

14. Entscheidungsprozeduren

Entscheidungsprozedur: Prozedur zur Lösung des Entscheidungsproblem in einer (entscheidbaren) Theorie.

Warum Entscheidungsprozeduren?

- Deduktionen in häufig benutzten, entscheidbaren Theorien werden automatisch ausgeführt –
Entscheidungsprozeduren als “*black boxes*”
- Lästige, langwierige und meistens uninteressante Routine-Manipulationen werden von der Maschine übernommen; der Benutzer kann sich auf die wesentlich(er)en Aspekte eines Beweises konzentrieren.

Entscheidungsprozeduren sind ein Bestandteil praktisch aller ernst zu nehmender Beweissysteme.

Entscheidungsprozeduren (2)

Typische entscheidbare Theorien:

- boolesche Ausdrücke (Aussagenlogik)
- Lineare Arithmetik (Presburger-Arithmetik) mit Ordnungsrelation
- *Allgemeine* Gleichungstheorie: Gleichungen mit uninterpretierten Funktionssymbolen
(im Gegensatz zu durch Gleichungen spezifizierte Theorien)
- einfache Datenstrukturen (bzw. bestimmte Teiltheorien dazu): Arrays, Records, einfache induktive Datenstrukturen mit Konstruktoren und Selektoren (Listen, Bäume, . . .).

Entscheidungsprozeduren (3)

Probleme und Beschänkungen:

- Viele für Anwendungen interessante Theorien sind nicht entscheidbar.
Beispiel: nicht-lineare Arithmetik
Partielle Lösung: Entwicklung von *unvollständigen* Verfahren, die einfachere Fälle erkennen und vereinfachen.
- Die meisten Entscheidungsprobleme sind inhärent komplex, d.h. effiziente Entscheidungsprozeduren kann es i.a. nicht geben.
~> Unterschiedliche Realisierungen von Entscheidungsprozeduren, die jeweils für eingeschränkte Anwendungsbereiche hinreichend effizient sind
(Beispiel: Aussagenlogik).
- Entscheidungsprobleme werden theoretisch meist in Isolation untersucht, aber Entscheidungsprozeduren werden i.a. zusammen mit anderen Beweisverfahren (und anderen Entscheidungsprozeduren) eingesetzt und müssen mit diesen harmonieren.
~> Problem der Kombination von Entscheidungsprozeduren und deren Integration in Beweissysteme.

Entscheidungsprozeduren (4)

Implementierung: z.B. durch

- kanonische Termersetzungssysteme
- spezielle Algorithmen

Z.B. Array-Theorie:

Sorten:

I – Index-Typ

E – Element-Typ

Arr – Arrays über I und E

Signatur:

$update : Arr \times I \times E \rightarrow Arr$

$select : Arr \times I \rightarrow E$

Axiome:

$select(update(A, i, e), j) = [\mathbf{if } i=j \mathbf{ then } e \mathbf{ else } select(A, j)]$

Kongruenz-Abschluß

(engl. *congruence closure*)

Ausgangspunkt: gerichteter markierter Graph mit markierten Kanten; zwischen zwei Knoten können mehrere Kanten existieren.

Für eine Knotenmenge V und $v \in V$ bezeichne

$l(v)$ die Marke von v ,

$\delta(v)$ die Anzahl der Nachfolger von v ,

$v[i]$ den i -ten Nachfolger-Knoten von v ($1 \leq i \leq \delta(v)$).

R sei eine Relation auf V .

