, Genaim, Puebla, Zanardini, 08)*
- . . .

- used at Microsoft for verifying Windows device drivers
- no use of TRS-techniques (stand-alone methods)

# Termination of Imperative Programs

## Rewrite-Based Approach

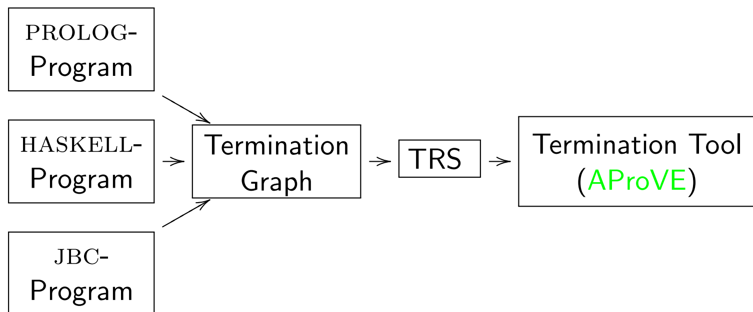
- analyze JAVA BYTECODE (JBC) instead of JAVA
- using TRS-techniques for JBC is challenging
  - sharing and aliasing
  - side effects
  - cyclic data objects
  - object-orientation
  - recursion
  - ...

# Termination of Imperative Programs

- **New approach**
  - Frontend
    - evaluate JBC a few steps ⇒ termination graph

      termination graph captures side effects, sharing, cyclic data objects etc.
    - transform termination graph ⇒ TRS
  - Backend
    - prove termination of the resulting TRS
      (using existing techniques & tools)
- implemented in **AProVE**
  - successfully evaluated on JBC-collection
  - competitive termination tool for JBC

# Termination of Imperative Programs




- implemented in **AProVE**
  - successfully evaluated on JBC-collection
  - competitive termination tool for JBC

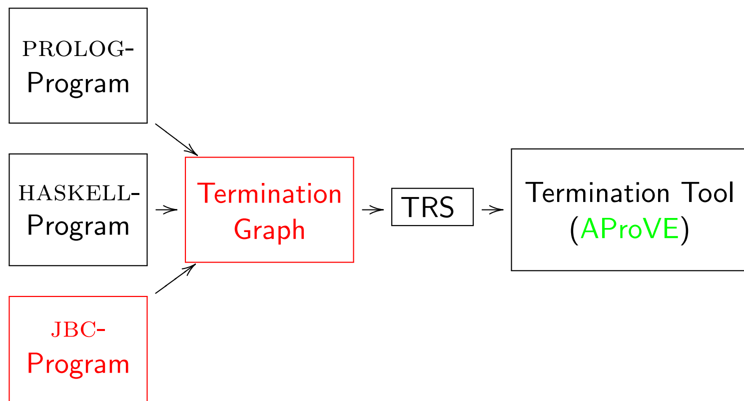
# Termination of Imperative Programs

- other techniques:
  abstract objects to numbers
  - `IntList`-object representing $[0, 1, 2]$
    is abstracted to length 3

```
public class IntList {
  int value;
  IntList next;
}
```

- our technique:
  abstract objects to terms
  - introduce function symbol for every class
  - `IntList`-object representing $[0, 1, 2]$
    is abstracted to term: $\mathsf{IntList}(0,\ \mathsf{IntList}(1,\ \mathsf{IntList}(2, \mathsf{null})\,)\,)$
  - TRS-techniques generate suitable orders to compare arbitrary terms
  - particularly powerful on user-defined data types
  - powerful on pre-defined data types by using Integer TRSs

# From JBC to Termination Graphs



# Example

```
00: aload_0       // load num to opstack
01: ifnull 8      // jump to line 8 if top
                  // of opstack is null
04: aload_1       // load limit
05: ifnonnull 9   // jump if not null
08: return
09: aload_0       // load num
10: astore_2      // store into copy
11: aload_0       // load num
12: getfield val  // load field val
15: aload_1       // load limit
16: getfield val  // load field val
19: if_icmpge 35  // jump if
                  // num.val >= limit.val
22: aload_2       // load copy
23: aload_2       // load copy
24: getfield val  // load field val
27: iconst_1      // load constant 1
28: iadd          // add copy.val and 1
29: putfield val  // store into copy.val
32: goto 11
35: return
```

```
public class Int {
  // only wrap a primitive int
  private int val;

  // count up to the value
  // in "limit"
  public static void count(
      Int num, Int limit) {

    if (num == null
        || limit == null) {
        return;
    }

    // introduce sharing
    Int copy = num;

    while (num.val < limit.val) {
        copy.val++;
    }
  }
}
```

# Abstract States of the JVM

```
00: aload_0        // load num to opstack
01: ifnull 8       // jump to line 8 if top
                   // of opstack is null
04: aload_1        // load limit
05: ifnonnull 9    // jump if not null
08: return
09: aload_0        // load num
10: astore_2       // store into copy
11: aload_0        // load num
12: getfield val   // load field val
15: aload_1        // load limit
16: getfield val   // load field val
19: if_icmpge 35   // jump if
                   // num.val >= limit.val
22: aload_2        // load copy
23: aload_2        // load copy
24: getfield val   // load field val
27: iconst_1       // load constant 1
28: iadd           // add copy.val and 1
29: putfield val   // store into copy.val
32: goto 11
35: return
```

| ifnull 8 $\mid$ n : $o_1$, l : $o_2$ $\mid$ $o_1$ |
|---|
| $o_1 = \texttt{Int}(\texttt{val} = i_1) \quad i_1 = (-\infty, \infty)$ |
| $o_2 = \texttt{Int}(?)$ |

**4 components**

1. next program instruction
2. values of local variables (value of `num` is *reference* $o_1$)
3. values on the operand stack
4. information about the heap
   - object at address $o_2$ is `null` or of type `Int`
   - object at $o_1$ has type `Int`, `val`-field has value $i_1$
   - $i_1$ is an arbitrary integer
   - no sharing

# From JBC to Termination Graphs

```
00: aload_0
01: ifnull 8
04: aload_1
     :
19: if_icmpge 35
     :
27: iconst_1
28: iadd
29: putfield val
32: goto 11
35: return
```

$A$

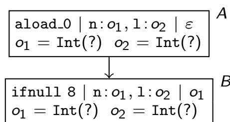
| aload_0 \| n: $o_1$, l: $o_2$ \| $\varepsilon$ |
|---|
| $o_1$ = Int(?) $o_2$ = Int(?) |

**State $A$:**

- do all calls of count terminate?
- num and limit are arbitrary, but distinct Int-objects

# From JBC to Termination Graphs

```
00: aload_0
01: ifnull 8
04: aload_1
      ⋮
19: if_icmpge 35
      ⋮
27: iconst_1
28: iadd
29: putfield val
32: goto 11
35: return
```

