Termersetzungssysteme
Vorlesung 5

Stephan Falke

Verifikation trifft Algorithmik
Karlsruher Institut für Technologie (KIT)

05.07.2010

Reduktionsordnungen

Definition

Eine fundierte transitive Relation $\succ$ auf $\mathcal{T}(\Sigma, \mathcal{V})$ ist eine Reduktionsordnung gdw.

1. $\succ$ ist stabil, d.h., für alle $s_1 \succ s_2$ und alle Substitutionen σ gilt $s_1\sigma \succ s_2\sigma$
2. $\succ$ ist monoton, d.h., für alle $s_1 \succ s_2$ und alle $f \in \Sigma$ mit Stelligkeit n gilt

$$f(t_1, \ldots, t_{i-1}, s_1, t_{i+1}, \ldots, t_n) \succ f(t_1, \ldots, t_{i-1}, s_2, t_{i+1}, \ldots, t_n)$$

für alle $1 \leq i \leq n$ und alle $t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n$

Polynominterpretationen

- Terme werden auf Polynome mit Koeffizienten aus $\mathbb{N}$ abgebildet
- Eine Polynominterpretation weist jedem $f \in \Sigma$ mit Stelligkeit n ein Polynom $\mathcal{P}ol(f) \in \mathbb{N}[X_1, \ldots, X_n]$ zu
- Dies weist auch jedem Term ein Polynom zu:
  - $\mathcal{P}ol(x) = x$
  - $\mathcal{P}ol(f(t_1, \ldots, t_n)) = \mathcal{P}ol(f)(\mathcal{P}ol(t_1), \ldots, \mathcal{P}ol(t_n))$
- Polynomordnung: $s \succ_{\mathcal{P}ol} t$ gdw. $\mathcal{P}ol(s\sigma) - \mathcal{P}ol(t\sigma) > 0$ für alle Instanziierungen der Variablen durch Grundterme (d.h., Terme ohne Variablen)
- Hinreichende Bedingung: $\mathcal{P}ol(s) - \mathcal{P}ol(t) > 0$ für alle Instanziierungen der Variablen durch natürliche Zahlen

Polynomordnungen als Reduktionsordnungen

Definition

Ein Polynom p ist monoton in X_i gdw.

$$p(x_1, \ldots, x_i + 1, \ldots, x_n) - p(x_1, \ldots, x_i, \ldots, x_n) > 0$$

für alle Instanziierungen der Variablen durch natürliche Zahlen
Eine Polynominterpretation $\mathcal{P}ol$ ist monoton gdw. $\mathcal{P}ol(f)$ für alle $f \in \Sigma$ monoton in allen X_i ist

Satz

Für eine monotone Polynominterpretation $\mathcal{P}ol$ ist $\succ_{\mathcal{P}ol}$ eine Reduktionsordnung

Schwächen

Beispiel

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

Terminierung kann nicht mit lexikographischer Pfadordnung gezeigt werden

Terminierung kann nicht mit Multimengen-Pfadordnung gezeigt werden

Terminierung kann nicht mit monotonen Polynominterpretationen gezeigt werden

Dependency Pairs: Idee

- Wenn $\mathcal{R}$ nicht terminierend ist dann existiert eine unendliche Folge von rekursiven Funktionsaufrufen
- Untersuche alle möglichen Aufruffolgen
- $\mathcal{R}$ ist terminierend gdw. alle Aufruffolgen endlich sind
- Techniken zur Analyse von Aufruffolgen:
  - SCC-Zerlegung
  - Reduktionsordnungen
  - nicht-monotone Polynominterpretationen
  - ...

