Automatische Analyse von Termination und Komplexität von (funktionalen) Programmen

Johannes Waldmann (HTWK Leipzig)

Univ. Bonn, Juni 2012

Wie teuer sind diese Funktionen?

- ```
  data List k = N | C k (List k)
  a N y = y ; a (C x y) z = C x (a y z)
  ```
- ```
  data Tree k = L | B (Tree k) k (Tree k)
  p L = N ; p (B l k r) = a (p l) (C k (p r))
  ```
- ```
  data N = O | S N
  f O = O ; f (S x) = S (S (f x))
  ```
- ```
  g O = S O ; g (S x) = f (g x)
  ```
- ```
  h (S (S x)) y = h x (S (S (S y)))
  h O (S y) = h (S y) O ; h (S O) y = O
  ```
  ```

Wie teuer sind diese Funktionen?

- ```
  data List k = N | C k (List k)
  a N y = y ; a (C x y) z = C x (a y z)
  ```

- ```
  data Tree k = L | B (Tree k) k (Tree k)
  p L = N ; p (B l k r) = a (p l) (C k (p r)
  ```

- ```
  data N = O | S N
  f O = O ; f (S x) = S (S (f x))
  ```

- ```
  g O = S O ; g (S x) = f (g x)
  ```

- ```
  h (S (S x)) y = h x (S (S (S y)))
  h O (S y) = h (S y) O ; h (S O) y = O
  ```
  ```

Wie teuer sind diese Funktionen?

- ```
  data List k = N | C k (List k)
  a N y = y ; a (C x y) z = C x (a y z)
  ```
- ```
  data Tree k = L | B (Tree k) k (Tree k)
  p L = N ; p (B l k r) = a (p l) (C k (p r)
  ```
- ```
  data N = O | S N
  f O = O ; f (S x) = S (S (f x))
  ```
- ```
  g O = S O ; g (S x) = f (g x)
  ```
- ```
  h (S (S x)) y = h x (S (S (S y)))
  h O (S y) = h (S y) O ; h (S O) y = O
  ```
  ```

Wie teuer sind diese Funktionen?

- ```
  data List k = N | C k (List k)
  a N y = y ; a (C x y) z = C x (a y z)
  ```
- ```
  data Tree k = L | B (Tree k) k (Tree k)
  p L = N ; p (B l k r) = a (p l) (C k (p r)
  ```
- ```
  data N = O | S N
  f O = O ; f (S x) = S (S (f x))
  ```
- ```
  g O = S O ; g (S x) = f (g x)
  ```
- ```
  h (S (S x)) y = h x (S (S (S y)))
  h O (S y) = h (S y) O ; h (S O) y = O
  ```
  ```

Wie teuer sind diese Funktionen?

- ```
  data List k = N | C k (List k)
  a N y = y ; a (C x y) z = C x (a y z)
  ```
- ```
  data Tree k = L | B (Tree k) k (Tree k)
  p L = N ; p (B l k r) = a (p l) (C k (p r)
  ```
- ```
  data N = O | S N
  f O = O ; f (S x) = S (S (f x))
  ```
- ```
  g O = S O ; g (S x) = f (g x)
  ```
- ```
  h (S (S x)) y = h x (S (S (S y)))
  h O (S y) = h (S y) O ; h (S O) y = O
  ```
  ```

Wie teuer ist Nicht-Termination in der Industrie?

- automatische Analyse von (third-party) (Maschinen)programmen (Gerätetreiber)
- zur Installationszeit
- ist im wirtschaftlichen Interesse des Betriebssystemanbieters
- SDV (static driver verifier), SLAM, Terminator
  `http://research.microsoft.com/en-us/um/cambridge/projects/terminator/`

Wie teuer ist Nicht-Termination in der Industrie?

- automatische Analyse von (third-party) (Maschinen)programmen (Gerätetreiber)
- zur Installationszeit
- ist im wirtschaftlichen Interesse des Betriebssystemanbieters
- SDV (static driver verifier), SLAM, Terminator
  `http://research.microsoft.com/en-us/um/cambridge/projects/terminator/`

Wie teuer ist Nicht-Termination in der Industrie?

- automatische Analyse von (third-party) (Maschinen)programmen (Gerätetreiber)
- zur Installationszeit
- ist im wirtschaftlichen Interesse des Betriebssystemanbieters
- SDV (static driver verifier), SLAM, Terminator
  `http://research.microsoft.com/en-us/um/cambridge/projects/terminator/`

Wie teuer ist Nicht-Termination in der Industrie?