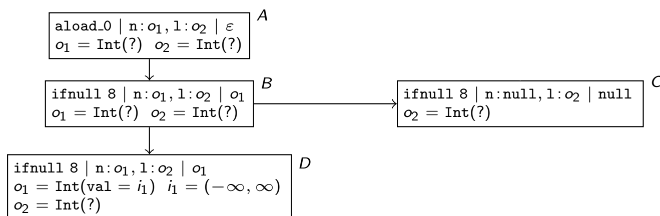



**State *B*:**

- "aload_0" loads value of num on operand stack
- *A* connected to *B* by *evaluation edge*

# From JBC to Termination Graphs

```
00: aload_0
01: ifnull 8
04: aload_1
     ⋮
19: if_icmpge 35
     ⋮
27: iconst_1
28: iadd
29: putfield val
32: goto 11
35: return
```

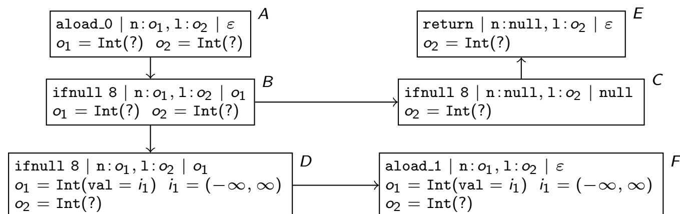



**States $C$ and $D$:**

- "ifnull 8" needs to know whether $o_1$ is null
- *refine* information about heap (*refinement edges*)

# From JBC to Termination Graphs

```
00: aload_0
01: ifnull 8
04: aload_1
     :
19: if_icmpge 35
     :
27: iconst_1
28: iadd
29: putfield val
32: goto 11
35: return
```

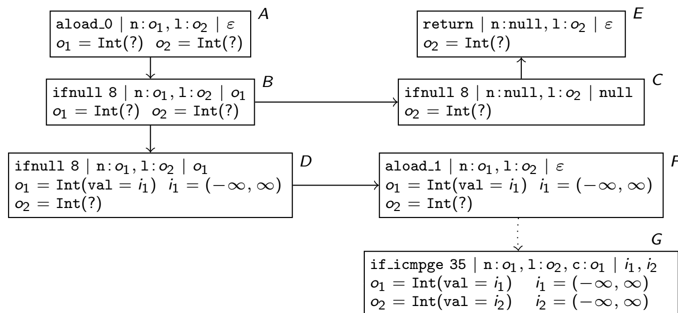



**States *E* and *F*:**

- evaluate "`ifnull 8`" in *C* and *D*
- *evaluation edges*

# From JBC to Termination Graphs

```
00: aload_0
01: ifnull 8
04: aload_1
     :
19: if_icmpge 35
     :
27: iconst_1
28: iadd
29: putfield val
32: goto 11
35: return
```

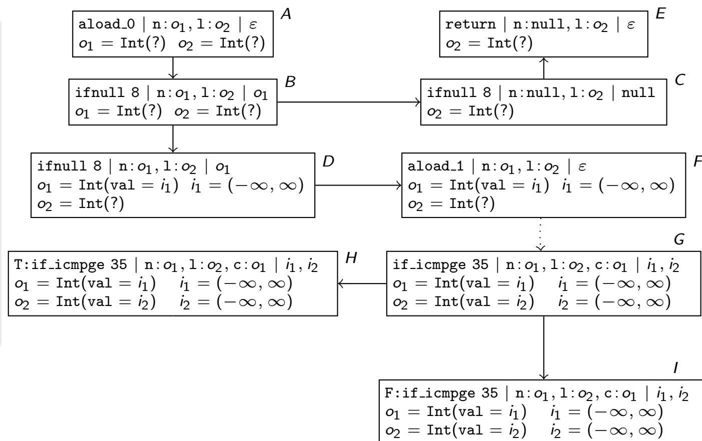



## State $G$:

- in state $F$, check if `limit` is `null` analogously
- aliasing in $G$: `num` and `copy` point to the same address $o_1$
- `val`-fields of `num` and `limit` pushed on operand stack

# From JBC to Termination Graphs

```
00: aload_0
01: ifnull 8
04: aload_1
      :
19: if_icmpge 35
      :
27: iconst_1
28: iadd
29: putfield val
32: goto 11
35: return
```

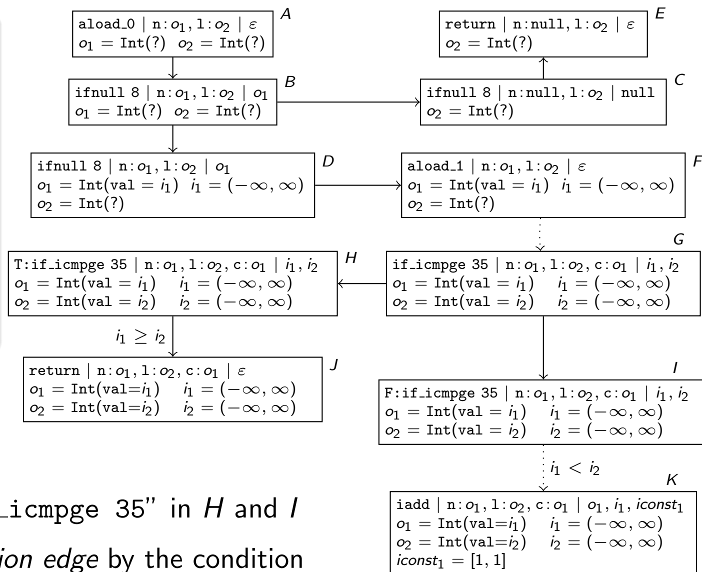



**States *H* and *I*:**

- “`if_icmpge 35`” needs to know whether $i_1 \geq i_2$
- *refine* information about heap (*refinement edges*)

# From JBC to Termination Graphs

```
00: aload_0
01: ifnull 8
04: aload_1
    :
19: if_icmpge 35
    :
27: iconst_1
28: iadd
29: putfield val
32: goto 11
35: return
```