Dependency Pairs

Definition

Definierte Funktionen $\mathcal{D}(\mathcal{R}) = \{f \mid f = \mathrm{root}(l) \text{ für ein } l \to r \in \mathcal{R}\}$
Konstruktoren $\mathcal{C}(\mathcal{R}) = \Sigma - \mathcal{D}(\mathcal{R})$

Definition

Für $l \to r \in \mathcal{R}$ und einen Teilterm t von r mit $\mathrm{root}(t) \in \mathcal{D}(\mathcal{R})$ ist

$$l^\sharp \to t^\sharp$$

ein Dependency Pair von $\mathcal{R}$
Für $t = f(t_1, \ldots, t_n)$ ist $t^\sharp = f^\sharp(t_1, \ldots, t_n)$ (Tupelsymbole)
$\mathrm{DP}(\mathcal{R})$: Dependency Pairs von $\mathcal{R}$

Beispiel

$$
\begin{aligned}
\mathrm{minus}(\mathcal{O}, y) &\to \mathcal{O} \\
\mathrm{minus}(\mathrm{s}(x), \mathcal{O}) &\to \mathrm{s}(x) \\
\mathrm{minus}(\mathrm{s}(x), \mathrm{s}(y)) &\to \mathrm{minus}(x, y) \\
\mathrm{div}(\mathcal{O}, \mathrm{s}(y)) &\to \mathcal{O} \\
\mathrm{div}(\mathrm{s}(x), \mathrm{s}(y)) &\to \mathrm{s}(\mathrm{div}(\mathrm{minus}(x, y), \mathrm{s}(y)))
\end{aligned}
$$

Dependency Pairs $\mathrm{DP}(\mathcal{R})$:

$$
\begin{aligned}
\mathrm{minus}^\sharp(\mathrm{s}(x), \mathrm{s}(y)) &\to \mathrm{minus}^\sharp(x, y) \\
\mathrm{div}^\sharp(\mathrm{s}(x), \mathrm{s}(y)) &\to \mathrm{div}^\sharp(\mathrm{minus}(x, y), \mathrm{s}(y)) \\
\mathrm{div}^\sharp(\mathrm{s}(x), \mathrm{s}(y)) &\to \mathrm{minus}^\sharp(x, y)
\end{aligned}
$$

Terminierungsanalyse mit Dependency Pairs

Definition

Eine (endliche oder unendliche) Folge $s_1 \to t_1, s_2 \to t_2, \ldots$ von Dependency Pairs ist eine $(\mathrm{DP}(\mathcal{R}), \mathcal{R})$-Kette gdw. es eine Substitution σ gibt so dass

- $t_i\sigma \to_{\mathcal{R}}^* s_{i+1}\sigma$ für alle benachbarten Dependency Pairs $s_i \to t_i, s_{i+1} \to t_{i+1}$
- kein $t_i\sigma$ eine unendliche $\to_{\mathcal{R}}$-Reduktion hat

Satz

$\mathcal{R}$ ist terminierend gdw. es keine unendlichen $(\mathrm{DP}(\mathcal{R}), \mathcal{R})$-Ketten gibt

DP Prozessoren

- DP Prozessoren erlauben flexible und modulare Terminierungsbeweise
- DP Problem: Eine Menge von Dependency Pairs
- DP Prozessor Proc: $\mathcal{P} \mapsto \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$
  - Korrektheit: es existieren keine unendlichen $(\mathcal{P}, \mathcal{R})$-Ketten wenn für kein i unendliche $(\mathcal{P}_i, \mathcal{R})$-Ketten existieren

Algorithmus

$todo := \{\mathrm{DP}(\mathcal{R})\}$;
while $todo \neq \emptyset$ **do**
    $\mathcal{P} := choose\_one(todo)$;
    $\mathrm{Proc} := choose\_proc(\mathcal{P})$;
    $todo := todo - \{\mathcal{P}\} \cup \mathrm{Proc}(\mathcal{P})$;
end
return "$\mathcal{R}$ terminiert";

Dependency Graph

- Idee: Baue einen Graph der zeigt welche Dependency Pairs hintereinander in einer Kette auftreten können
  - Eine unendliche Kette entspricht einem unendlichen Pfad in diesem Graph
  - Nach einem endlichen Anfangsstück bleibt dieser unendliche Pfad in einem SCC des Graphs

Definition

Für eine Menge $\mathcal{P}$ von Dependency Pairs ist der Dependency Graph $\mathrm{DG}(\mathcal{P}) = (V, E)$ der gerichtete Graph mit