- automatische Analyse von (third-party) (Maschinen)programmen (Gerätetreiber)
- zur Installationszeit
- ist im wirtschaftlichen Interesse des Betriebssystemanbieters
- SDV (static driver verifier), SLAM, Terminator
  `http://research.microsoft.com/en-us/um/cambridge/projects/terminator/`

Inhalt dieses Vortrags

Grundlagen der automatischen Analyse von Termination und Komplexität von abstrakten Programmen für Berechnungsmodelle:

- Termersetzungssystem
- funktionales Programm
- imperatives Programm = Zustandsübergangssystem

Zustandsraum ist unendlich (Daten sind beliebig groß), alle natürlichen Zahlen, alle Terme (Bäume)

Abgrenzung vom *model checking*: dort wird Zustandsraum auf endliche Menge abgebildet

Inhalt dieses Vortrags

Grundlagen der automatischen Analyse von Termination und Komplexität von abstrakten Programmen für Berechnungsmodelle:

- Termersetzungssystem
- funktionales Programm
- imperatives Programm = Zustandsübergangssystem

Zustandsraum ist unendlich (Daten sind beliebig groß), alle natürlichen Zahlen, alle Terme (Bäume)

Abgrenzung vom *model checking*: dort wird Zustandsraum auf endliche Menge abgebildet

Inhalt dieses Vortrags

Grundlagen der automatischen Analyse von Termination und Komplexität von abstrakten Programmen für Berechnungsmodelle:

- Termersetzungssystem
- funktionales Programm
- imperatives Programm = Zustandsübergangssystem

Zustandsraum ist unendlich (Daten sind beliebig groß), alle natürlichen Zahlen, alle Terme (Bäume)

Abgrenzung vom *model checking*: dort wird Zustandsraum auf endliche Menge abgebildet

Inhalt dieses Vortrags

Grundlagen der automatischen Analyse von Termination und Komplexität von abstrakten Programmen für Berechnungsmodelle:

- Termersetzungssystem
- funktionales Programm
- imperatives Programm = Zustandsübergangssystem

Zustandsraum ist unendlich (Daten sind beliebig groß), alle natürlichen Zahlen, alle Terme (Bäume)

Abgrenzung vom *model checking*: dort wird Zustandsraum auf endliche Menge abgebildet

Inhalt dieses Vortrags

Grundlagen der automatischen Analyse von Termination und Komplexität von abstrakten Programmen für Berechnungsmodelle:

- Termersetzungssystem
- funktionales Programm
- imperatives Programm = Zustandsübergangssystem

Zustandsraum ist unendlich (Daten sind beliebig groß), alle natürlichen Zahlen, alle Terme (Bäume)

Abgrenzung vom *model checking*: dort wird Zustandsraum auf endliche Menge abgebildet

Inhalt dieses Vortrags

Grundlagen der automatischen Analyse von Termination und Komplexität von abstrakten Programmen für Berechnungsmodelle:

- Termersetzungssystem
- funktionales Programm
- imperatives Programm = Zustandsübergangssystem

Zustandsraum ist unendlich (Daten sind beliebig groß), alle natürlichen Zahlen, alle Terme (Bäume)

Abgrenzung vom *model checking*: dort wird Zustandsraum auf endliche Menge abgebildet

Termersetzungssysteme (allgemein)

Berechnungsmodell

- Programm = Regelmenge, operiert auf:
- Daten = Term.

ist Turing-vollständig

ist (eine) Grundlage für funktionale Programmierung

(eine andere ist der Lambda-Kalkül)

Termersetzungssysteme (allgemein)

Berechnungsmodell

- Programm = Regelmenge, operiert auf:
- Daten = Term.

ist Turing-vollständig

ist (eine) Grundlage für funktionale Programmierung

(eine andere ist der Lambda-Kalkül)

Termersetzungssysteme (allgemein)

Berechnungsmodell

- Programm = Regelmenge, operiert auf:
- Daten = Term.

ist Turing-vollständig

ist (eine) Grundlage für funktionale Programmierung

(eine andere ist der Lambda-Kalkül)

Termersetzungssysteme (allgemein)

Berechnungsmodell

- Programm = Regelmenge, operiert auf:
- Daten = Term.

ist Turing-vollständig

ist (eine) Grundlage für funktionale Programmierung

(eine andere ist der Lambda-Kalkül)

Terme, Teilterme

- Signatur Σ:
  Menge von Funktionssymbolen mit Stelligkeiten,
  Bsp:
  $\Sigma = \{N/0, C/2, L/0, B/3, S/1, O/0, a/2, p/1\}$
- Terme über einer Signatur: $\mathrm{Term}(\Sigma)$
  Bsp: $p(B(L, S(O), B(L, O, L)))$
- Position (Adresse eines Teilterms): $\mathbb{N}^*$ Bsp:
  $[0, 1]$
- Menge aller Positionen eines Terms: $\mathrm{Pos}(t)$
- Teilterm an Position: $t[p]$
- Teilterm an Position p ersetzen: $t[p := t']$

Terme, Teilterme

- Signatur Σ:
  Menge von Funktionssymbolen mit Stelligkeiten,
  Bsp:
  $\Sigma = \{N/0, C/2, L/0, B/3, S/1, O/0, a/2, p/1\}$
- Terme über einer Signatur: $\mathrm{Term}(\Sigma)$
  Bsp: $p(B(L, S(O), B(L, O, L)))$
- Position (Adresse eines Teilterms): $\mathbb{N}^*$ Bsp: $[0, 1]$
- Menge aller Positionen eines Terms: $\mathrm{Pos}(t)$
- Teilterm an Position: $t[p]$
- Teilterm an Position p ersetzen: $t[p := t']$

Terme, Teilterme

- Signatur Σ:
  Menge von Funktionssymbolen mit Stelligkeiten,
  Bsp:
  $\Sigma = \{N/0, C/2, L/0, B/3, S/1, O/0, a/2, p/1\}$
- Terme über einer Signatur: $\mathrm{Term}(\Sigma)$
  Bsp: $p(B(L, S(O), B(L, O, L)))$
- Position (Adresse eines Teilterms): $\mathbb{N}^*$ Bsp:
  $[0, 1]$
- Menge aller Positionen eines Terms: $\mathrm{Pos}(t)$
- Teilterm an Position: $t[p]$
- Teilterm an Position p ersetzen: $t[p := t']$

Terme, Teilterme

- Signatur Σ:
  Menge von Funktionssymbolen mit Stelligkeiten,
  Bsp:
  $\Sigma = \{N/0, C/2, L/0, B/3, S/1, O/0, a/2, p/1\}$
- Terme über einer Signatur: $\mathrm{Term}(\Sigma)$
  Bsp: $p(B(L, S(O), B(L, O, L)))$
- Position (Adresse eines Teilterms): $\mathbb{N}^*$ Bsp:
  $[0, 1]$
- Menge aller Positionen eines Terms: $\mathrm{Pos}(t)$
- Teilterm an Position: $t[p]$
- Teilterm an Position p ersetzen: $t[p := t']$

Terme, Teilterme

- Signatur Σ:
  Menge von Funktionssymbolen mit Stelligkeiten,
  Bsp:
  $\Sigma = \{N/0, C/2, L/0, B/3, S/1, O/0, a/2, p/1\}$
- Terme über einer Signatur: $\mathrm{Term}(\Sigma)$
  Bsp: $p(B(L, S(O), B(L, O, L)))$
- Position (Adresse eines Teilterms): $\mathbb{N}^*$ Bsp:
  $[0, 1]$
- Menge aller Positionen eines Terms: $\mathrm{Pos}(t)$
- Teilterm an Position: $t[p]$
- Teilterm an Position p ersetzen: $t[p := t']$

Terme, Teilterme

- Signatur Σ:
  Menge von Funktionssymbolen mit Stelligkeiten,
  Bsp:
  $\Sigma = \{N/0, C/2, L/0, B/3, S/1, O/0, a/2, p/1\}$
- Terme über einer Signatur: $\mathrm{Term}(\Sigma)$
  Bsp: $p(B(L, S(O), B(L, O, L)))$
- Position (Adresse eines Teilterms): $\mathbb{N}^*$ Bsp:
  $[0, 1]$
- Menge aller Positionen eines Terms: $\mathrm{Pos}(t)$
- Teilterm an Position: $t[p]$
- Teilterm an Position p ersetzen: $t[p := t']$

Variablen, Substitutionen

- Terme mit Variablen: $\mathsf{Term}(\Sigma, V)$
  Bsp: $p(B(l, k, r)) \in \mathsf{Term}(\Sigma, \{l, k, r\})$
- Substitution: $\sigma : V \to \mathsf{Term}(\Sigma)$
  Bsp: $\sigma = \{l \mapsto L, k \mapsto S(O), r \mapsto B(L, O, L)\}$
- Substitution anwenden auf Term: $t\sigma$
  Bsp: $(p(B(l, k, r)))\sigma$,
  Bsp: $(a(p(l), C(k, (p(r)))))\sigma$

Variablen, Substitutionen

- Terme mit Variablen: $\mathsf{Term}(\Sigma, V)$
  Bsp: $p(B(l, k, r)) \in \mathsf{Term}(\Sigma, \{l, k, r\})$
- Substitution: $\sigma : V \to \mathsf{Term}(\Sigma)$
  Bsp: $\sigma = \{l \mapsto L, k \mapsto S(O), r \mapsto B(L, O, L)\}$
- Substitution anwenden auf Term: $t\sigma$
  Bsp: $(p(B(l, k, r)))\sigma$,
  Bsp: $(a(p(l), C(k, (p(r)))))\sigma$

Variablen, Substitutionen

- Terme mit Variablen: $\mathrm{Term}(\Sigma, V)$
  Bsp: $p(B(l, k, r)) \in \mathrm{Term}(\Sigma, \{l, k, r\})$
- Substitution: $\sigma : V \to \mathrm{Term}(\Sigma)$
  Bsp: $\sigma = \{l \mapsto L, k \mapsto S(O), r \mapsto B(L, O, L)\}$
- Substitution anwenden auf Term: $t\sigma$
  Bsp: $(p(B(l, k, r)))\sigma$,
  Bsp: $(a(p(l), C(k, (p(r)))))\sigma$

Ersetzungsregeln, Ersetzungssysteme

- Regelmenge $R \subseteq \mathrm{Term}(\Sigma, V)^2$
  Bsp: $\{p(L) \to N, p(B(l, k, r)) \to a(p(l), C(k, p(r)))\}$
- eine Regel an einer Position mit einer Substitution anwenden: $t \to_{(l,r),p,\sigma} t'$
  falls $t[p] = l\sigma \wedge t' = t[p := r\sigma]$
- ein Ersetzungs-Schritt: $t \to_R t'$
  falls $\exists (l, r) \in R : \exists p \in \mathrm{Pos}(t) : \exists \sigma \in \mathrm{Var}(l) \to \mathrm{Term}(\Sigma) : t \to_{(l,r),p,\sigma} t'$
  Bsp: $p(B(L, S(O), B(L, O, L))) \to_R a(p(L), C(S(O), p(B(L, O, L))))$

Ersetzungsregeln, Ersetzungssysteme

- Regelmenge $R \subseteq \text{Term}(\Sigma, V)^2$
  Bsp: $\{p(L) \to N, p(B(l, k, r)) \to a(p(l), C(k, p(r)))\}$
- eine Regel an einer Position mit einer Substitution anwenden: $t \to_{(l,r),p,\sigma} t'$
  falls $t[p] = l\sigma \wedge t' = t[p := r\sigma]$
- ein Ersetzungs-Schritt: $t \to_R t'$
  falls $\exists (l, r) \in R : \exists p \in \text{Pos}(t) : \exists \sigma \in \text{Var}(l) \to \text{Term}(\Sigma) : t \to_{(l,r),p,\sigma} t'$
  Bsp: $p(B(L, S(O), B(L, O, L))) \to_R a(p(L), C(S(O), p(B(L, O, L))))$

Ersetzungsregeln, Ersetzungssysteme

- Regelmenge $R \subseteq \text{Term}(\Sigma, V)^2$
  Bsp: $\{p(L) \to N, p(B(l, k, r)) \to a(p(l), C(k, p(r)))\}$
- eine Regel an einer Position mit einer Substitution anwenden: $t \to_{(l,r),p,\sigma} t'$
  falls $t[p] = l\sigma \wedge t' = t[p := r\sigma]$
- ein Ersetzungs-Schritt: $t \to_R t'$
  falls $\exists (l, r) \in R : \exists p \in \text{Pos}(t) : \exists \sigma \in \text{Var}(l) \to \text{Term}(\Sigma) : t \to_{(l,r),p,\sigma} t'$
  Bsp: $p(B(L, S(O), B(L, O, L))) \to_R a(p(L), C(S(O), p(B(L, O, L))))$

Monotone Algebren

Σ-Algebra A : Universum U und für jedes k-stellige Symbol $f \in \Sigma$ eine Funktion $[f]_A \in U^k \to U$.