**States $J$ and $K$:**

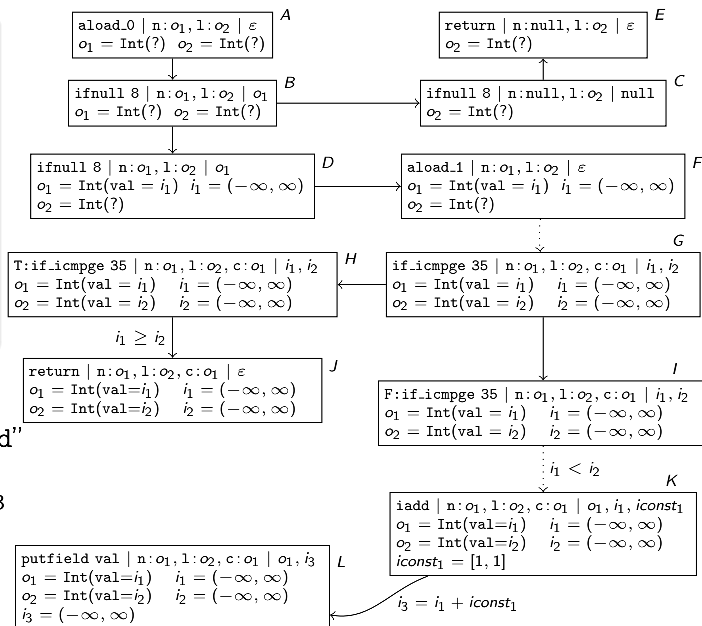
- evaluate "if_icmpge 35" in $H$ and $I$
- label *evaluation edge* by the condition
- val-field of copy and integer variable with value 1 on operand stack

# From JBC to Termination Graphs

```
00: aload_0
01: ifnull 8
04: aload_1
    ⋮
19: if_icmpge 35
    ⋮
27: iconst_1
28: iadd
29: putfield val
32: goto 11
35: return
```

**State $L$:**

- evaluate "iadd"
- new variable $i_3$
- label edge by connection

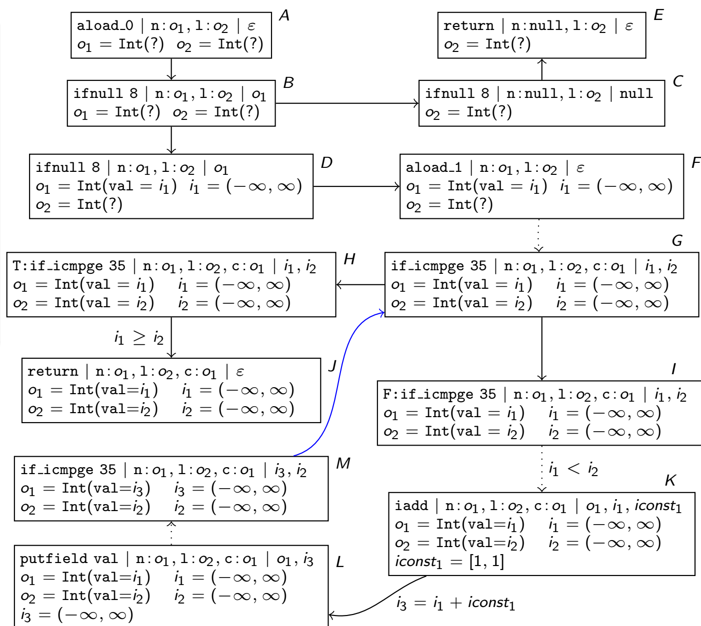


# From JBC to Termination Graphs

```
00: aload_0
01: ifnull 8
04: aload_1
      :
19: if_icmpge 35
      :
27: iconst_1
28: iadd
29: putfield val
32: goto 11
35: return
```

**State *M*:**

- again reaches "`if_icmpge`"
- *M* *instance* of *G*
- *instantiation edge*



# From JBC to Termination Graphs

## Termination Graphs

- expand nodes until all leaves correspond to program ends
- by appropriate generalization steps,
  one always reaches a *finite* termination graph
- state $s_1$ is *instance* of $s_2$ iff
  every concrete state described by $s_1$ is also described by $s_2$

## Using Termination Graphs for Termination Proofs

- every JBC-computation of concrete states
  corresponds to a *computation path* in the termination graph
- termination graph is called *terminating* iff
  it has no infinite computation path

# Example with User-Defined Data Type

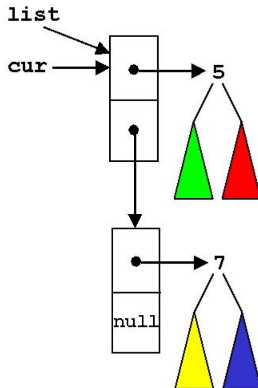
```
public class Flatten {
  public static IntList
         flatten(TreeList list) {
    TreeList cur = list;
    IntList result = null;

    while (cur != null) {
      Tree tree = cur.value;
      if (tree != null) {
         IntList oldIntList = result;
         result = new IntList();
         result.value = tree.value;
         result.next = oldIntList;
         TreeList oldCur = cur;
         cur = new TreeList();
         cur.value = tree.left;
         cur.next = oldCur;
         oldCur.value = tree.right;
      } else cur = cur.next;
    }
    return result;
  }
}
```

```
public class Tree {
  int value;
  Tree left;
  Tree right;
}

public class TreeList {
  Tree value;
  TreeList next;
}

public class IntList {
  int value;
  IntList next;
}
```

# Example with User-Defined Data Type

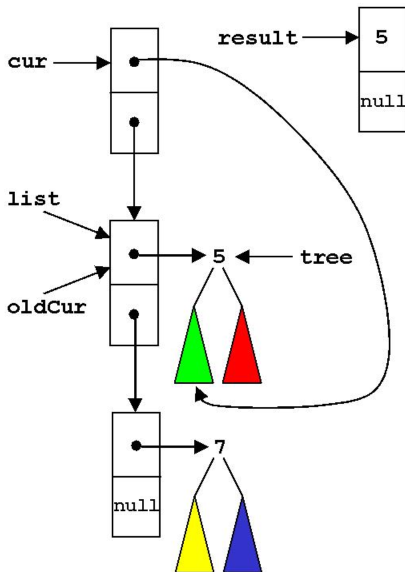
```
public class Flatten {
  public static IntList
         flatten(TreeList list) {
    TreeList cur = list;
    IntList result = null;

    while (cur != null) {
      Tree tree = cur.value;
      if (tree != null) {
         IntList oldIntList = result;
         result = new IntList();
         result.value = tree.value;
         result.next = oldIntList;
         TreeList oldCur = cur;
         cur = new TreeList();
         cur.value = tree.left;
         cur.next = oldCur;
         oldCur.value = tree.right;
      } else cur = cur.next;
    }
    return result;
  }
}
```

result: null



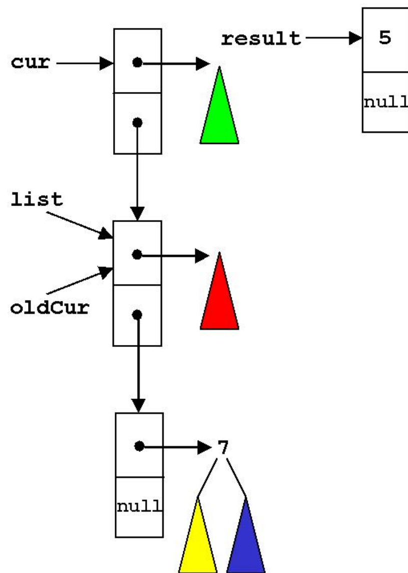
# Example with User-Defined Data Type

```
public class Flatten {
  public static IntList
         flatten(TreeList list) {
    TreeList cur = list;
    IntList result = null;

    while (cur != null) {
      Tree tree = cur.value;
      if (tree != null) {
         IntList oldIntList = result;
         result = new IntList();
         result.value = tree.value;
         result.next = oldIntList;
         TreeList oldCur = cur;
         cur = new TreeList();
         cur.value = tree.left;
         cur.next = oldCur;
         oldCur.value = tree.right;
      } else cur = cur.next;
    }
    return result;
  }
}
```