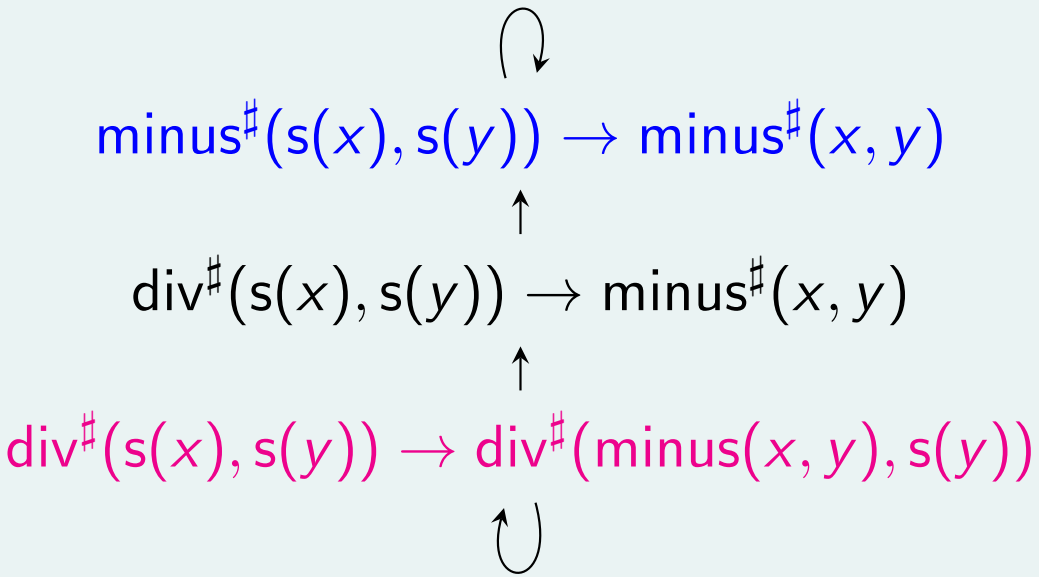
- $V = \mathcal{P}$
- $E = \{(s_1 \to t_1, s_2 \to t_2) \mid s_1 \to t_1, s_2 \to t_2 \text{ ist eine } (\mathcal{P}, \mathcal{R})\text{-Kette}\}$

- $\mathrm{DG}(\mathcal{P})$ kann effizient approximiert werden

Beispiel

Dependency Pairs $\mathrm{DP}(\mathcal{R})$:

$$
\begin{array}{lrcl}
(1) & \mathsf{minus}^\sharp(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{minus}^\sharp(x,y) \\
(2) & \mathsf{div}^\sharp(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{div}^\sharp(\mathsf{minus}(x,y),\mathsf{s}(y)) \\
(3) & \mathsf{div}^\sharp(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{minus}^\sharp(x,y)
\end{array}
$$



SCC-Zerlegung

Satz

Seien $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ *die SCCs von* $\mathrm{DG}(\mathcal{P})$
Dann ist der DP Prozessor mit $\mathrm{Proc}(\mathcal{P}) = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ *korrekt*

Beispiel

$$\mathrm{Proc}(\mathcal{P}) = \{\ \{\mathsf{minus}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{minus}^\sharp(x, y)\},\ \{\mathsf{div}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{div}^\sharp(\mathsf{minus}(x, y), \mathsf{s}(y))\}\ \}$$

Vorteil: Die neuen DP Probleme können unabhängig voneinander bearbeitet werden

Reduktionspaare

- In Kombination mit Dependency Pairs können die Bedingungen an Reduktionsordnungen abgeschwächt werden

Definition

Ein Reduktionspaar $(\succeq, >)$ besteht aus Relationsn $\succeq$ und $>$ so dass

- $\succeq$ ist reflexiv, transitiv, monoton und stabil
- $>$ ist fundiert und stabil
- $>$ ist kompatibel mit $\succeq$, d.h., $\succeq \circ > \circ \succeq \subseteq >$

Nutzen von Reduktionspaaren

- Idee: Falls
  - $s > t$ für alle $s \to t \in \mathcal{P}$
  - $l \succeq r$ für alle $l \to r \in \mathcal{R}$

  so gibt es keine unendlichen $(\mathcal{P}, \mathcal{R})$-Ketten (warum?)

