Confluence Framework

International School on Rewriting 2022

Raúl Gutiérrez

TBILISI, SEPTEMBER 24TH, 2022

Universidad Politécnica de Madrid
Spain

Motivation

- Solving undecidable problems: a lot of different partial techniques.
- What those techniques share in common.
- Can we encapsulate those techiques in order to combine them in a flexible way?

The Dependency Pair Framework

- There is a precedence in termination of what we want.
- Based on two notions: **DP problems** and **DP processors**.
- DP problems allow us to encapsulate systems.
- DP processors (techniques) are combined generating a proof strategy.

DP problems

A DP problem (P, Q, R, f) consists of three TRSs P, Q, R and a flag $f \in \{m, a\}$. A DP problem (P, Q, R, m) is **finite** iff there is no infinite minimal (P, Q, R)-chain and (P, Q, R, a) is **finite** iff there is no infinite (P, Q, R)-chain. A DP problem (P, Q, R, f) is **infinite** iff it is not finite or if R is not Q-terminating

The Dependency Pair Framework

Example [GTS04]

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

Example [GTS04]

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

DP processors

A DP processor is a function Proc which takes a DP problem as input and returns either a set of DP problems or the result "no". A DP processor Proc is **sound** if for all DP problems d, d is finite whenever $\mathit{Proc}(d)$ is not "no" and all DP problems in $\mathit{Proc}(d)$ are finite. A DP processor Proc is **complete** if for all DP problems d, d is infinite whenever $\mathit{Proc}(d)$ is "no" or when $\mathit{Proc}(d)$ contains an infinite DP problem.

How DP Problems and Processors are Combined?

Dependency pair framework

Let R and Q be TRSs. We construct a tree whose nodes are labelled with DP problems or "yes" or "no" and whose root is labelled with $(DP(R), Q, R, m)$. For every inner node labelled with d, there is a sound DP processor $Proc$ satisfying one of the following conditions:

- $Proc(d) =$ no and the node has just one child, labelled with "no".
- $Proc(d) = \varnothing$ and the node has just one child, labelled with "yes".
- $Proc(d) \neq$ no, $Proc(d) \neq \varnothing$, and the children of the node are labelled with the DP problems in $Proc(d)$.

If all leaves of the tree are labelled with "yes", then R is Q-terminating. Otherwise, if there is a leaf labelled with "no" and if all processors used on the path from the root to this leaf are complete, then R is not Q-terminating.

The Dependency Pair Framework

- The success of the framework spreads to other variants of TRSs.
- Understand and formalize what is specific in the variant, parameterize the framework with the new notions.

Example 1: The Context-Sensitive Dependency Pair Framework

DP problems

A CS problem τ is a tuple $\tau = (P, R, S, \mu)$, where R, P and S are TRSs, and $\mu \in M_{R \cup P \cup S}$. The CS problem (P, R, S, μ) is **finite** if there is no infinite (P, R, S, μ)-chain. The CS problem (P, R, S, μ) is **infinite** if R is non-μ-terminating or there is an infinite minimal (P, R, S, μ)-chain.

Example 1: The Context-Sensitive Dependency Pair Framework

Example [AEF+08]

$$\begin{array}{rcl}
gt(0, y) & \rightarrow & false \\
gt(s(x), 0) & \rightarrow & true \\
gt(s(x), s(y)) & \rightarrow & gt(x, y) \\
if(true, x, y) & \rightarrow & x \\
if(false, x, y) & \rightarrow & y \\
p(0) & \rightarrow & 0 \\
p(s(x)) & \rightarrow & x \\
minus(x, y) & \rightarrow & if(gt(y, 0), minus(p(x), p(y)), x) \\
div(0, s(y)) & \rightarrow & 0 \\
div(s(x), s(y)) & \rightarrow & s(div(minus(x, y), s(y)))
\end{array}$$

where $\mu(if) = \{1\}$

The Dependency Pair Framework

Example [AEF+08]

P:

$$
\begin{array}{rcl}
GT(s(x), s(y)) & \to & GT(x, y) \\
IF(true, x, y) & \to & x \\
IF(false, x, y) & \to & y \\
MINUS(x, y) & \to & IF(gt(y, 0), minus(p(x), p(y)), x) \\
MINUS(x, y) & \to & GT(y, 0) \\
DIV(s(x), s(y)) & \to & DIV(minus(x, y), s(y)) \\
DIV(s(x), s(y)) & \to & MINUS(x, y)
\end{array}
$$

S:

$$
\begin{array}{rcl}
minus(p(x), p(y)) & \to & MINUS(p(x), p(y)) \\
p(x) & \to & P(x) \\
minus(x, y) & \to & x \\
minus(x, y) & \to & y
\end{array}
$$

Example 1: The Context-Sensitive Dependency Pair Framework

DP processors

A CS processor Proc is a mapping from CS problems into sets of CS problems. A CS-processor Proc is **sound** if for all CS problems τ, τ is finite whenever $\forall \tau \in \mathit{Proc}(\tau)$, τ is finite.

A CS processor Proc is **complete** if for all CS problems τ, τ is infinite whenever $\mathit{Proc}(\tau)$ is “no” or $\exists \tau' \in \mathit{Proc}(\tau)$ such that τ' is infinite.