# Example with User-Defined Data Type

```
public class Flatten {
  public static IntList
         flatten(TreeList list) {
    TreeList cur = list;
    IntList result = null;

    while (cur != null) {
      Tree tree = cur.value;
      if (tree != null) {
         IntList oldIntList = result;
         result = new IntList();
         result.value = tree.value;
         result.next = oldIntList;
         TreeList oldCur = cur;
         cur = new TreeList();
         cur.value = tree.left;
         cur.next = oldCur;
         oldCur.value = tree.right;
      } else cur = cur.next;
    }
    return result;
  }
}
```




no termination by *path length*

# Example with User-Defined Data Type

```
public class Flatten {
  public static IntList
         flatten(TreeList list) {
    TreeList cur = list;
    IntList result = null;

    while (cur != null) {
      Tree tree = cur.value;
      if (tree != null) {
         IntList oldIntList = result;
         result = new IntList();
         result.value = tree.value;
         result.next = oldIntList;
         TreeList oldCur = cur;
         cur = new TreeList();
         cur.value = tree.left;
         cur.next = oldCur;
         oldCur.value = tree.right;
      } else cur = cur.next;
    }
    return result;
  }
}
```

**General state at beginning of loop body**

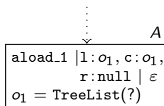
| $\texttt{aload_1} \mid \texttt{l}:o_1, \texttt{c}:o_2, \texttt{r}:o_3 \mid \varepsilon$ |
|---|
| $o_1 = \texttt{TreeList(?)}\ o_2 = \texttt{TreeList(?)}$ |
| $o_3 = \texttt{IntList(?)}$ |
| $o_1 =^? o_2 \qquad o_1 \searrow\!\swarrow o_2$ |

## Annotations

- $o_1 =^? o_2$: $o_1$ and $o_2$ *may be equal*
- $o_1 \searrow\!\swarrow o_2$: $o_1$ and $o_2$ *may join*
  - $o \to o'$ iff object at address $o$ has a field with value $o'$
  - $o_1 \searrow\!\swarrow o_2$: $o_1 \to^* \circ \leftarrow^+ o_2$ or $o_1 \to^+ \circ \leftarrow^* o_2$
- $o\,!$: $o$ *does not have to be a tree*

# Example with User-Defined Data Type

```
public class Flatten {
  public static IntList
         flatten(TreeList list) {
    TreeList cur = list;
    IntList result = null;

    while (cur != null) {
      Tree tree = cur.value;
      if (tree != null) {
         IntList oldIntList = result;
         result = new IntList();
         result.value = tree.value;
         result.next = oldIntList;
         TreeList oldCur = cur;
         cur = new TreeList();
         cur.value = tree.left;
         cur.next = oldCur;
         oldCur.value = tree.right;
      } else cur = cur.next;
    }
    return result;
  }
}
```

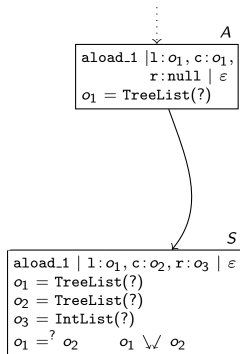



**State *A*:**

- reaches loop condition "`cur != null`" for the first time
- `list` and `cur` ($o_1$) are equal

# Example with User-Defined Data Type

```
public class Flatten {
  public static IntList
         flatten(TreeList list) {
    TreeList cur = list;
    IntList result = null;

    while (cur != null) {
      Tree tree = cur.value;
      if (tree != null) {
         IntList oldIntList = result;
         result = new IntList();
         result.value = tree.value;
         result.next = oldIntList;
         TreeList oldCur = cur;
         cur = new TreeList();
         cur.value = tree.left;
         cur.next = oldCur;
         oldCur.value = tree.right;
      } else cur = cur.next;
    }
    return result;
  }
}
```



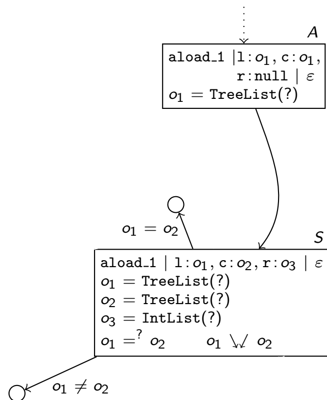

**State $S$:**

- *generalize* $A$ to obtain finite termination graph
- `list` ($o_1$) and `cur` ($o_2$) may be equal and may join

# Example with User-Defined Data Type

```
public class Flatten {
  public static IntList
        flatten(TreeList list) {
    TreeList cur = list;
    IntList result = null;

    while (cur != null) {
      Tree tree = cur.value;
      if (tree != null) {
         IntList oldIntList = result;
         result = new IntList();
         result.value = tree.value;
         result.next = oldIntList;
         TreeList oldCur = cur;
         cur = new TreeList();
         cur.value = tree.left;
         cur.next = oldCur;
         oldCur.value = tree.right;
      } else cur = cur.next;
    }
    return result;
  }
}
```

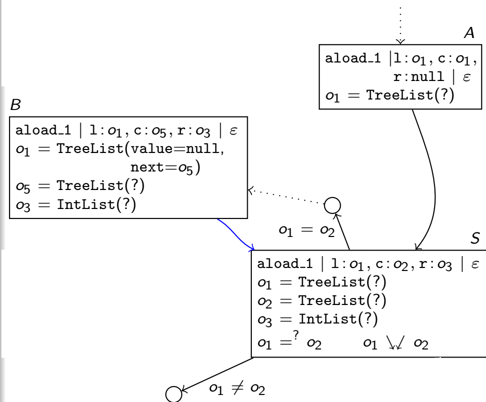
**State $S$:**




- *refinement* of annotation $o_1 =^? o_2$

# Example with User-Defined Data Type

```
public class Flatten {
  public static IntList
         flatten(TreeList list) {
    TreeList cur = list;
    IntList result = null;

    while (cur != null) {
      Tree tree = cur.value;
      if (tree != null) {
         IntList oldIntList = result;
         result = new IntList();
         result.value = tree.value;
         result.next = oldIntList;
         TreeList oldCur = cur;
         cur = new TreeList();
         cur.value = tree.left;
         cur.next = oldCur;
         oldCur.value = tree.right;
      } else cur = cur.next;
    }
    return result;
  }
}
```