Satz

Unter diesen Bedinungen ist $\mathrm{Proc}(\mathcal{P}) = \emptyset$ *korrekt*

- Verfeinerung: Falls
  - $s > t$ für alle $s \to t \in \mathcal{P}' \subseteq \mathcal{P}$
  - $s \succeq t$ für alle $s \to t \in \mathcal{P} - \mathcal{P}'$ und $l \succeq r$ für alle $l \to r \in \mathcal{R}$

  so gibt es unendlichen $(\mathcal{P}, \mathcal{R})$-Ketten nur falls es unendliche $(\mathcal{P} - \mathcal{P}', \mathcal{R})$-Ketten gibt (warum?)

Satz

Unter diesen Bedinungen ist $\mathrm{Proc}(\mathcal{P}) = \{\mathcal{P} - \mathcal{P}'\}$ *korrekt*

Polynominterpretationen als Reduktionspaare

- Erinnerung: $s \succ_{\mathcal{P}ol} t$ wenn $\mathcal{P}ol(s) - \mathcal{P}ol(t) > 0$ für alle Instanziierungen der Variablen durch natürliche Zahlen
- Analog: $s \succeq_{\mathcal{P}ol} t$ wenn $\mathcal{P}ol(s) - \mathcal{P}ol(t) \geq 0$ für alle Instanziierungen der Variablen durch natürliche Zahlen

Satz

Für eine Polynominterpretation $\mathcal{P}ol$ ist $(\succeq_{\mathcal{P}ol}, \succ_{\mathcal{P}ol})$ ein Reduktionspaar

- Beachte: $\mathcal{P}ol$ muss nicht monoton sein

Beispiel

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

$$
\begin{array}{rcl|rcl}
\mathsf{minus}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{minus}^\sharp(x, y) & \mathsf{div}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{div}^\sharp(\mathsf{minus}(x, y), \mathsf{s}(y)) \\
\mathcal{P}ol(\mathsf{minus}^\sharp) & = & X_1 & \mathcal{P}ol(\mathsf{div}^\sharp) & = & X_1 \\
\mathcal{P}ol(\mathcal{O}) & = & 0 & \mathcal{P}ol(\mathcal{O}) & = & 0 \\
\mathcal{P}ol(\mathsf{s}) & = & X_1 + 1 & \mathcal{P}ol(\mathsf{s}) & = & X_1 + 1 \\
\mathcal{P}ol(\mathsf{minus}) & = & X_1 & \mathcal{P}ol(\mathsf{minus}) & = & X_1 \\
\mathcal{P}ol(\mathsf{div}) & = & X_1 & \mathcal{P}ol(\mathsf{div}) & = & X_1
\end{array}
$$

In beiden Fällen: $\mathrm{Proc}(\ldots) = \{\emptyset\}$ (was nun?)

Reduktionsordnungen als Reduktionspaare 1/2

- Reduktionsordnungen sind monoton, aber in Reduktionspaaren muss nur $\succeq$ monoton sein
- Idee: Führe eine Vorverarbeitung durch so dass Reduktionsordnungen diesen Vorteil nutzen können

Definition

Ein Argumentfilter π bildet jedes $f \in \Sigma$ mit $\mathrm{ar}(f) = n$ auf ein $i \in \{1, \ldots, n\}$ oder eine (möglicherweise leere) Liste $[i_1, \ldots, i_m]$ mit $1 \leq i_1 < \ldots < i_m \leq n$ ab

$$\pi(t) = \begin{cases} t & \text{falls } t \in \mathcal{V} \\ \pi(t_i) & \text{falls } t = f(t_1, \ldots, t_n) \text{ und } \pi(f) = i \\ f(\pi(t_{i_1}), \ldots, \pi(t_{i_m})) & \text{falls } t = f(t_1, \ldots, t_n) \text{ und } \pi(f) = [i_1, \ldots, i_m] \end{cases}$$