Bsp: $\Sigma = \{S/1, f/1, O/0\}$, $U = \mathbb{N}$,
$[S] = (u \mapsto u + 1), [f] = (u \mapsto 3u), [O] = 0$

monotone Algebra: Halbordnung $(U, >)$ und jede Funktion ist in jedem Argument streng monoton:

$$x > y \Rightarrow g(\ldots, x, \ldots) > g(\ldots, y, \ldots)$$

es folgt Monotonie bzgl. Einsetzen:

$$p \in \mathrm{Pos}(t), [s_1]_A > [s_2]_A \Rightarrow [t[p := s_1]]_A > [t[p := s_2]]_A$$

Monotone Algebren

Σ-Algebra A : Universum U und für jedes k-stellige Symbol $f \in \Sigma$ eine Funktion $[f]_A \in U^k \to U$.

Bsp: $\Sigma = \{S/1, f/1, O/0\}$, $U = \mathbb{N}$,
$[S] = (u \mapsto u + 1), [f] = (u \mapsto 3u), [O] = 0$

monotone Algebra: Halbordnung $(U, >)$ und jede Funktion ist in jedem Argument streng monoton:

$$x > y \Rightarrow g(\ldots, x, \ldots) > g(\ldots, y, \ldots)$$

es folgt Monotonie bzgl. Einsetzen:

$$p \in \mathrm{Pos}(t), [s_1]_A > [s_2]_A \Rightarrow [t[p := s_1]]_A > [t[p := s_2]]_A$$

Monotone Algebren

Σ-Algebra A : Universum U und für jedes k-stellige Symbol $f \in \Sigma$ eine Funktion $[f]_A \in U^k \to U$.

Bsp: $\Sigma = \{S/1, f/1, O/0\}$, $U = \mathbb{N}$,
$[S] = (u \mapsto u + 1), [f] = (u \mapsto 3u), [O] = 0$

monotone Algebra: Halbordnung $(U, >)$ und jede Funktion ist in jedem Argument streng monoton:

$$x > y \Rightarrow g(\ldots, x, \ldots) > g(\ldots, y, \ldots)$$

es folgt Monotonie bzgl. Einsetzen:

$$p \in \mathrm{Pos}(t), [s_1]_A > [s_2]_A \\ \Rightarrow [t[p := s_1]]_A > [t[p := s_2]]_A$$

Kombatibilität, Termination

A ist *kompatibel* mit Regel (l, r): $\forall \sigma : [l\sigma]_A > [r\sigma]_A$

Bsp: $f(S(x)) \to S(S(f(x)))$

$\forall u : [f][S](u) > [S][S][f](u)$

A monoton und kompatibel mit jeder Regel aus R

$\Rightarrow \forall s, t : s \to_R t \Rightarrow [s]_A > [t]_A$

wenn $(U, >)$ *wohlfundiert* ist (jede absteigende Kette ist endlich, Bsp: $(\mathbb{N}, >)$), folgt daraus Termination von R. (Manna, Ness, 1970)

`http://perso.ens-lyon.fr/pierre.lescanne/not_accessible.html`

Kombatibilität, Termination

A ist *kompatibel* mit Regel (l, r): $\forall \sigma : [l\sigma]_A > [r\sigma]_A$

Bsp: $f(S(x)) \to S(S(f(x)))$

$\forall u : [f][S](u) > [S][S][f](u)$

A monoton und kompatibel mit jeder Regel aus R

$\Rightarrow \forall s, t : s \to_R t \Rightarrow [s]_A > [t]_A$

wenn $(U, >)$ *wohlfundiert* ist (jede absteigende Kette ist endlich, Bsp: $(\mathbb{N}, >)$), folgt daraus Termination von R. (Manna, Ness, 1970)

http://perso.ens-lyon.fr/pierre.lescanne/not_accessible.html

Kombatibilität, Termination

A ist *kompatibel* mit Regel (l, r): $\forall \sigma : [l\sigma]_A > [r\sigma]_A$

Bsp: $f(S(x)) \to S(S(f(x)))$

$\forall u : [f][S](u) > [S][S][f](u)$

A monoton und kompatibel mit jeder Regel aus R

$\Rightarrow \forall s, t : s \to_R t \Rightarrow [s]_A > [t]_A$

wenn $(U, >)$ *wohlfundiert* ist (jede absteigende Kette ist endlich, Bsp: $(\mathbb{N}, >)$), folgt daraus Termination von R. (Manna, Ness, 1970)

`http://perso.ens-lyon.fr/pierre.lescanne/not_accessible.html`

Interpretationen finden

durch symbolisches Rechnen in einer Klasse von Interpretationen mit unbekannten Koeffizienten

Bsp: lineare Funktionen $f(x) = ax + b$ auf $\mathbb{N}$

Monotonie und Kompatibilität mit Regelsystem ergeben Constraint-System für Koeffizienten.

Monotonie: $a > 0$. Bereich $f : \mathbb{N} \to \mathbb{N}$: $b \geq 0$.