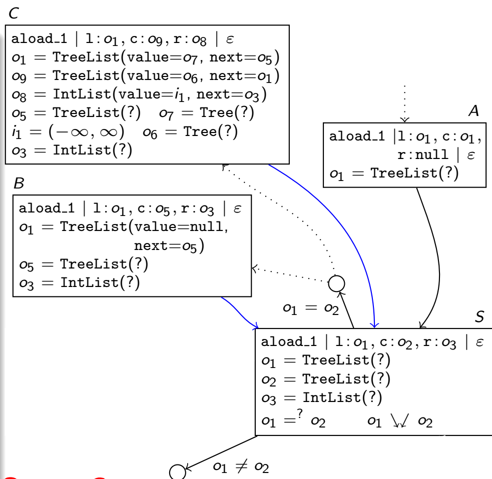



**State $B$:**

- reach loop condition if `tree == null`
- `list` $\to^{+} \circ \leftarrow^{*}$ `cur`
- $B$ is *instance* of $S$

# Example with User-Defined Data Type

```
public class Flatten {
  public static IntList
        flatten(TreeList list) {
    TreeList cur = list;
    IntList result = null;

    while (cur != null) {
      Tree tree = cur.value;
      if (tree != null) {
         IntList oldIntList = result;
         result = new IntList();
         result.value = tree.value;
         result.next = oldIntList;
         TreeList oldCur = cur;
         cur = new TreeList();
         cur.value = tree.left;
         cur.next = oldCur;
         oldCur.value = tree.right;
      } else cur = cur.next;
    }
    return result;
  }
}
```




**State $C$:**

- Tree(value=$i_1$, left=$o_6$, right=$o_7$)
- list $\to^* \circ \leftarrow^+$ cur
- $C$ is *instance* of $S$

# Example with User-Defined Data Type

```
public class Flatten {
  public static IntList
        flatten(TreeList list) {
    TreeList cur = list;
    IntList result = null;

    while (cur != null) {
      Tree tree = cur.value;
      if (tree != null) {
         IntList oldIntList = result;
         result = new IntList();
         result.value = tree.value;
         result.next = oldIntList;
         TreeList oldCur = cur;
         cur = new TreeList();
         cur.value = tree.left;
         cur.next = oldCur;
         oldCur.value = tree.right;
      } else cur = cur.next;
    }
    return result;
  }
}
```

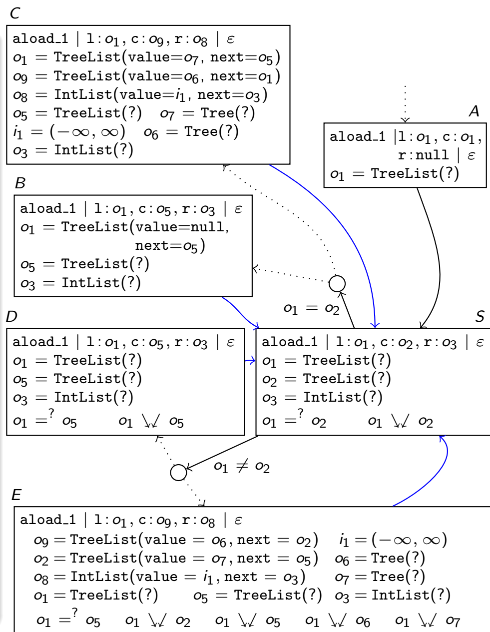


**State *D*:**

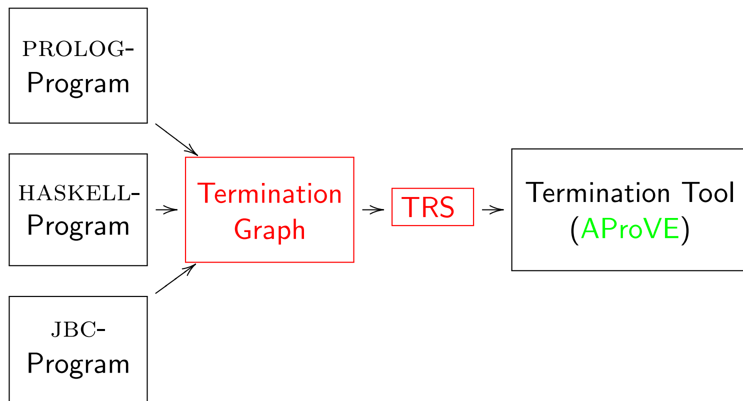
- $o_2$ = TreeList(value=$o_4$, next=$o_5$)
- tree ($o_4$) is null
- *D* is *instance* of *S*

# Example with User-Defined Data Type

```
public class Flatten {
  public static IntList
         flatten(TreeList list) {
    TreeList cur = list;
    IntList result = null;

    while (cur != null) {
      Tree tree = cur.value;
      if (tree != null) {
         IntList oldIntList = result;
         result = new IntList();
         result.value = tree.value;
         result.next = oldIntList;
         TreeList oldCur = cur;
         cur = new TreeList();
         cur.value = tree.left;
         cur.next = oldCur;
         oldCur.value = tree.right;
      } else cur = cur.next;
    }
    return result;
  }
}
```



# From Termination Graphs to TRSs



# Transforming Objects to Terms

$\texttt{aload_1} \mid \texttt{l}: o_1, \texttt{c}: o_9, \texttt{r}: o_8 \mid \varepsilon$

$o_9 = \texttt{TreeList}(\texttt{value} = o_6, \texttt{next} = o_2)$ $\quad i_1 = (-\infty, \infty)$

$o_2 = \texttt{TreeList}(\texttt{value} = o_7, \texttt{next} = o_5)$ $\quad o_6 = \texttt{Tree}(?)$

$o_8 = \texttt{IntList}(\texttt{value} = i_1, \texttt{next} = o_3)$ $\quad o_7 = \texttt{Tree}(?)$

$o_1 = \texttt{TreeList}(?)$ $\quad o_5 = \texttt{TreeList}(?)$ $\quad o_3 = \texttt{IntList}(?)$

$o_1 =^? o_5 \quad o_1 \searrow\!\swarrow o_2 \quad o_1 \searrow\!\swarrow o_5 \quad o_1 \searrow\!\swarrow o_6 \quad o_1 \searrow\!\swarrow o_7$

For every class C with $n$ fields,
introduce function symbol C with $n$ arguments

- term for $o_1$: $o_1$
- term for $o_2$: $\mathsf{TL}(o_7, o_5)$
- term for $o_9$: $\mathsf{TL}(o_6, \mathsf{TL}(o_7, o_5))$
- term for $o_8$: $\mathsf{IL}(i_1, o_3)$