Reduktionsordnungen als Reduktionspaare 2/2

Definition

Für einen Argumentfilter π und eine Reduktionsordnung $\succ$ definiere

- $s \succ_\pi t$ gdw. $\pi(s) \succ \pi(t)$
- $s \succeq_\pi t$ gdw. $\pi(s) \succ \pi(t)$ oder $\pi(s) = \pi(t)$

Satz

Für einen Argumentfilter π und eine Reduktionsordnung $\succ$ ist $(\succeq_\pi, \succ_\pi)$ ein Reduktionspaar

Beispiel

$$
\begin{array}{rcl}
\mathsf{minus}(\mathcal{O}, y) & \to & \mathcal{O} \\
\mathsf{minus}(\mathsf{s}(x), \mathcal{O}) & \to & \mathsf{s}(x) \\
\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{minus}(x, y) \\
\mathsf{div}(\mathcal{O}, \mathsf{s}(y)) & \to & \mathcal{O} \\
\mathsf{div}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{div}(\mathsf{minus}(x, y), \mathsf{s}(y))) \\
\mathsf{minus}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{minus}^\sharp(x, y) \\
\mathsf{div}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{div}^\sharp(\mathsf{minus}(x, y), \mathsf{s}(y))
\end{array}
$$

Argumentfilter π mit $\pi(\mathsf{minus}) = \pi(\mathsf{div}) = \pi(\mathsf{minus}^\sharp) = \pi(\mathsf{div}^\sharp) = 1$

$$
\begin{array}{rcl}
\mathcal{O} & \succeq & \mathcal{O} \\
\mathsf{s}(x) & \succeq & \mathsf{s}(x) \\
\mathsf{s}(x) & \succeq & x \\
\mathcal{O} & \succeq & \mathcal{O} \\
\mathsf{s}(x) & \succeq & \mathsf{s}(x) \\
\mathsf{s}(x) & > & x \\
\mathsf{s}(x) & > & x
\end{array}
$$

Erfüllt von der Einbettungsordnung!

Beispiel

$$
\begin{array}{rcl}
\mathcal{O} \leq y & \to & \mathsf{true} \\
\mathsf{s}(x) \leq \mathcal{O} & \to & \mathsf{false} \\
\mathsf{s}(x) \leq \mathsf{s}(y) & \to & x \leq y \\
\mathsf{app}(\mathsf{nil}, y) & \to & y \\
\mathsf{app}(\mathsf{cons}(n, x), y) & \to & \mathsf{cons}(n, \mathsf{app}(x, y)) \\
\mathsf{low}(n, \mathsf{nil}) & \to & \mathsf{nil} \\
\mathsf{low}(n, \mathsf{cons}(m, x)) & \to & \mathsf{if}_{\mathsf{low}}(m \leq n, n, \mathsf{cons}(m, x)) \\
\mathsf{if}_{\mathsf{low}}(\mathsf{true}, n, \mathsf{cons}(m, x)) & \to & \mathsf{cons}(m, \mathsf{low}(n, x)) \\
\mathsf{if}_{\mathsf{low}}(\mathsf{false}, n, \mathsf{cons}(m, x)) & \to & \mathsf{low}(n, x) \\
\mathsf{high}(n, \mathsf{nil}) & \to & \mathsf{nil} \\
\mathsf{high}(n, \mathsf{cons}(m, x)) & \to & \mathsf{if}_{\mathsf{high}}(m \leq n, n, \mathsf{cons}(m, x)) \\
\mathsf{if}_{\mathsf{high}}(\mathsf{true}, n, \mathsf{cons}(m, x)) & \to & \mathsf{high}(n, x) \\
\mathsf{if}_{\mathsf{high}}(\mathsf{false}, n, \mathsf{add}(m, x)) & \to & \mathsf{cons}(m, \mathsf{high}(n, x)) \\
\mathsf{qsort}(\mathsf{nil}) & \to & \mathsf{nil} \\
\mathsf{qsort}(\mathsf{cons}(n, x)) & \to & \mathsf{app}(\mathsf{qsort}(\mathsf{low}(n, x)), \mathsf{cons}(n, \mathsf{qsort}(\mathsf{high}(n, x))))
\end{array}
$$