Kompatibel mit $fSx \to SSfx$:

$a_f a_S \geq a_s a_s a_f \wedge b_f + a_f b_S > b_S + a_S b_S + a_S a_S b_f$

Lösung: $a_s = 1, a_f > 2, b_s > 0$

Interpretationen finden

durch symbolisches Rechnen in einer Klasse von Interpretationen mit unbekannten Koeffizienten

Bsp: lineare Funktionen $f(x) = ax + b$ auf $\mathbb{N}$

Monotonie und Kompatibilität mit Regelsystem ergeben Constraint-System für Koeffizienten.

Monotonie: $a > 0$. Bereich $f : \mathbb{N} \to \mathbb{N}$: $b \geq 0$.

Kompatibel mit $fSx \to SSfx$:

$a_f a_S \geq a_s a_s a_f \wedge b_f + a_f b_S > b_S + a_S b_S + a_S a_S b_f$

Lösung: $a_s = 1, a_f > 2, b_s > 0$

Interpretationen finden

durch symbolisches Rechnen in einer Klasse von Interpretationen mit unbekannten Koeffizienten

Bsp: lineare Funktionen $f(x) = ax + b$ auf $\mathbb{N}$

Monotonie und Kompatibilität mit Regelsystem ergeben Constraint-System für Koeffizienten.

Monotonie: $a > 0$. Bereich $f : \mathbb{N} \to \mathbb{N}$: $b \geq 0$.

Kompatibel mit $fSx \to SSfx$:

$a_f a_S \geq a_s a_s a_f \wedge b_f + a_f b_S > b_S + a_S b_S + a_S a_S b_f$

Lösung: $a_s = 1, a_f > 2, b_s > 0$

Interpretationen finden

durch symbolisches Rechnen in einer Klasse von Interpretationen mit unbekannten Koeffizienten

Bsp: lineare Funktionen $f(x) = ax + b$ auf $\mathbb{N}$

Monotonie und Kompatibilität mit Regelsystem ergeben Constraint-System für Koeffizienten.

Monotonie: $a > 0$. Bereich $f : \mathbb{N} \to \mathbb{N}$: $b \geq 0$.

Kompatibel mit $fSx \to SSfx$:

$$a_f a_S \geq a_s a_s a_f \wedge b_f + a_f b_S > b_S + a_S b_S + a_S a_S b_f$$

Lösung: $a_s = 1, a_f > 2, b_s > 0$

Interpretationen finden

durch symbolisches Rechnen in einer Klasse von Interpretationen mit unbekannten Koeffizienten

Bsp: lineare Funktionen $f(x) = ax + b$ auf $\mathbb{N}$

Monotonie und Kompatibilität mit Regelsystem ergeben Constraint-System für Koeffizienten.

Monotonie: $a > 0$. Bereich $f : \mathbb{N} \to \mathbb{N}$: $b \geq 0$.

Kompatibel mit $fSx \to SSfx$:

$a_f a_S \geq a_s a_s a_f \wedge b_f + a_f b_S > b_S + a_S b_S + a_S a_S b_f$

Lösung: $a_s = 1, a_f > 2, b_s > 0$

Interpretationen finden

durch symbolisches Rechnen in einer Klasse von Interpretationen mit unbekannten Koeffizienten

Bsp: lineare Funktionen $f(x) = ax + b$ auf $\mathbb{N}$

Monotonie und Kompatibilität mit Regelsystem ergeben Constraint-System für Koeffizienten.

Monotonie: $a > 0$. Bereich $f : \mathbb{N} \to \mathbb{N}$: $b \geq 0$.

Kompatibel mit $fSx \to SSfx$:

$a_f a_S \geq a_s a_s a_f \wedge b_f + a_f b_S > b_S + a_S b_S + a_S a_S b_f$

Lösung: $a_s = 1, a_f > 2, b_s > 0$

Implementierung der Lösungssuche

Constraint-System f. Koeffizienten ist i. A. nichtlineares (polynomielles) Ungleichungssystem.

dann kann man auch lineare $\rightarrow$ polynomielle Interpretation verallgemeinern (Charifa und Lescanne, 1987, `http://perso.ens-lyon.fr/pierre.lescanne/publications.html`

Lösbarkeit ist unentscheidbar (über $\mathbb{N}$, Hilberts 10. Problem) oder sehr komplex (über $\mathbb{R}$, QEPCAD).

Lösung in kleinen ganzen Zahlen durch SAT-Kodierung und SAT-Solver.

Implementierung der Lösungssuche

Constraint-System f. Koeffizienten ist i. A. nichtlineares (polynomielles) Ungleichungssystem.

dann kann man auch lineare $\rightarrow$ polynomielle Interpretation verallgemeinern (Charifa und Lescanne, 1987, `http://perso.ens-lyon.fr/pierre.lescanne/publications.html`

Lösbarkeit ist unentscheidbar (über $\mathbb{N}$, Hilberts 10. Problem) oder sehr komplex (über $\mathbb{R}$, QEPCAD).

Lösung in kleinen ganzen Zahlen durch SAT-Kodierung und SAT-Solver.

Implementierung der Lösungssuche

Constraint-System f. Koeffizienten ist i. A. nichtlineares (polynomielles) Ungleichungssystem.

dann kann man auch lineare $\rightarrow$ polynomielle Interpretation verallgemeinern (Charifa und Lescanne, 1987, `http://perso.ens-lyon.fr/pierre.lescanne/publications.html`

Lösbarkeit ist unentscheidbar (über $\mathbb{N}$, Hilberts 10. Problem) oder sehr komplex (über $\mathbb{R}$, QEPCAD).