# Transforming Objects to Terms

## Class Hierarchy

- for every class `C` with $n$ fields,
  introduce function symbol C with $n + 1$ arguments
- first argument: part of the object corresponding to subclasses of `C`

```
public class A {
  int a;
}
```

```
public class B extends A {
  int b;
}
```

```
A x = new A();
x.a = 1;
```

```
B y = new B();
y.a = 2;
y.b = 3;
```

- term for `x`: $\mathsf{jlO}(\mathsf{A}(\mathsf{eoc}, 1))$ (eoc for "end of class")
- term for `y`: $\mathsf{jlO}(\mathsf{A}(\mathsf{B}(\mathsf{eoc}, 3), 2))$ (jlO for "`java.lang.Object`")

# Transforming States to Tuples of Terms

**Transforming $D$**

$f_D(\,\text{jlO}(\text{Int}(\text{eoc}, i_1)),$
$\quad o_2,$
$\quad \text{jlO}(\text{Int}(\text{eoc}, i_1))\,)$

**Transforming $F$**

$f_F(\,\text{jlO}(\text{Int}(\text{eoc}, i_1)),$
$\quad o_2\,)$

A: aload_0 | n: $o_1$, l: $o_2$ | $\varepsilon$
$o_1$ = Int(?) $o_2$ = Int(?)

B: ifnull 8 | n: $o_1$, l: $o_2$ | $o_1$
$o_1$ = Int(?) $o_2$ = Int(?)

C: ifnull 8 | n: null, l: $o_2$ | null
$o_2$ = Int(?)

E: return | n: null, l: $o_2$ | $\varepsilon$
$o_2$ = Int(?)

D: ifnull 8 | n: $o_1$, l: $o_2$ | $o_1$
$o_1$ = Int(val = $i_1$) $i_1 = (-\infty, \infty)$
$o_2$ = Int(?)

F: aload_1 | n: $o_1$, l: $o_2$ | $\varepsilon$
$o_1$ = Int(val = $i_1$) $i_1 = (-\infty, \infty)$
$o_2$ = Int(?)

G: if_icmpge 35 | n: $o_1$, l: $o_2$, c: $o_1$ | $i_1$, $i_2$
$o_1$ = Int(val = $i_1$) $i_1 = (-\infty, \infty)$
$o_2$ = Int(val = $i_2$) $i_2 = (-\infty, \infty)$

H: T:if_icmpge 35 | n: $o_1$, l: $o_2$, c: $o_1$ | $i_1$, $i_2$
$o_1$ = Int(val = $i_1$) $i_1 = (-\infty, \infty)$
$o_2$ = Int(val = $i_2$) $i_2 = (-\infty, \infty)$

$i_1 \geq i_2$

J: return | n: $o_1$, l: $o_2$, c: $o_1$ | $\varepsilon$
$o_1$ = Int(val=$i_1$) $i_1 = (-\infty, \infty)$
$o_2$ = Int(val=$i_2$) $i_2 = (-\infty, \infty)$

I: F:if_icmpge 35 | n: $o_1$, l: $o_2$, c: $o_1$ | $i_1$, $i_2$
$o_1$ = Int(val = $i_1$) $i_1 = (-\infty, \infty)$
$o_2$ = Int(val = $i_2$) $i_2 = (-\infty, \infty)$

$i_1 < i_2$

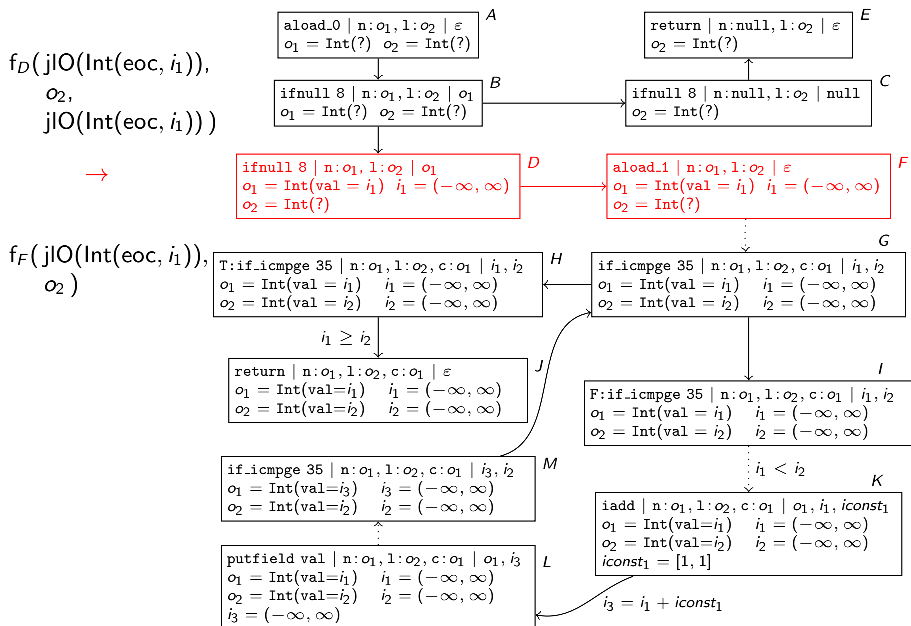
K: iadd | n: $o_1$, l: $o_2$, c: $o_1$ | $o_1$, $i_1$, $iconst_1$
$o_1$ = Int(val=$i_1$) $i_1 = (-\infty, \infty)$
$o_2$ = Int(val=$i_2$) $i_2 = (-\infty, \infty)$
$iconst_1 = [1, 1]$

$i_3 = i_1 + iconst_1$

L: putfield val | n: $o_1$, l: $o_2$, c: $o_1$ | $o_1$, $i_3$
$o_1$ = Int(val=$i_1$) $i_1 = (-\infty, \infty)$
$o_2$ = Int(val=$i_2$) $i_2 = (-\infty, \infty)$
$i_3 = (-\infty, \infty)$

M: if_icmpge 35 | n: $o_1$, l: $o_2$, c: $o_1$ | $i_3$, $i_2$
$o_1$ = Int(val=$i_3$) $i_3 = (-\infty, \infty)$
$o_2$ = Int(val=$i_2$) $i_2 = (-\infty, \infty)$