Example 1: The Context-Sensitive Dependency Pair Framework

Context-Senstivite Dependency pair framework

Let R be a TRS and $\mu \in M_R$. We construct a tree whose nodes are labeled with CS problems or “yes” or “no”, and whose root is labeled with $(DP(R,\mu), R, unh(R,\mu), \mu)$. For every inner node labeled with τ, there is a sound processor $Proc$ satisfying one of the following conditions:

- $Proc(\tau) = no$ and the node has just one child, labeled with “no”.
- $Proc(\tau) = \varnothing$ and the node has just one child, labeled with “yes”.
- $Proc(\tau) \neq no$, $Proc(\tau) \neq \varnothing$, and the children of the node are labeled with the CS problems in $Proc(\tau)$.

If all leaves of the tree are labeled with “yes”, then R is μ-terminating. Otherwise, if there is a leaf labeled with “no” and if all processors used on the path from the root to this leaf are complete, then R is non-μ-terminating.

Example 2: The 2D DP Framework

2D DP problems

A CTRS problem is a tuple $\tau = (P, Q, R, f)$, where P, Q, and R are CTRSs, and $f \in F$.

A CTRS problem $\tau = (P, Q, R, f)$ with $f \in F$ is **finite** iff there is no infinite τ-chain; τ is **infinite** iff there is an infinite τ-chain (iff τ is not finite).

Example 2: The 2D DP Framework

Example [Ohl02]

$$
\begin{array}{rcl}
less(x, 0) & \to & false \\
less(0, s(y)) & \to & true \\
less(s(x), s(y)) & \to & less(x, y) \\
monus(0, s(y)) & \to & 0 \\
monus(s(x), s(y)) & \to & monus(x, y) \\
quotrem(0, s(y)) & \to & pair(0, 0) \\
quotrem(s(x), s(y)) & \to & pair(0, s(x)) \Leftarrow less(x, y) \to true \\
quotrem(s(x), s(y)) & \to & pair(s(q), r) \Leftarrow less(x, y) \to false, \\
 & & \quad quotrem(monus(x, y), s(y)) \to pair(q, r)
\end{array}
$$

Example 2: The 2D DP Framework

Example [Ohl02]

DP_H:

$$\begin{array}{rcl} LESS(s(x), s(y)) & \to & LESS(x, y) \\ MONUS(s(x), s(y)) & \to & MONUS(x, y) \end{array}$$

DP_V:

$$\begin{array}{rcl} QUOTREM(s(x), s(y)) & \to & LESS(x, y) \\ QUOTREM(s(x), s(y)) & \to & QUOTREM(monus(x, y), s(y)) \Rightarrow less(x, y) \to false \\ QUOTREM(s(x), s(y)) & \to & MONUS(x, y) \Rightarrow less(x, y) \to false \end{array}$$

Example 2: The 2D DP Framework

2D DP problems

A CTRS problem is a tuple $\tau = (P, Q, R, f)$, where P, Q, and R are CTRSs, and $f \in F$.

A CTRS problem $\tau = (P, Q, R, f)$ with $f \in F$ is **finite** iff there is no infinite τ-chain; τ is **infinite** iff there is an infinite τ-chain (iff τ is not finite).

Example 2: The 2D DP Framework

2D DP processors

A CTRS processor P is a partial function from CTRS problems into sets of CTRS problems. Alternatively, it can return “no”. The domain of P, denoted $Dom(P)$, is the set of CTRS problems τ for which P is defined.

Let P be a processor and $\tau \in Dom(P)$. We say that P is:

- τ-**sound** iff τ is finite whenever $P(\tau) = no$ and for all $\tau' \in P(\tau)$, τ' is finite.
- τ-**complete** iff τ is infinite whenever $P(\tau) = no$ or there is $\tau' \in P(\tau)$ such that τ' is infinite.

Example 2: The 2D DP Framework

2D DP framework

Let τ_I be a CTRS problem. A CTRS Proof tree T (CTRSP-tree for short) for τ_I is a tree whose nodes are labeled with CTRS problems; the leaves may also be labeled with either "yes" or "no". The root of T is labeled with τ_I. For every inner node n with label τ, there is a processor P such that $\tau \in Dom(P)$ and:

- If $P(\tau) = no$, then n has just one child n' with label "no".
- If $P(\tau) = \varnothing$, then n has just one child n' with label "yes".
- If $P(\tau) = \{\tau_1, \ldots, \tau_k\}$ with $k > 0$, then n has exactly k children $n_1, \ldots, n_k$ with labels $\tau_1, \ldots, \tau_k$, respectively.

Let τ be a CTRS problem and T be a CTRSP-tree for τ. Then,

1. If all leaves in T are labeled with "yes" and all involved processors are sound for the CTRS problems they are applied to, then τ is finite.
2. If T has a leaf with label "no" and all processors from τ to the leaf are complete for the CTRS problems they are applied to, then τ is infinite.

Example 2: The 2D DP Framework

Multiple uses?