Lösung in kleinen ganzen Zahlen durch SAT-Kodierung und SAT-Solver.

Implementierung der Lösungssuche

Constraint-System f. Koeffizienten ist i. A. nichtlineares (polynomielles) Ungleichungssystem.

dann kann man auch lineare $\rightarrow$ polynomielle Interpretation verallgemeinern (Charifa und Lescanne, 1987, `http://perso.ens-lyon.fr/pierre.lescanne/publications.html`

Lösbarkeit ist unentscheidbar (über $\mathbb{N}$, Hilberts 10. Problem) oder sehr komplex (über $\mathbb{R}$, QEPCAD).

Lösung in kleinen ganzen Zahlen durch SAT-Kodierung und SAT-Solver.

Grenzen des Verfahrens

System $R = \{a(a(x)) \to a(b(a(x)))\}$

- terminiert (betrachte Anzahl der a-Paare)
- Ableitungskomplexität ist nur linear
- besitzt keine kompatible lineare Interpretation (wegen $[b](x) \geq x$)
- Ausweg: a-Paare zählen durch lineare Funktionen auf Vektoren: $[a](x, y) = (x + y, 1), [b](x, y) = (x, 0)$
- Ordnung? Monotonie? Kompatibilität?

Grenzen des Verfahrens

System $R = \{a(a(x)) \to a(b(a(x)))\}$

- terminiert (betrachte Anzahl der a-Paare)
- Ableitungskomplexität ist nur linear
- besitzt keine kompatible lineare Interpretation (wegen $[b](x) \geq x$)
- Ausweg: a-Paare zählen durch lineare Funktionen auf Vektoren: $[a](x, y) = (x + y, 1), [b](x, y) = (x, 0)$
- Ordnung? Monotonie? Kompatibilität?

Grenzen des Verfahrens

System $R = \{a(a(x)) \to a(b(a(x)))\}$

- terminiert (betrachte Anzahl der a-Paare)
- Ableitungskomplexität ist nur linear
- besitzt keine kompatible lineare Interpretation (wegen $[b](x) \geq x$)
- Ausweg: a-Paare zählen durch lineare Funktionen auf Vektoren: $[a](x, y) = (x + y, 1), [b](x, y) = (x, 0)$
- Ordnung? Monotonie? Kompatibilität?

Grenzen des Verfahrens

System $R = \{a(a(x)) \to a(b(a(x)))\}$

- terminiert (betrachte Anzahl der a-Paare)
- Ableitungskomplexität ist nur linear
- besitzt keine kompatible lineare Interpretation (wegen $[b](x) \geq x$)
- Ausweg: a-Paare zählen durch lineare Funktionen auf Vektoren: $[a](x, y) = (x + y, 1), [b](x, y) = (x, 0)$
- Ordnung? Monotonie? Kompatibilität?

Grenzen des Verfahrens

System $R = \{a(a(x)) \to a(b(a(x)))\}$

- terminiert (betrachte Anzahl der a-Paare)
- Ableitungskomplexität ist nur linear
- besitzt keine kompatible lineare Interpretation (wegen $[b](x) \geq x$)
- Ausweg: a-Paare zählen durch lineare Funktionen auf Vektoren: $[a](x, y) = (x + y, 1), [b](x, y) = (x, 0)$
- Ordnung? Monotonie? Kompatibilität?

Grenzen des Verfahrens

System $R = \{a(a(x)) \to a(b(a(x)))\}$

- terminiert (betrachte Anzahl der a-Paare)
- Ableitungskomplexität ist nur linear
- besitzt keine kompatible lineare Interpretation (wegen $[b](x) \geq x$)
- Ausweg: a-Paare zählen durch lineare Funktionen auf Vektoren:
  $[a](x, y) = (x + y, 1), [b](x, y) = (x, 0)$
- Ordnung? Monotonie? Kompatibilität?

Grenzen des Verfahrens

System $R = \{a(a(x)) \to a(b(a(x)))\}$

- terminiert (betrachte Anzahl der a-Paare)
- Ableitungskomplexität ist nur linear
- besitzt keine kompatible lineare Interpretation (wegen $[b](x) \geq x$)
- Ausweg: a-Paare zählen durch lineare Funktionen auf Vektoren:
  $[a](x, y) = (x + y, 1), [b](x, y) = (x, 0)$
- Ordnung? Monotonie? Kompatibilität?