# Transforming Edges to Rewrite Rules



# Transforming Edges to Rewrite Rules

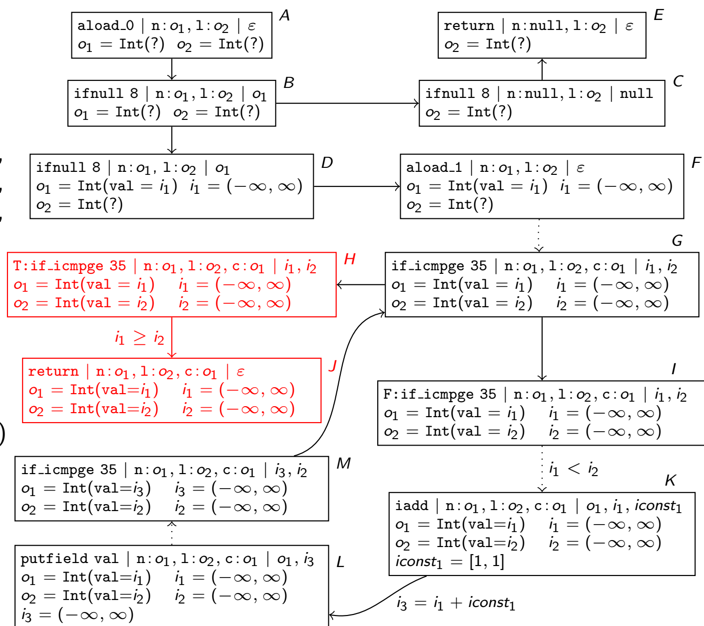
**Transforming Evaluation Edges with Conditions**

$$f_H(\,\mathrm{jlO}(\mathrm{Int}(\mathrm{eoc}, i_1)),\ \mathrm{jlO}(\mathrm{Int}(\mathrm{eoc}, i_2)),\ \mathrm{jlO}(\mathrm{Int}(\mathrm{eoc}, i_1)),\ i_1,\ i_2\,)$$

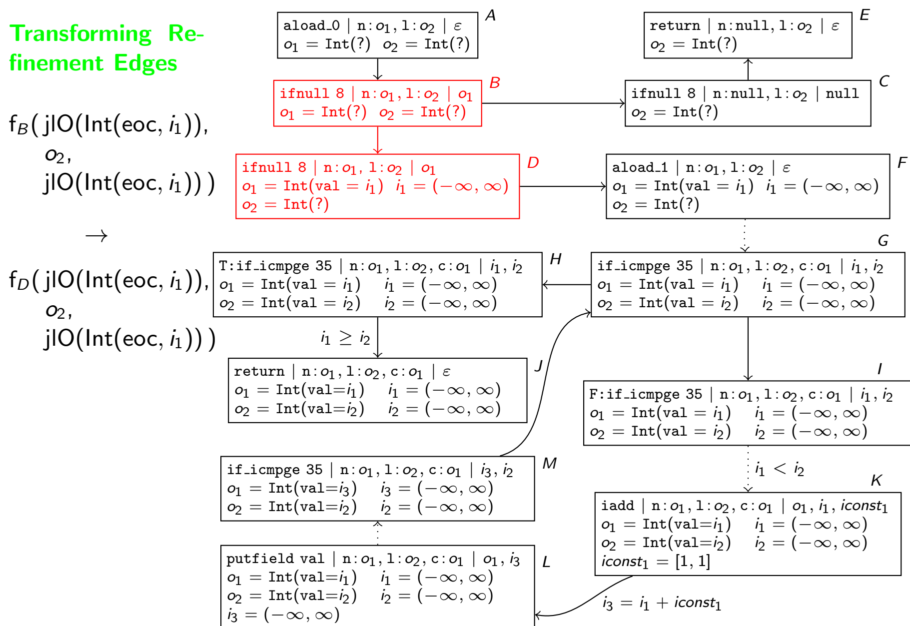
$$\rightarrow$$

$$f_J(\,\mathrm{jlO}(\mathrm{Int}(\mathrm{eoc}, i_1)),\ \mathrm{jlO}(\mathrm{Int}(\mathrm{eoc}, i_2)),\ \mathrm{jlO}(\mathrm{Int}(\mathrm{eoc}, i_1))\,)$$

$$\mid\ i_1 \geq i_2$$



# Transforming Edges to Rewrite Rules



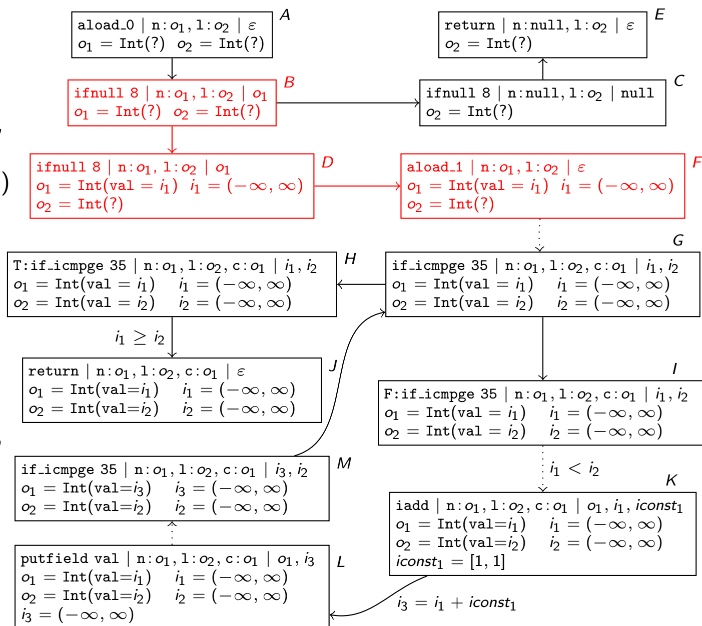
# Transforming Edges to Rewrite Rules

**Merging Rewrite Rules**

$$f_B(\,\text{jlO}(\text{Int}(\text{eoc}, i_1)),\; o_2,\; \text{jlO}(\text{Int}(\text{eoc}, i_1))\,)$$

$$\to$$

$$f_F(\,\text{jlO}(\text{Int}(\text{eoc}, i_1)),\; o_2\,)$$



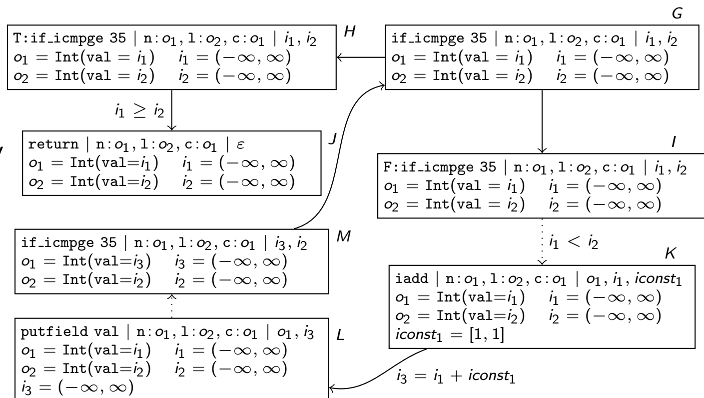
# Transforming Edges to Rewrite Rules

## TRS for count

$$\begin{array}{llllll} f_G(\,\mathrm{jlO}(\mathrm{Int}(\mathrm{eoc}, i_1)), & \mathrm{jlO}(\mathrm{Int}(\mathrm{eoc}, i_2)), & \mathrm{jlO}(\mathrm{Int}(\mathrm{eoc}, i_1)), & i_1, & i_2) & \to \\ f_G(\,\mathrm{jlO}(\mathrm{Int}(\mathrm{eoc}, i_1+1)), & \mathrm{jlO}(\mathrm{Int}(\mathrm{eoc}, i_2)), & \mathrm{jlO}(\mathrm{Int}(\mathrm{eoc}, i_1+1)), & i_1+1, & i_2) & \mid \quad i_1 < i_2 \end{array}$$