5 SCCs:

$$
\begin{array}{l|l}
\mathsf{s}(x) \leq^\sharp \mathsf{s}(y) \to x \leq^\sharp y & \mathsf{app}^\sharp(\mathsf{cons}(n, x), y) \to \mathsf{app}^\sharp(x, y) \\
\hline
\mathsf{low}^\sharp(n, \mathsf{cons}(m, x)) \to \mathsf{if}^\sharp_{\mathsf{low}}(m \leq n, n, \mathsf{cons}(m, x)) & \mathsf{high}^\sharp(n, \mathsf{cons}(m, x)) \to \mathsf{if}^\sharp_{\mathsf{high}}(m \leq n, n, \mathsf{cons}(m, x)) \\
\mathsf{if}^\sharp_{\mathsf{low}}(\mathsf{true}, n, \mathsf{cons}(m, x)) \to \mathsf{low}^\sharp(n, x) & \mathsf{if}^\sharp_{\mathsf{high}}(\mathsf{true}, n, \mathsf{cons}(m, x)) \to \mathsf{high}^\sharp(n, x) \\
\mathsf{if}^\sharp_{\mathsf{low}}(\mathsf{false}, n, \mathsf{cons}(m, x)) \to \mathsf{low}^\sharp(n, x) & \mathsf{if}^\sharp_{\mathsf{high}}(\mathsf{false}, n, \mathsf{add}(m, x)) \to \mathsf{high}^\sharp(n, x) \\
\hline
\mathsf{qsort}^\sharp(\mathsf{cons}(n, x)) \to \mathsf{qsort}^\sharp(\mathsf{low}(n, x)) & \\
\mathsf{qsort}^\sharp(\mathsf{cons}(n, x)) \to \mathsf{qsort}^\sharp(\mathsf{high}(n, x)) &
\end{array}
$$

Der erste SCC kann erfolgreich behandelt werden
Bei den übrigen SCCs ist es nicht möglich die Bedingungen an ein Reduktionspaar zu erfüllen ($l \succeq r$ für alle $l \to r \in \mathcal{R}$)

Funktionsabhängigkeiten

- Wunsch: Anstatt $l \succeq r$ für alle $l \to r \in \mathcal{R}$ fordere dies nur für die Regeln von denen $\mathcal{P}$ "abhängt"

Definition

Seien $f, g \in \mathcal{D}(\mathcal{R})$. Dann

- $\mathcal{P} \sqsupset f$ falls f in t vorkommt für ein $s \to t \in \mathcal{P}$
- $f \sqsupset g$ falls g in r vorkommt für ein $l \to r \in \mathcal{R}$ mit $\mathrm{root}(l) = f$

$\Delta(\mathcal{P}) = \{f \mid \mathcal{P} \sqsupset^+ f\}$

$\mathcal{R}(\Delta(\mathcal{P})) = \{l \to r \in \mathcal{R} \mid \mathrm{root}(l) \in \Delta(\mathcal{P})\}$

Nutzen von Funktionsabhängigkeiten

Satz

Sei $(\succeq, >)$ *ein Reduktionspaar so dass*

- $s > t$ *für alle* $s \to t \in \mathcal{P}' \subseteq \mathcal{P}$
- $s \succeq t$ *für alle* $s \to t \in \mathcal{P} - \mathcal{P}'$
- $l \succeq r$ *für alle* $l \to r \in \mathcal{R}(\Delta(\mathcal{P}))$

Dann ist $\mathrm{Proc} : \mathcal{P} \mapsto \{\mathcal{P} - \mathcal{P}'\}$ *korrekt*

- Korrektheitsbeweis ist nicht-trivial da beliebige Regeln aus $\mathcal{R}$ in $(\mathcal{P}, \mathcal{R})$-Ketten verwendet werden können
- Effizienter und mächtiger als die vorherige Methode