Vektorwertige Interpretationen

$$U = \mathbb{N}^d,\ [f] = (x_1, \ldots, x_k) \mapsto F_0 + F_1 x_1 + \ldots + F_i x_k$$

$$\text{Bsp } [a](x) = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \cdot x + \begin{pmatrix} 0 \\ 1 \end{pmatrix}, [b](x) = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \cdot x$$

wohlfundierte Ordnung auf U:

$$(u_1, \ldots, u_k)^T > (v_1, \ldots, v_k)^T :=$$
$$u_1 > v_1 \wedge u_2 \geq v_1 \wedge \ldots \wedge u_k \geq v_k$$

Monotonie: $(F_*)_{*,*} \geq 0, (F_{>0})_{1,1} \geq 1$

Kompatibilität durch Koeffizientenvergleich

Hofbauer, Waldmann, Endrullis, Zantema 2007

Vektorwertige Interpretationen

$U = \mathbb{N}^d$, $[f] = (x_1, \ldots, x_k) \mapsto F_0 + F_1 x_1 + \ldots + F_i x_k$

Bsp $[a](x) = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \cdot x + \begin{pmatrix} 0 \\ 1 \end{pmatrix}, [b](x) = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \cdot x$

wohlfundierte Ordnung auf U:

$(u_1, \ldots, u_k)^T > (v_1, \ldots, v_k)^T :=$
$u_1 > v_1 \wedge u_2 \geq v_1 \wedge \ldots \wedge u_k \geq v_k$

Monotonie: $(F_*)_{*,*} \geq 0, (F_{>0})_{1,1} \geq 1$

Kompatibilität durch Koeffizientenvergleich

Hofbauer, Waldmann, Endrullis, Zantema 2007

Vektorwertige Interpretationen

$U = \mathbb{N}^d$, $[f] = (x_1, \ldots, x_k) \mapsto F_0 + F_1 x_1 + \ldots + F_i x_k$

Bsp $[a](x) = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \cdot x + \begin{pmatrix} 0 \\ 1 \end{pmatrix}, [b](x) = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \cdot x$

wohlfundierte Ordnung auf U:

$(u_1, \ldots, u_k)^T > (v_1, \ldots, v_k)^T :=$
$u_1 > v_1 \wedge u_2 \geq v_1 \wedge \ldots \wedge u_k \geq v_k$

Monotonie: $(F_*)_{*,*} \geq 0, (F_{>0})_{1,1} \geq 1$

Kompatibilität durch Koeffizientenvergleich

Hofbauer, Waldmann, Endrullis, Zantema 2007

Vektorwertige Interpretationen

$U = \mathbb{N}^d$, $[f] = (x_1, \ldots, x_k) \mapsto F_0 + F_1 x_1 + \ldots + F_i x_k$

Bsp $[a](x) = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \cdot x + \begin{pmatrix} 0 \\ 1 \end{pmatrix}, [b](x) = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \cdot x$

wohlfundierte Ordnung auf U:

$(u_1, \ldots, u_k)^T > (v_1, \ldots, v_k)^T :=$
$u_1 > v_1 \wedge u_2 \geq v_1 \wedge \ldots \wedge u_k \geq v_k$

Monotonie: $(F_*)_{*,*} \geq 0, (F_{>0})_{1,1} \geq 1$

Kompatibilität durch Koeffizientenvergleich

Hofbauer, Waldmann, Endrullis, Zantema 2007

Ableitungslängen

Ableitungshöhe eines Terms t bzgl. Programm R

$\mathrm{dh}_R(t) = \sup\{k \mid \exists t' : t \to_R^k t'\}$

Größe eines Terms: $|t| = |\mathrm{Pos}(t)|$

Ableitungskomplexität = Ableitungshöhe bzgl. Eingabegröße

$\mathrm{dc}_R(s) = \sup\{\mathrm{dh}_R(t) \mid t \in \mathrm{Term}(\Sigma), |t| \leq s\}$

linear, polynomiell, höher?

Ableitungslängen

Ableitungshöhe eines Terms t bzgl. Programm R

$\mathrm{dh}_R(t) = \sup\{k \mid \exists t' : t \to_R^k t'\}$

Größe eines Terms: $|t| = |\mathrm{Pos}(t)|$

Ableitungskomplexität = Ableitungshöhe bzgl. Eingabegröße

$\mathrm{dc}_R(s) = \sup\{\mathrm{dh}_R(t) \mid t \in \mathrm{Term}(\Sigma), |t| \leq s\}$

linear, polynomiell, höher?

Ableitungslängen

Ableitungshöhe eines Terms t bzgl. Programm R

$\mathrm{dh}_R(t) = \sup\{k \mid \exists t' : t \to_R^k t'\}$

Größe eines Terms: $|t| = |\mathrm{Pos}(t)|$

Ableitungskomplexität = Ableitungshöhe bzgl. Eingabegröße

$\mathrm{dc}_R(s) = \sup\{\mathrm{dh}_R(t) \mid t \in \mathrm{Term}(\Sigma), |t| \leq s\}$

linear, polynomiell, höher?

Ableitungskomplexität durch Interpretationen

Wenn A eine monotone Algebra über wohlfundiertem $(U, >)$, kompatibel zu System R,

dann $\mathrm{dh}_R(t) \leq \mathrm{dh}_>([t]_A)$

Schranke für $\mathrm{dh}_>([t])$ in Abhängigkeit von $|t|$ liefert Schranke für dc_R.

Z. B. Funktionen der Form $(u \mapsto 1 \cdot u + b)$ liefern lineare Schranke. (Form $(u \mapsto a \cdot u + b)$ liefert. . . ?)

polynomiell beschränkte Matrixinterpretationen Hofbauer, Moser, Waldmann, 2008

Ableitungskomplexität durch Interpretationen

Wenn A eine monotone Algebra über wohlfundiertem $(U, >)$, kompatibel zu System R,

dann $\mathrm{dh}_R(t) \leq \mathrm{dh}_>([t]_A)$

Schranke für $\mathrm{dh}_>([t])$ in Abhängigkeit von $|t|$ liefert Schranke für dc_R.

Z. B. Funktionen der Form $(u \mapsto 1 \cdot u + b)$ liefern lineare Schranke. (Form $(u \mapsto a \cdot u + b)$ liefert. . . ?)

polynomiell beschränkte Matrixinterpretationen Hofbauer, Moser, Waldmann, 2008

Ableitungskomplexität durch Interpretationen

Wenn A eine monotone Algebra über wohlfundiertem $(U, >)$, kompatibel zu System R,

dann $\mathrm{dh}_R(t) \leq \mathrm{dh}_>([t]_A)$

Schranke für $\mathrm{dh}_>([t])$ in Abhängigkeit von $|t|$ liefert Schranke für dc_R.

Z. B. Funktionen der Form $(u \mapsto 1 \cdot u + b)$ liefern lineare Schranke. (Form $(u \mapsto a \cdot u + b)$ liefert...?)