- If $(DPH(R), \varnothing, R, a)$ is finite and R preserves terminating substitutions, then R is terminating.
- If $(DPH(R), \varnothing, R, f)$ is infinite for some $f \in F$, then R is nonterminating.
- If $(DPV(R), DPVH(R), R, a)$ is finite and R is a deterministic 3-CTRS, then R is V-terminating.
- If $(DPV(R), DPVH(R), R, f)$ is infinite for some $f \in F$, then R is non-V-terminating.
- If both $(DPH(R), \varnothing, R, f)$ and $(DPV(R), DPVH(R), R, f')$ are finite for some $f, f' \in F$ and R is a deterministic 3-CTRS, then R is operationally terminating.
- If $(DPH(R), \varnothing, R, f)$ or $(DPV(R), DPVH(R), R, f')$ are infinite for some $f, f' \in F$, then R is operationally nonterminating.

Example 3: DP Framework for Innermost Complexity

DT Tuple

A dependency tuple is a rule of the form $s^\sharp \to COM_n(t_1^\sharp, \ldots, t_n^\sharp)$ for $s^\sharp, t_1^\sharp, \ldots, t_n^\sharp \in T^\sharp$. Let $\ell \to r$ be a rule with $Pos_d(r) = \{\pi_1, \ldots, \pi_n\}$. Then $DT(\ell \to r)$ is defined to be $\ell^\sharp \to COM_n(r|_{\pi_1}^\sharp, \ldots, r|_{\pi_n}^\sharp)$. For a TRS R, let $DT(R) = \{DT(\ell \to r) | \ell \to r \in R\}$.

DT Problem

Let R be a TRS, D a set of DTs, $S \subseteq D$. Then $\langle D, S, R\rangle$ is a DT problem and R's canonical DT problem is $\langle DT(R), DT(R), R\rangle$.

Example 3: DP Framework for Innermost Complexity

Example [NEG11]

$$
\begin{array}{rcl}
gt(0, y) & \to & \mathit{false} \\
gt(s(x), 0) & \to & \mathit{true} \\
gt(s(x), s(y)) & \to & gt(x, y) \\
p(0) & \to & 0 \\
p(s(x)) & \to & x \\
\mathit{minus}(x, y) & \to & \mathit{if}(gt(x, y), x, y) \\
\mathit{if}(\mathit{true}, x, y) & \to & s(\mathit{minus}(p(x), y)) \\
\mathit{if}(\mathit{false}, x, y) & \to & 0
\end{array}
$$

Example 3: DP Framework for Innermost Complexity

Example [NEG11]

$$
\begin{aligned}
GT(0, y) &\rightarrow COM_0 \\
GT(s(x), 0) &\rightarrow COM_0 \\
GT(s(x), s(y)) &\rightarrow COM_1(GT(x, y)) \\
P(0) &\rightarrow COM_0 \\
P(s(x)) &\rightarrow COM_0 \\
MINUS(x, y) &\rightarrow COM_2(IF(gt(x, y), x, y), GT(x, y)) \\
IF(true, x, y) &\rightarrow COM_2(minus(p(x), y), P(x)) \\
IF(false, x, y) &\rightarrow COM_0
\end{aligned}
$$

DT Processor, $\oplus$

A DT processor $Proc$ is a function $Proc(P) = (c, P')$ mapping any DT problem P to a complexity $c \in \mathcal{C}$ and a DT problem P'. A processor is **sound** if $\iota_P \sqsubseteq c \oplus \iota_{P'}$. Here, $\oplus$ is the “maximum” function on $\mathcal{C}$, i.e., for any $c, d \in \mathcal{C}$, we define $c \oplus d = d$ if $c \sqsubseteq d$ and $c \oplus d = c$ otherwise.

Divide and Conquer Frameworks

- Deal with undecidable problems.
- Many different techniques along the history.
- Is it possible to combine those techniques?

Confluence Framework

Confluence Problem

- Let $\mathcal{R}$ be a CTRS (TRSs are included). A *confluence problem*, denoted $CR(\mathcal{R})$, is *positive* if $\mathcal{R}$ is confluent; otherwise, it is *negative*.
- Let $\mathcal{R}$ be a TRS and μ be a replacement map. A μ-confluence problem, denoted $CR(\mathcal{R}, \mu)$, is *positive* if $\mathcal{R}$ is μ-confluent; otherwise, it is *negative*.

Joinability Problem

- Let $\mathcal{R}$ be a CTRS and π be a *conditional pair* where the conditional part has no occurrence of $\hookrightarrow$. A *joinability problem*, denoted $JO(\mathcal{R}, \pi)$, is *positive* if π is joinable in $\mathcal{R}$; otherwise, it is *negative*.
- Let $\mathcal{R}$ be a TRS, μ be a replacement map, and π be a *conditional pair* where the conditional part contains at most an occurrence of $\hookrightarrow$ [1]. A μ*-joinability problem*, denoted $JO(\mathcal{R}, \mu, \pi)$, is *positive* if π is μ-joinable in $\mathcal{R}$; otherwise, it is *negative*.

[1] This is necessary to deal with the extended μ-critical pairs

Processor

A *processor* P is a partial function from problems into sets of problems; alternatively it can return "no". The domain of P (i.e., the set of problems on which P is defined) is denoted $\mathcal{D}om(\mathsf{P})$. We say that P is

- *sound* if for all $\tau \in \mathcal{D}om(\mathsf{P})$, τ is positive whenever $\mathsf{P}(\tau) \neq$ "no" and all $\tau' \in \mathsf{P}(\tau)$ are positive.
- *complete* if for all $\tau \in \mathcal{D}om(\mathsf{P})$, τ is negative whenever $\mathsf{P}(\tau) =$ "no" or τ' is negative for some $\tau' \in \mathsf{P}(\tau)$.