TRS is "*natural*"

termination easy to prove automatically



# From Termination Graphs to TRSs

## TRS for count

$$\begin{array}{llllll} f_G(\,\mathrm{jlO}(\mathrm{Int}(\mathrm{eoc}, i_1)), & \mathrm{jlO}(\mathrm{Int}(\mathrm{eoc}, i_2)), & \mathrm{jlO}(\mathrm{Int}(\mathrm{eoc}, i_1)), & i_1, & i_2) & \to \\ f_G(\,\mathrm{jlO}(\mathrm{Int}(\mathrm{eoc}, i_1 + 1)), & \mathrm{jlO}(\mathrm{Int}(\mathrm{eoc}, i_2)), & \mathrm{jlO}(\mathrm{Int}(\mathrm{eoc}, i_1 + 1)), & i_1 + 1, & i_2) & \mid \quad i_1 < i_2 \end{array}$$

- every JBC-computation of concrete states corresponds to a *computation path* in the termination graph
- termination graph is called *terminating* iff it has no infinite computation path
- every computation path corresponds to rewrite sequence in TRS

## Theorem

TRS corresponding to termination graph is terminating $\Rightarrow$

termination graph is terminating $\Rightarrow$

JBC-program terminating for all states represented in termination graph

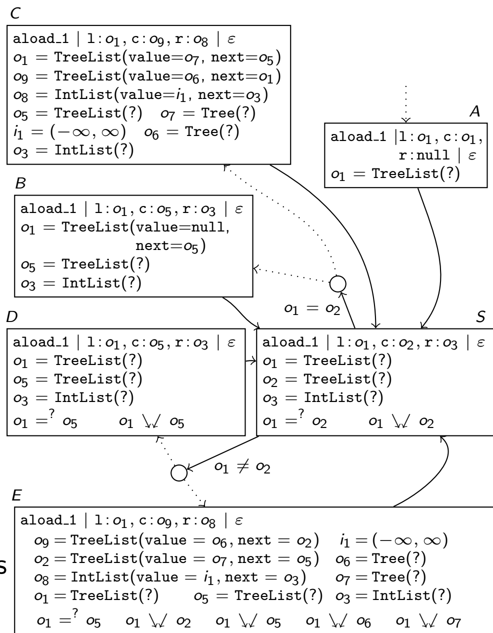
# From Termination Graphs to TRSs

$$f_S(\mathsf{TL}(\mathsf{null}, o_5), \mathsf{TL}(\mathsf{null}, o_5), o_3) \to$$
$$f_S(\mathsf{TL}(\mathsf{null}, o_5), \quad o_5 \;, o_3)$$

$$f_S(\ldots, \mathsf{TL}(\mathsf{T}(i_1, o_6, o_7), o_5), \quad o_3 \;) \to$$
$$f_S(\ldots, \mathsf{TL}(\; o_6, \mathsf{TL}(o_7, o_5)), \mathsf{IL}(i_1, o_3))$$

$$f_S(o_1, \mathsf{TL}(\mathsf{null}, o_5), o_3) \to$$
$$f_S(o_1, \quad o_5 \;, o_3)$$

$$f_S(o_1, \mathsf{TL}(\mathsf{T}(i_1, o_6, o_7), o_5), \quad o_3 \;) \to$$
$$f_S(o_1', \mathsf{TL}(\; o_6, \mathsf{TL}(o_7, o_5)), \mathsf{IL}(i_1, o_3))$$




## Rewrite Rules & Annotations

- when writing to a field of $o_2$ with $o_1 \searrow\!\!\swarrow o_2$: $o_1$ on lhs, fresh variable $o_1'$ on rhs
- cyclic objects: fresh variable on rhs

# From Termination Graphs to TRSs

$$f_S(\mathsf{TL}(\mathsf{null}, o_5), \mathsf{TL}(\mathsf{null}, o_5), o_3) \to$$
$$f_S(\mathsf{TL}(\mathsf{null}, o_5), \quad o_5 \ , o_3)$$

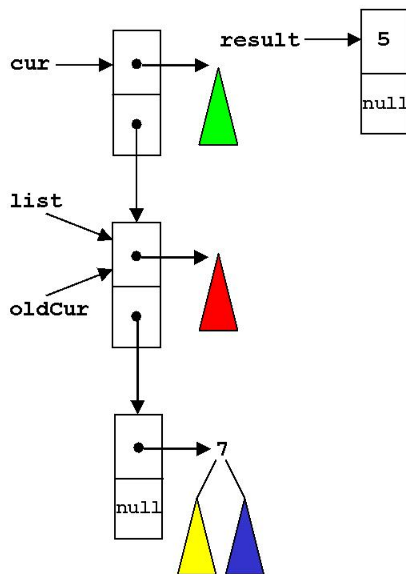
$$f_S(\dots, \mathsf{TL}(\mathsf{T}(i_1, o_6, o_7), o_5), \quad o_3 \ ) \to$$
$$f_S(\dots, \mathsf{TL}( \ o_6, \mathsf{TL}(o_7, o_5)), \mathsf{IL}(i_1, o_3))$$

$$f_S(o_1, \mathsf{TL}(\mathsf{null}, o_5), o_3) \to$$
$$f_S(o_1, \quad o_5 \ , o_3)$$

$$f_S(o_1, \mathsf{TL}(\mathsf{T}(5, o_6, o_7), o_5), \quad \mathsf{null} \ ) \to$$
$$f_S(o_1', \mathsf{TL}( \ o_6, \mathsf{TL}(o_7, o_5)), \mathsf{IL}(5, \mathsf{null}))$$

TRS is "*natural*"

termination easy
to prove automatically



# Automated Termination Analysis of JAVA BYTECODE by Term Rewriting

- implemented in AProVE and evaluated on collection of 387 JAVA-programs (including `java.util.LinkedList` and `HashMap`)
- extended for *recursion* and *cyclic data*
- adapted to detect *non-termination* and `NullPointerExceptions`

| | Yes | No | Failure | Timeout | Runtime |
|---|---|---|---|---|---|
| AProVE | 267 | 81 | 11 | 28 | 9.5 |
| Julia | 191 | 22 | 174 | 0 | 4.7 |
| COSTA | 160 | 0 | 181 | 46 | 11.0 |

- AProVE winner of the *International Termination Competition* for JAVA, HASKELL, PROLOG, term rewriting
- termination of "real" languages can be analyzed automatically, term rewriting is a suitable approach