Beispiel

$$
\begin{array}{rcl}
\mathcal{O} \leq y & \to & \mathsf{true} \\
\mathsf{s}(x) \leq \mathcal{O} & \to & \mathsf{false} \\
\mathsf{s}(x) \leq \mathsf{s}(y) & \to & x \leq y \\
\mathsf{app}(\mathsf{nil}, y) & \to & y \\
\mathsf{app}(\mathsf{cons}(n, x), y) & \to & \mathsf{cons}(n, \mathsf{app}(x, y)) \\
\mathsf{low}(n, \mathsf{nil}) & \to & \mathsf{nil} \\
\mathsf{low}(n, \mathsf{cons}(m, x)) & \to & \mathsf{if}_{\mathsf{low}}(m \leq n, n, \mathsf{cons}(m, x)) \\
\mathsf{if}_{\mathsf{low}}(\mathsf{true}, n, \mathsf{cons}(m, x)) & \to & \mathsf{cons}(m, \mathsf{low}(n, x)) \\
\mathsf{if}_{\mathsf{low}}(\mathsf{false}, n, \mathsf{cons}(m, x)) & \to & \mathsf{low}(n, x) \\
\mathsf{high}(n, \mathsf{nil}) & \to & \mathsf{nil} \\
\mathsf{high}(n, \mathsf{cons}(m, x)) & \to & \mathsf{if}_{\mathsf{high}}(m \leq n, n, \mathsf{cons}(m, x)) \\
\mathsf{if}_{\mathsf{high}}(\mathsf{true}, n, \mathsf{cons}(m, x)) & \to & \mathsf{high}(n, x) \\
\mathsf{if}_{\mathsf{high}}(\mathsf{false}, n, \mathsf{add}(m, x)) & \to & \mathsf{cons}(m, \mathsf{high}(n, x)) \\
\mathsf{qsort}(\mathsf{nil}) & \to & \mathsf{nil} \\
\mathsf{qsort}(\mathsf{cons}(n, x)) & \to & \mathsf{app}(\mathsf{qsort}(\mathsf{low}(n, x)), \mathsf{cons}(n, \mathsf{qsort}(\mathsf{high}(n, x))))
\end{array}
$$

$\mathsf{s}(x) \leq^\sharp \mathsf{s}(y) \to x \leq^\sharp y$	$\mathsf{app}^\sharp(\mathsf{cons}(n, x), y) \to \mathsf{app}^\sharp(x, y)$
$\mathsf{low}^\sharp(n, \mathsf{cons}(m, x)) \to \mathsf{if}^\sharp_{\mathsf{low}}(m \leq n, n, \mathsf{cons}(m, x))$ $\mathsf{if}^\sharp_{\mathsf{low}}(\mathsf{true}, n, \mathsf{cons}(m, x)) \to \mathsf{low}^\sharp(n, x)$ $\mathsf{if}^\sharp_{\mathsf{low}}(\mathsf{false}, n, \mathsf{cons}(m, x)) \to \mathsf{low}^\sharp(n, x)$	$\mathsf{high}^\sharp(n, \mathsf{cons}(m, x)) \to \mathsf{if}^\sharp_{\mathsf{high}}(m \leq n, n, \mathsf{cons}(m, x))$ $\mathsf{if}^\sharp_{\mathsf{high}}(\mathsf{true}, n, \mathsf{cons}(m, x)) \to \mathsf{high}^\sharp(n, x)$ $\mathsf{if}^\sharp_{\mathsf{high}}(\mathsf{false}, n, \mathsf{add}(m, x)) \to \mathsf{high}^\sharp(n, x)$
$\mathsf{qsort}^\sharp(\mathsf{cons}(n, x)) \to \mathsf{qsort}^\sharp(\mathsf{low}(n, x))$ $\mathsf{qsort}^\sharp(\mathsf{cons}(n, x)) \to \mathsf{qsort}^\sharp(\mathsf{high}(n, x))$	

Für die ersten beiden SCCs ist $\Delta(\mathcal{P}) = \emptyset$
Für den dritten und vierten SCC ist $\Delta(\mathcal{P}) = \{\leq\}$
Für den fünften SCC ist $\Delta(\mathcal{P}) = \{\mathsf{low}, \mathsf{if}_{\mathsf{low}}, \mathsf{high}, \mathsf{if}_{\mathsf{high}}, \leq\}$
Polynominterpretationen sind für alle SCCs erfolgreich