List of processors

Simplification processor

$$\mathsf{P}_{\mathsf{Simp}}(CR(\mathcal{R})) = \{CR(\mathcal{R}')\}$$

Simplifications

1. *Removing or transforming rules (CS-TRSs and CTRSs).* All rules $t \to t$ or $t \to t \mid c$ for some term t are removed.
2. *Removing infeasible rules (CTRSs only).* Rules $\ell \to r \mid c$ with an infeasible condition c (i.e., such that no substitution σ makes $\sigma(c)$ to hold in $\mathcal{R}$) are removed. We use infChecker for that purpose.
3. *Inlining conditional rules (CTRSs only).* We can often reduce the number of conditions of a rule, by using *inlining*.

Modular decomposition processor

$$\mathsf{P}_{\mathsf{MD}}(CR(\mathcal{R})) = \{CR(\mathcal{R}_1), CR(\mathcal{R}_2)\}$$

Modularity conditions

1. Disjoint TRSs.
2. Constructor-sharing and left-linear TRSs.
3. Constructor-sharing and layer-preserving TRSs.

Huet&Levi processor

$$\mathsf{P}_{\mathsf{HL}}(CR(\mathcal{R})) = \varnothing$$

Orthogonality conditions

1. Weakly orthogonal TRSs are confluent.
2. Left-linear and has no extended μ-critical pairs.
3. Orthogonal, properly oriented and right-stable.

Extended Huet processor

$$\mathsf{P}_{\mathsf{Huet}}(CR(\mathcal{R})) = \{JO(\mathcal{R}, \pi_1), \ldots, JO(\mathcal{R}, \pi_n)\}$$

$$\mathsf{P}_{\mathsf{Huet}}(CR(\mathcal{R}, \mu)) = \{JO(\mathcal{R}, \mu, \pi_1), \ldots, JO(\mathcal{R}, \mu, \pi_n)\}$$

Conditions

1. $\pi_1, \ldots, \pi_n$ are (extended μ-)critical pairs.

Extended Huet&Newman processor

$$\mathsf{P}_{\mathsf{HN}}(CR(\mathcal{R})) = \{JO(\mathcal{R}, \pi_1), \ldots, JO(\mathcal{R}, \pi_n)\}$$

$$\mathsf{P}_{\mathsf{HN}}(CR(\mathcal{R}, \mu)) = \{JO(\mathcal{R}, \mu, \pi_1), \ldots, JO(\mathcal{R}, \mu, \pi_n)\}$$

Conditions

1. $\mathcal{R}$ is (μ-)terminating.
2. $\pi_1, \ldots, \pi_n$ are (extended μ-)critical pairs.
3. For normal CTRSs, if $\mathcal{R}$ is terminating and left-linear.

Canonical joinability processor

$$\mathsf{P}_{\mathsf{CanJ}}(CR(\mathcal{R})) = \{JO(\mathcal{R},\mu,\pi_1),\ldots,JO(\mathcal{R},\mu,\pi_n)\}$$

Conditions

1. $\mathcal{R}$ is left-linear and level decreasing.
2. μ is the canonical replacement map.
3. $\mathcal{R}$ is μ-terminating.
4. $\pi_1,\ldots,\pi_n$ are μ-critical pairs.

Canonical μ-confluence processor

$$\mathsf{P}_{\mathsf{CanCR}}(CR(\mathcal{R})) = \{CR(\mathcal{R}, \mu)\}$$

Conditions

1. $\mathcal{R}$ is left-linear and normalizing.
2. μ is the canonical replacement map.

Unravelings processor

$$\mathsf{P}_{\mathsf{U}}(CR(\mathcal{R})) = \{CR(U(\mathcal{R}))\}$$

Conditions

1. $\mathcal{R}$ is terminating.
2. $\mathcal{R}$ is a deterministic 3-CTRS.

Joinability processor

$$\mathsf{P}_{\mathsf{JO}}(JO(\mathcal{R},\pi)) = \begin{cases} \varnothing & \text{if } \pi \text{ is joinable w.r.t. } \mathcal{R} \\ \text{no} & \text{if } \pi \text{ is not joinable w.r.t. } \mathcal{R} \end{cases}$$

$$\mathsf{P}_{\mathsf{JO}}(JO(\mathcal{R},\mu,\pi)) = \begin{cases} \varnothing & \text{if } \pi \text{ is } \mu\text{-joinable w.r.t. } \mathcal{R} \\ \text{no} & \text{if } \pi \text{ is not } \mu\text{-joinable w.r.t. } \mathcal{R} \end{cases}$$

External tools

1. We use tools like Prover9, AGES or Mace4.

Processors in the confluence framework

	$\mathsf{P}_{\mathsf{Simp}}$	P_{MD}	$\mathsf{P}_{\mathsf{Huet}}$	P_{HL}	P_{HN}	$\mathsf{P}_{\mathsf{CanJ}}$	$\mathsf{P}_{\mathsf{CanCR}}$	P_{U}
TRSs	✓	✓	✓	✓	✓	✓	✓	×
CS-TRSs	✓	×	✓	✓	✓	×	×	×
CTRSs	✓	×	✓	✓	✓	×	×	✓
Sound	✓	✓	×	✓	✓	✓	✓	✓
Complete	✓	✓	✓	✓	✓	×	×	×