Knoten $u, v \in V$ heißen *kongruent unter R* , falls sie gleich markiert sind, die gleiche Anzahl von Nachfolgern haben, und die jeweils entsprechenden Nachfolger in der Relation R stehen:

$$u \approx_R v \iff l(u) = l(v) \wedge \delta(u) = \delta(v) \wedge \\ \forall i : \text{Nat. } 1 \leq i \leq \delta(u) \Rightarrow (u[i], v[i]) \in R$$

Kongruenz-Abschluß (2)

R ist *abgeschlossen unter Kongruenz*, falls $(u, v) \in R$ gilt für alle Paare u, v , die kongruent unter R sind, d.h. falls gilt

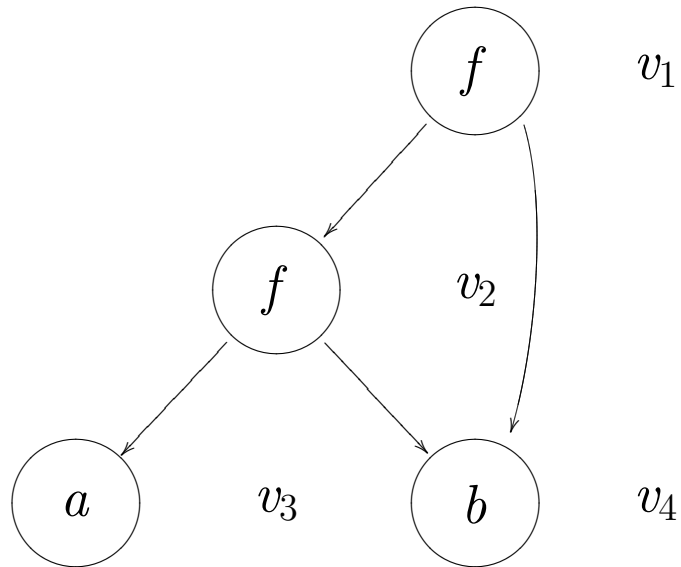
$$u \approx_R v \Rightarrow (u, v) \in R$$

Der *Kongruenzabschluß* \hat{R} von R ist die kleinste Äquivalenzrelation, die R enthält und unter Kongruenz abgeschlossen ist.

Intendierte Interpretation: Graph stellt Terme eindeutig dar (derselbe Term taucht höchstens einmal auf); die Ausgangsrelation R stellt eine Menge von Gleichungen dar; der Kongruenzabschluß von R repräsentiert alle Gleichungen, die sich aus R ableiten lassen.

Konstruktion des Kongruenz-Abschlusses wird benutzt als Grundlage einer Entscheidungsprozedur für quantorenfreie Gleichungstheorie mit uninterpretierten Funktionssymbolen (d.h. implizit universell quantifizierte Gleichungen zwischen Termen): einziges Prädikat ist die Gleichheit (=).

Beispiel 1:



Gegebene Gleichung: $f(a, b) = a$
ergibt Ausgangsrelation

$$R := \{(v_2, v_3)\}$$

d.h. v_1, v_2 sind unter R kongruent.
Kongruenzabschluß \hat{R} von R :

"Äquivalenzklassen $\{v_1, v_2, v_3\}, \{v_4\}$ mit
 $v_1 : f(f(a, b), b)$ usw.

$v_1 \approx_{\hat{R}} v_3$ entspricht der Aussage
 $f(f(a, b), b) = a$

Beispiel 2:

Ausgangsrelation:

$$R := \{(v_1, v_6), (v_3, v_6)\}$$

Konstruktion des Kongruenzabschlusses \hat{R} :

(1) v_2, v_5 kongruent unter $\hat{R} \rightsquigarrow (v_2, v_5) \in \hat{R}$

(2) v_1, v_4 kongruent unter $\hat{R} \rightsquigarrow (v_1, v_4) \in \hat{R}$

Wegen "Äquivalenzeigenschaft:

(3) $\rightsquigarrow (v_4, v_6) \in \hat{R}$

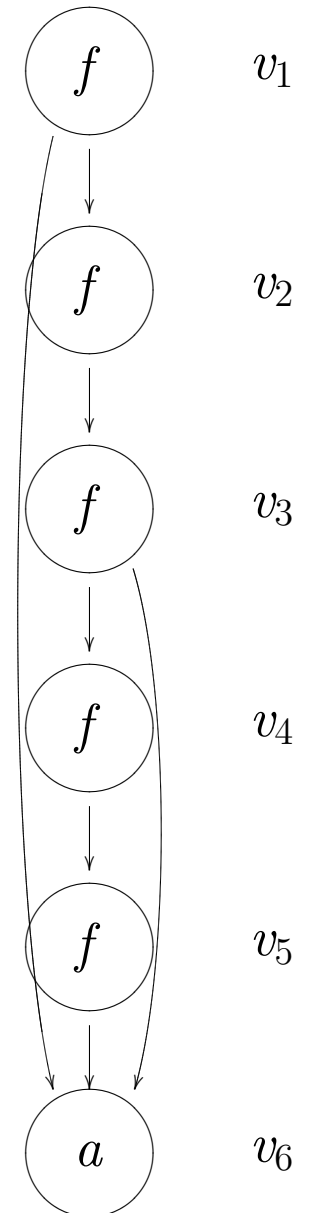
(4) $\rightsquigarrow (v_3, v_5) \in \hat{R}$

\rightsquigarrow alle Knoten sind "äquivalent" in \hat{R}

Logisch:

$$f^3(a) = a \wedge f^5(a) = a \Rightarrow f(a) = a$$

Frage: Kann $f(a) = a$ aus $f^3(a) = a$ und $f^5(a) = a$ mit Hilfe von Termersetzung geschlossen werden?



Algorithmus zur Berechnung des Kongruenz-Abschlusses:

Die Äquivalenz-Relation wird durch eine Partition (disjunkte Zerlegung) der Knotenmenge V in Äquivalenz-Klassen repräsentiert.

Vorgegebene Prozeduren:

$Union(u, v)$: Vereinigung der Äquivalenzklassen von u und v

$Find(u)$: Auffinden der Äquivalenzklasse von u – gibt eindeutigen “Namen” der Äquivalenzklasse zurück

$Merge(u, v)$: Füge (u, v) zur Relation hinzu und schließe unter Kongruenz ab.

Falls $Find(u) = Find(v)$, fertig.

P_u bezeichne jeweils die Menge aller Vorgänger der zu u äquivalenten Knoten; entsprechend P_v .

Rufe $Union(u, v)$ auf.

Für jedes Paar (x, y) mit $x \in P_u$ und $y \in P_v$, falls $Find(x) \neq Find(y)$ und $Congruent(x, y)$, rufe $Union(x, y)$ auf.

Algorithmus zur Berechnung des Kongruenz-Abschlusses (2):

Congruent(x, y): Test, ob x und y in der derzeitigen Relation kongruent sind.

Falls $\delta(x) \neq \delta(y)$, Ergebnis *Falsch*.

Für alle $1 \leq i \leq \delta(x)$, falls immer $Find(x[i]) = Find(y[i])$, Ergebnis *Wahr*.

Andernfalls Ergebnis *Falsch*.

Gültigkeit einer Formel ist äquivalent zur Unerfüllbarkeit der negierten Formel in DNF, damit reduziert auf die (Un-)Erfüllbarkeit von Konjunktionen von Literalen.

Entscheidungsprozedur: Erfüllbarkeit einer Konjunktion K von Gleichungen und Ungleichungen (d.h. negierten Gleichungen)

$$t_1 = u_1 \wedge \dots \wedge t_p = u_p \wedge r_1 \neq s_1 \wedge \dots \wedge r_q \neq s_q$$

1. Konstruiere einen Graphen G , der eindeutig alle Terme repräsentiert, die in K vorkommen.
Für einen Term t bezeichne $\tau(t)$ den ihn repräsentierenden Knoten.
 R ist zu Beginn die Identitätsrelation.
2. Für $i \leq p$, rufe $Merge(\tau(t_i), \tau(u_i))$ auf.
3. K ist unerfüllbar, falls $\tau(r_j)$ und $\tau(s_j)$ äquivalent sind für ein $j \leq q$.
4. Andernfalls ist K erfüllbar.