polynomiell beschränkte Matrixinterpretationen
Hofbauer, Moser, Waldmann, 2008

Ableitungskomplexität durch Interpretationen

Wenn A eine monotone Algebra über wohlfundiertem $(U, >)$, kompatibel zu System R,

dann $\mathrm{dh}_R(t) \leq \mathrm{dh}_>([t]_A)$

Schranke für $\mathrm{dh}_>([t])$ in Abhängigkeit von $|t|$ liefert Schranke für dc_R.

Z. B. Funktionen der Form $(u \mapsto 1 \cdot u + b)$ liefern lineare Schranke. (Form $(u \mapsto a \cdot u + b)$ liefert. . . ?)

polynomiell beschränkte Matrixinterpretationen Hofbauer, Moser, Waldmann, 2008

Analyse von Programmen

Übersetzung des Programms P (Haskell, Prolog, Java-Bytecode) in Termersetungssystem R,

dann Analyse von R mit hier beschriebenen (und weiteren) Methoden.

Bsp (analysiert Termination):

AProVE (automated programm verification environment) Giesl et al, 2004, `http://aprove.informatik.rwth-aachen.de/`

Resource Aware ML (RAML)

Hoffmann, Hofman, 2010

- first-order, monomorphe funktionale Sprache
- Beschreibung des Resourcenverbrauchs (Zeit, Platz) durch Differenz der Interpretationen von Argument und Resultat
- automatische Bestimmung der Koeffizienten in der Interpretation durch Constraint-Solver (CLP)
- Erweiterung auf higher-order, Polymorphie: ongoing work (Bau, H., H., Waldmann, 2012)

`http://raml.tcs.ifi.lmu.de/`

Termination/Complexity Competition

seit 2003 (Termination), 2008 (Complexity)

- Eingabe:
  TPDB: Termination Problem Data Base
  (TRS, SRS, Prolog, Haskell, Java Bytecode)
- Ausgabe:
  Beweis für Termination oder Nichttermination,
  für polynomielle Komplexität

2. Phase: maschinelle Verifikation des Beweise

`http://www.termination-portal.org/wiki/Termination_Competition`

vgl. Competitions: SAT, SMT, Theorembeweiser, ...

Termination/Complexity Competition

seit 2003 (Termination), 2008 (Complexity)

- Eingabe:
  TPDB: Termination Problem Data Base
  (TRS, SRS, Prolog, Haskell, Java Bytecode)
- Ausgabe:
  Beweis für Termination oder Nichttermination,
  für polynomielle Komplexität

2. Phase: maschinelle Verifikation des Beweise

`http://www.termination-portal.org/wiki/Termination_Competition`

vgl. Competitions: SAT, SMT, Theorembeweiser, ...

Alternativen

Termination und Komplexität als Typ formulieren.

- Typinferenz (d. h. Compiler beweist Termination, bestimmt Komplexität)
  Bsp: dieser Vortrag (für TRS), RAML
- Typdeklaration (d. h. Programmierer schreibt Terminations/Komplexitätsbeweis auf, Compiler prüft), Bsp: totale Funktionen in Coq, Agda

Implicit Computational Complexity: syntaktische Einschränkungen

- Termination: Bsp: keine Rekursion, nur fold
- polynomielle Komplexität: nur eingeschränkte Rekursion

Alternativen

Termination und Komplexität als Typ formulieren.

- Typinferenz (d. h. Compiler beweist Termination, bestimmt Komplexität)
  Bsp: dieser Vortrag (für TRS), RAML
- Typdeklaration (d. h. Programmierer schreibt Terminations/Komplexitätsbeweis auf, Compiler prüft), Bsp: totale Funktionen in Coq, Agda

Implicit Computational Complexity: syntaktische Einschränkungen

- Termination: Bsp: keine Rekursion, nur fold
- polynomielle Komplexität: nur eingeschränkte Rekursion

Alternativen

Termination und Komplexität als Typ formulieren.

- Typinferenz (d. h. Compiler beweist Termination, bestimmt Komplexität)
  Bsp: dieser Vortrag (für TRS), RAML
- Typdeklaration (d. h. Programmierer schreibt Terminations/Komplexitätsbeweis auf, Compiler prüft), Bsp: totale Funktionen in Coq, Agda

Implicit Computational Complexity: syntaktische Einschränkungen

- Termination: Bsp: keine Rekursion, nur fold
- polynomielle Komplexität: nur eingeschränkte Rekursion

Alternativen

Termination und Komplexität als Typ formulieren.

- Typinferenz (d. h. Compiler beweist Termination, bestimmt Komplexität)
  Bsp: dieser Vortrag (für TRS), RAML
- Typdeklaration (d. h. Programmierer schreibt Terminations/Komplexitätsbeweis auf, Compiler prüft), Bsp: totale Funktionen in Coq, Agda

Implicit Computational Complexity: syntaktische Einschränkungen

- Termination: Bsp: keine Rekursion, nur fold
- polynomielle Komplexität: nur eingeschränkte Rekursion

Alternativen

Termination und Komplexität als Typ formulieren.

- Typinferenz (d. h. Compiler beweist Termination, bestimmt Komplexität)
  Bsp: dieser Vortrag (für TRS), RAML
- Typdeklaration (d. h. Programmierer schreibt Terminations/Komplexitätsbeweis auf, Compiler prüft), Bsp: totale Funktionen in Coq, Agda

Implicit Computational Complexity: syntaktische Einschränkungen

- Termination: Bsp: keine Rekursion, nur fold
- polynomielle Komplexität: nur eingeschränkte Rekursion