Confluence proof tree

Confluence proof tree

Let $\mathcal{R}$ be a CTRS. A confluence proof tree $\mathcal{T}$ for $\mathcal{R}$ is a tree whose root label is $CR(\mathcal{R})$, whose inner nodes are labeled with problems, and whose leaves are labeled either with problems, “yes” or “no”. For every inner node n labeled with τ, there is a processor P such that $\tau \in \mathcal{D}om(\mathsf{P})$ and:

1. if $\mathsf{P}(\tau) =$ “no” then n has just one child, labeled with“no”.
2. if $\mathsf{P}(\tau) = \varnothing$ then n has just one child, labeled with “yes”.
3. if $\mathsf{P}(\tau) = \{\tau_1, \ldots, \tau_m\}$ with $m > 0$, then n has m children labeled with the problems $\tau_1, \ldots, \tau_m$.

Confluence framework

Let $\mathcal{R}$ be a CTRS and $\mathcal{T}$ be a confluence proof tree for $\mathcal{R}$. Then:

1. if all leaves in $\mathcal{T}$ are labeled with "yes" and all involved processors are sound for the problems they are applied to, then $\mathcal{R}$ is confluent.
2. if $\mathcal{T}$ has a leaf labeled with "no" and all processors in the path from the root to such a leaf are complete for the problems they are applied to, then $\mathcal{R}$ is non-confluent.

Strategy

- Processors can be used through a strategy.
- Strategies use combinators as and, or or try. This combinators can be parallelized.
- A simple **proof strategy** using the previous processors could be:
  1. we apply simplification and modular techniques;
  2. we analyze the system to extract good properties (external tools can be used to check termination and operational termination);
  3. we compute the set of conditional critical pairs;
  4. we apply the semantic based techniques to prove or disprove the joinability of the conditional critial pairs.

Feasibility

Example

$$
\begin{array}{rcl}
le(0, s(y)) & \to & true \\
le(s(x), s(y)) & \to & le(x, y) \\
le(x, 0) & \to & false \\
min(cons(x, nil)) & \to & x \\
min(cons(x, xs)) & \to & x \Leftarrow min(xs) \to^* y, le(x, y) \to^* true \\
min(cons(x, xs)) & \to & y \Leftarrow min(xs) \to^* y, le(x, y) \to^* false
\end{array}
$$

(Rf) $$\frac{}{x \to^* x}$$

(Re)$_\beta$ $$\frac{s_1 \to^* t_1 \quad \ldots \quad s_n \to^* t_n}{\ell \to r}$$

for $\beta : \ell \to r \Leftarrow s_1 \to^* t_1, \ldots, s_1 \to^* t_n \in \mathcal{R}$

(T) $$\frac{x \to y \qquad y \to^* z}{x \to^* z}$$

(C)$_{f,i}$ $$\frac{x_i \to y_i}{f(x_1, \ldots, x_i, \ldots, x_k) \to f(x_1, \ldots, y_i, \ldots, x_k)}$$

for $f \in \mathcal{F}^{(k)}$ and $1 \leq i \leq k$

First-Order Theory $\overline{\mathcal{R}}$ associated to $\mathcal{R}$

$$(\forall x)\ x \to^* x \tag{1}$$

$$(\forall x, y, z)\ x \to y \wedge y \to^* z \Rightarrow x \to^* z \tag{2}$$

$$(\forall x, y)\ x \to y \Rightarrow s(x) \to s(y) \tag{3}$$

$$(\forall x, y, z)\ x \to y \Rightarrow cons(x, z) \to cons(y, z) \tag{4}$$

$$(\forall x, y, z)\ x \to y \Rightarrow cons(z, x) \to cons(z, y) \tag{5}$$

$$(\forall x, y, z)\ x \to y \Rightarrow le(x, z) \to le(y, z) \tag{6}$$

$$(\forall x, y, z)\ x \to y \Rightarrow le(z, x) \to le(z, y) \tag{7}$$

$$(\forall x, y)\ x \to y \Rightarrow min(x) \to min(y) \tag{8}$$

$$(\forall y)\ le(0, s(y)) \to true \tag{9}$$

$$(\forall x, y)\ le(s(x), s(y)) \to le(x, y) \tag{10}$$

$$(\forall x)\ le(x, 0) \to false \tag{11}$$

$$(\forall x)\ min(cons(x, nil)) \to x \tag{12}$$

$$(\forall x, y, xs)\ min(xs) \to^* y \wedge le(x, y) \to^* true \Rightarrow min(cons(x, xs)) \to x \tag{13}$$

$$(\forall x, xs)\ min(xs) \to^* y \wedge le(x, y) \to^* false \Rightarrow min(cons(x, xs)) \to y \tag{14}$$

Proving Program Properties on $\overline{\mathcal{R}}$

Models

Given a first-order theory $\overline{\mathcal{R}}$ and a sentence φ, **finding a model** $\mathcal{A}$ of $\overline{\mathcal{R}} \cup \{\neg\varphi\}$ ($\mathcal{A} \models \overline{\mathcal{R}} \cup \{\neg\varphi\}$) shows that φ is **not** a logical consequence of $\overline{\mathcal{R}}$.

Therefore, $\mathcal{A} \models \overline{\mathcal{R}} \cup \{\neg\varphi\}$ **disproves** φ regarding $\overline{\mathcal{R}}$.

Feasibility

Definition

We say that the reachibility condition $s \to^* t$ is **feasible** if there is a substitution σ instantiating the variables in s and t such that the reachability test $\sigma(s) \to^* \sigma(t)$ **succeeds**; otherwise, we call it **infeasible**.

Uses of Infeasibility

1. disable the use of **conditional rules** in reductions,
2. discard **conditional dependency pairs** $u \to v \Leftarrow c$ in the analysis of operational termination of CTRSs,
3. discard **conditional critical pairs** $u \downarrow v \Leftarrow c$ that arise in the analysis of confluence of CTRSs,
4. prove **root-stability** of a term t,
5. prove **irreducibility** of ground terms t (which is undecidable for CTRSs),
6. prove the **non-joinability** of terms s and t as the infeasibility of $s \to^* x, t \to^* x$,
7. discard **arcs** in the **dependency graphs** that are obtained during the analysis of termination using dependency pairs.

Feasibility Framework

Feasibility Framework

fProblem

An **fProblem** τ is a pair $\tau = (\mathcal{R}, \mathcal{G})$, where $\mathcal{R}$ is a CTRS and $\mathcal{G}$ is a sequence $(s_i \to^* t_i)_{i=1}^n$. The **fProblem** τ is **feasible** if $\mathcal{G}$ is **$\mathcal{R}$-feasible**; otherwise it is **$\mathcal{R}$-infeasible**.

fProcessor

An **fProcessor** P is a partial function from fProblems into sets of fProblems. Alternatively, it can return "yes".

- An fProcessor P is **sound** if for all $\tau \in \mathcal{D}om(\mathsf{P})$, τ is feasible whenever either $\mathsf{P}(\tau) =$"yes" or $\exists \tau' \in \mathsf{P}(\tau)$, such that τ' is feasible.
- An fProcessor P is **complete** if for all $\tau \in \mathcal{D}om(\mathsf{P})$, τ is infeasible whenever $\forall \tau' \in \mathsf{P}(\tau)$, τ' is infeasible.

Feasibility Proof Tree (1/2)

Definition

Let τ be an fProblem. A **feasibility proof tree** $\mathcal{T}$ for τ is a tree whose inner nodes are labeled with fProblems and the leaves are labeled with **fProblems**, “yes” or “no”.

The root of $\mathcal{T}$ is labeled with τ and for every inner node τ', there is a fProcessor P such that $\tau' \in \mathcal{D}om(\mathsf{P})$ and:

1. if $\mathsf{P}(\tau') =$ “yes” then n has just **one child**, labeled with“yes”.
2. if $\mathsf{P}(\tau') = \varnothing$ then n has just **one child**, labeled with “no”.
3. if $\mathsf{P}(\tau') = \{\tau_1, \ldots, \tau_k\}$ with $k > 0$, then n has k **children** labeled with the fProblems $\tau_1, \ldots, \tau_k$.

Feasibility Proof Tree (2/2)

Theorem

Let $\mathcal{T}$ be a feasibility proof tree for $\tau_I = (\mathcal{R}, \mathcal{G})$. Then:

1. if all leaves in $\mathcal{T}$ are labeled with "no" and all involved fProcessors are **complete** for the fProblems they are applied to, then $\mathcal{G}$ is $\mathcal{R}$-**infeasible**.
2. if $\mathcal{T}$ has a leaf labeled with "yes" and all fProcessors in the path from τ_I to the leaf are **sound** for the fProblems they are applied to, then $\mathcal{G}$ is $\mathcal{R}$-**feasible**.

Satisfiability fProcessor

Theorem

Let $\tau = (\mathcal{R}, \mathcal{G})$ be an fProblem with $\mathcal{G} = (s_i \to^* t_i)_{i=1}^{n}$. Let $\mathcal{A}$ be a structure such that $\mathcal{A} \neq \varnothing$ and $\mathcal{A} \models \overline{\mathcal{R}} \cup \{\neg(\exists\vec{x}) \bigwedge_{i=1}^{n} s_i \to^* t_i\}$. The fProcessor $\mathsf{P}_{\mathsf{Sat}}$ given by $\mathsf{P}_{\mathsf{Sat}}(\tau) = \varnothing$ is **sound** and **complete**.

Example 903.trs - Satisfiability fProcessor

$$(Rf), (T), (C)_{f,i}$$

$$(\forall y)\ le(0, s(y)) \to true$$

$$(\forall x, y)\ le(s(x), s(y)) \to le(x, y)$$

$$(\forall x)\ le(x, 0) \to false$$

$$(\forall x)\ min(cons(x, nil)) \to x$$

$$(\forall x, y, xs)\ min(xs) \to^* y \wedge$$

$$le(x, y) \to^* true \Rightarrow min(cons(x, xs)) \to x$$

$$(\forall x, xs)\ min(xs) \to^* y \wedge$$

$$le(x, y) \to^* false \Rightarrow min(cons(x, xs)) \to y$$

$$\neg((\exists x, y) min(nil) \to^* x \wedge le(y, x) \to^* true)$$

- AGES output:

```
Domain: |N U {-1}

Function Interpretations:
[le(x,y)] = y        [0] = 1
[min(x)] = x         [false] = 1
[s(x)] = 1 + x       [nil] =  - 1
[cons(x,y)] = 1 + x + y
[true] = 0

Predicate Interpretations:
x ->* y <=> (x >= y)
x -> y <=> ((x >= y) /\
            (1 + y >= 0))
```

Example 903.trs - Satisfiability fProcessor

$$(Rf), (T), (C)_{f,i}$$

$$(\forall y)\ le(0, s(y)) \to true$$

$$(\forall x, y)\ le(s(x), s(y)) \to le(x, y)$$

$$(\forall x)\ le(x, 0) \to false$$

$$(\forall x)\ min(cons(x, nil)) \to x$$

$$(\forall x, y, xs)\ min(xs) \to^* y \wedge$$
$$le(x, y) \to^* true \Rightarrow min(cons(x, xs)) \to x$$

$$(\forall x, xs)\ min(xs) \to^* y \wedge$$
$$le(x, y) \to^* false \Rightarrow min(cons(x, xs)) \to y$$

$$(\forall x, y)\ \neg(min(nil) \to^* x) \vee \neg(le(y, x) \to^* true)$$

- AGES output:

```
Domain: |N U {-1}

Function Interpretations:
[le(x,y)] = y        [0] = 1
[min(x)] = x         [false] = 1
[s(x)] = 1 + x       [nil] =  - 1
[cons(x,y)] = 1 + x + y
[true] = 0

Predicate Interpretations:
x ->* y <=> (x >= y)
x -> y <=> ((x >= y) /\
            (1 + y >= 0))
```

Provability fProcessor

Theorem

Let $\tau = (\mathcal{R}, \mathcal{G})$ be an fProblem with $\mathcal{G} = (s_i \to^* t_i)_{i=1}^n$ such that $\overline{\mathcal{R}} \vdash \ (\exists \vec{x}) \ \bigwedge_{i=1}^n s_i \to^* t_i$ holds. The fProcessor $\mathsf{P}_{\mathsf{Prov}}$ given by $\mathsf{P}_{\mathsf{Prov}}(\tau) =$ "yes" is **sound** and **complete**.

Example 836.trs - Provability fProcessor (Prover9 output)

```
(1) (x ->* x) # [reflexivity]
(2) -(x -> y) | -(y ->* z) | (x ->* z) # [transitivity]
(7) -(x -> y) | le(z,x) -> le(z,y) # [congruence]
(11) le(x,0) -> false # [replacement]
(12) min(cons(x,nil)) -> x # [replacement]
(16) (exists x (exists y (le(x,min(y)) ->* false &
     min(y) ->* x))) # [goal]
-----------------------------------------------------
(17) -(le(x,min(y)) ->* false) |
     -(min(y) ->* x) # [deny(16)]
(18) le(x,0) ->* false [ur(2,11,1)]
(19) -(le(min(x),min(x)) ->* false) [resolve(17,1)]
(20) -(le(min(x),min(x)) -> y) |
     -(y ->* false) [resolve(19,2)]
(21) -(le(min(x),y) ->* false) |
     -(min(x) -> y) [resolve(20,7)]
(22) -(le(min(cons(x,nil)),x) ->* false) [resolve(21,12)]
(23) $F [resolve(22,18)]
```

Definition

Let $\tau = (\mathcal{R}, \mathcal{G})$ be an fProblem, $s_i \to^* t_i \in \mathcal{G}$, and $\mathcal{N} \subseteq \overline{\mathcal{N}}(\mathcal{R}, \mathcal{G}, i)$ finite. $\mathsf{P}_{\mathsf{NarrCond}}$ is given by $\mathsf{P}_{\mathsf{NarrCond}}(\tau) = \{(\mathcal{R}, \mathcal{N})\}$.

Theorem

$\mathsf{P}_{\mathsf{NarrCond}}$ is sound. If $\mathcal{N} = \overline{\mathcal{N}}(\mathcal{R}, \mathcal{G}, i)$ and $s_i \to^* t_i \in \mathcal{G}$ is such that s_i and t_i do *not* unify and either s_i is *ground* or (1) $\mathit{NRules}(\mathcal{R}, s_i)$ is a TRS and (2) s_i is linear, then $\mathsf{P}_{\mathsf{NarrCond}}$ is complete.

Example 896.trs - Narrowing fProcessor

Example (903.trs)

$$
\begin{array}{ll}
add(0,x) \to x & add(s(x),y) \to s(add(x,y)) \\
 & \quad div(minus(x,y),s(y)) \to^* q \\
lte(0,y) \to true & lte(s(x),0) \to false \\
lte(s(x),s(y)) \to lte(x,y) & minus(0,s(y)) \to 0 \\
minus(s(x),s(y)) \to minus(x,y) & minus(x,0) \to x \\
mod(0,y) \to 0 & mod(x,0) \to x \\
mod(x,s(y)) \to mod(minus(x,s(y)),s(y)) & \Leftarrow lte(s(y),x) \to^* true \\
mod(x,s(y)) \to x & \Leftarrow lte(s(y),x) \to^* false \\
mult(0,y) \to 0 & mult(s(x),y) \to add(mult(x,y),y) \\
div(s(x),s(y)) \to 0 & \Leftarrow lte(s(x),y) \to^* true \\
div(s(x),s(y)) \to s(q) & \Leftarrow lte(s(x),y) \to^* false, \\
div(0,s(x)) \to 0 & power(x,0) \to s(0) \\
power(x,n) \to mult(mult(y,y),s(0)) & \Leftarrow n \to^* s(n'), mod(n,s(s(0))) \to^* 0, \\
 & \quad power(x,div(n,s(s(0)))) \to^* y \\
power(x,n) \to mult(mult(y,y),x) & \Leftarrow n \to^* s(n'), mod(n,s(s(0))) \to^* s(z), \\
 & \quad power(x,div(n,s(s(0)))) \to^* y
\end{array}
$$

Example 896.trs - Narrowing fProcessor

Example (903.trs)

$$
\begin{array}{ll}
add(0,x) \to x & add(s(x),y) \to s(add(x,y)) \\
 & \quad div(minus(x,y),s(y)) \to^* q \\
lte(0,y) \to true & lte(s(x),0) \to false \\
lte(s(x),s(y)) \to lte(x,y) & minus(0,s(y)) \to 0 \\
minus(s(x),s(y)) \to minus(x,y) & minus(x,0) \to x \\
mod(0,y) \to 0 & \\
mod(x,s(y)) \to & \\
mod(x,s(y)) \to & \\
mult(0,y) \to 0 & x,y),y) \\
div(s(x),s(y)) \to 0 & \Leftarrow lte(s(x),y) \to^* true \\
div(s(x),s(y)) \to s(q) & \Leftarrow lte(s(x),y) \to^* false, \\
div(0,s(x)) \to 0 & power(x,0) \to s(0) \\
power(x,n) \to mult(mult(y,y),s(0)) & \Leftarrow n \to^* s(n'), mod(n,s(s(0))) \to^* 0, \\
 & \quad power(x,div(n,s(s(0)))) \to^* y \\
power(x,n) \to mult(mult(y,y),x) & \Leftarrow n \to^* s(n'), mod(n,s(s(0))) \to^* s(z), \\
 & \quad power(x,div(n,s(s(0)))) \to^* y
\end{array}
$$

Feasibility Conditions

$$\{lte(s(x),0) \to^* true\}$$

Example 896.trs - Narrowing fProcessor

Example (903.trs)

$add(0,x) \to x$ | $add(s(x),y) \to s(add(x,y))$

$div(minus(x,y),s(y)) \to^* q$

$minus(s(x),s(y)) \to minus(x,y)$ | $minus(x,0) \to x$

$mod(0,y) \to 0$ | $mod(x,0) \to x$

$mod(x,s(y)) \to mod(minus(x,s(y)),s(y)) \Leftarrow lte(s(y),x) \to^* true$

$mod(x,s(y)) \to$ …

$mult(0,y) \to 0$ … $x,y),y)$

$div(s(x),s(y)) \to$ …

$div(s(x),s(y)) \to$ …

$div(0,s(x)) \to 0$ | $power(x,0) \to s(0)$

$power(x,n) \to mult(mult(y,y),s(0)) \Leftarrow n \to^* s(n'), mod(n,s(s(0))) \to^* 0, power(x,div(n,s(s(0)))) \to^* y$

$power(x,n) \to mult(mult(y,y),x) \Leftarrow n \to^* s(n'), mod(n,s(s(0))) \to^* s(z), power(x,div(n,s(s(0)))) \to^* y$

Feasibility Conditions

$$\{false \to^* true\}$$

Example 896.trs - Satisfiability fProcessor (Mace4 output)

```
Domain: {0,1}

Function Interpretations:
[false] = 0          [true] = 1           [0] = 0
[s](0) = 0           [s](1) = 1           [add](0,0) = 0
[add](0,1) = 1       [add](1,0) = 0       [add](1,1) = 1
[div](0,0) = 0       [div](0,1) = 0       [div](1,0) = 0
[div](1,1) = 0       [lte](0,0) = 1       [lte](0,1) = 1
[lte](1,0) = 1       [lte](1,1) = 1       [minus](0,0) = 0
[minus](0,1) = 0     [minus](1,0) = 1     [minus](1,1) = 1
[mod](0,0) = 0       [mod](0,1) = 0       [mod](1,0) = 1
[mod](1,1) = 1       [mult](0,0) = 0      [mult](0,1) = 1
[mult](1,0) = 0      [mult](1,1) = 1      [power](0,0) = 0
[power](0,1) = 0     [power](1,0) = 1     [power](1,1) = 1
Predicate Interpretations:
0 ->* 0 = true       0 ->* 1 = false      1 ->* 0 = true
1 ->* 1 = true       0 -> 0 = true        0 -> 1 = false
1 -> 0 = true        1 -> 1 = true
```

Strategy

Strategy

1. we try to prove feasibility using P_{Prov};
2. if P_{Prov} fails, we apply P_{Sat};
3. if P_{Sat} fails, we apply P_{NarrCond};
4. if P_{NarrCond} succeeds and modifies the feasibility sequence, we go to Item 2, otherwise we return MAYBE.