Erweiterung für induktive Datenstrukturen

Beispiel: quantorenfreie Theorie der Binärbäume als Erweiterung der obigen Gleichungstheorie:

Funktionssymbole $comp$, $left$, $right$

Prädikatssymbolen $atom?$, $comp?$

Axiome:

$$left(comp(x, y)) = x$$

$$right(comp(x, y)) = y$$

$$comp?(x) \Rightarrow comp(left(x), right(x)) = x$$

$$comp?(comp(x, y))$$

Beispiel eines Theorems der Theorie:

$$left(x) = left(y) \wedge right(x) = right(y) \wedge comp?(x) \wedge comp?(y)$$

$$\Rightarrow f(x) = f(y)$$

Bemerkung: Negierte Vorkommen von $atom?(t)$ bzw. $comp?(t)$ können immer vermieden werden durch Verwendung des jeweils komplementären Prädikats.

Erweiterung für induktive Datenstrukturen (2)

Entscheidungsprozedur für Erfüllbarkeit von Konjunktionen der Form

$$atom?(u_1) \wedge \dots \wedge atom?(u_q) \wedge \\ v_1 = w_1 \wedge \dots \wedge v_r = w_r \wedge x_1 \neq y_1 \dots \wedge x_s \neq y_s$$

Terme können *comp*, *left*, *right* und uninterpretierte Funktionssymbole enthalten.

1. Konstruiere Graphen G für alle Terme:

$$Merge(\tau(v_i), \tau(w_i)) \text{ für alle } i \leq r.$$

2. Für jeden Knoten, der einen Term $comp(x, y)$ repräsentiert, füge Knoten für $left(comp(x, y))$ und $right(comp(x, y))$ hinzu;
Merge mit x bzw. y .

3. Teste auf (Un-)Erfüllbarkeit der Ungleichungen.

Kombination von Entscheidungsprozeduren nach Shostak

in gewisser Weise eine “Verfeinerung” des Ansatzes von Nelson und Oppen
Basis sind *kanonisierbare* und *lösbar*e Theorien

Beispiel

$$2 * car(x) - 3 * cdr(x) = f(cdr(x))$$

$$\Rightarrow f(cons(4 * car(x) - 2 * f(cdr(x)), y)) = f(cons(6 * cdr(x), y))$$

Beispiel kombiniert Elemente verschiedener Theorien: uninterpretierte Funktionssymbole (U), Listen (L), lineare Arithmetik (A)

1. Schritt: Verarbeitung der zu beweisenden Formel (*process*)

Erzeugung von Gleichungsmengen für die Teiltheorien

$$S_V : \{\}$$

$$S_U : \{w = f(v)\}$$

$$S_L : \{u = car(x), v = cdr(x)\}$$

$$S_A : \{w = 2 * u - 3 * v\}$$

Beispiel (2)

$$2 * car(x) - 3 * cdr(x) = f(cdr(x))$$

$$\Rightarrow f(cons(4 * car(x) - 2 * f(cdr(x)), y)) = f(cons(6 * cdr(x), y))$$

$$S_V : \{\} \quad S_U : \{w = f(v)\} \quad S_L : \{u = car(x), v = cdr(x)\} \quad S_A : \{w = 2 * u - 3 * v\}$$

2. Schritt: Reduktion der Terme der Zielgleichung auf Normalform (*Canonize*)

Linke Seite:

$$4 * car(x) - 2 * f(cdr(x))$$

$$\rightsquigarrow 4 * u - 2 * f(v) \rightsquigarrow 4 * u - 2 * w$$

$$\rightsquigarrow 4 * u - 2 * (2 * u - 3 * v)$$

$$\rightsquigarrow 6 * v$$

Rechte Seite:

$$6 * cdr(x) \rightsquigarrow 6 * v$$

Shostak-Theorien

- Eine kanonisierbare und lösbare Theorie ist eine “Shostak-Theorie”.
- Ein Kanonisierer σ bildet Terme auf Terme in Normalform ab, so dass Terme, die in der Theorie gleich sind, auf dieselbe Normalform abgebildet werden.
- Ein Solver *solve* bildet eine Gleichung auf eine äquivalente gelöste Form ab.

Kanonisierbare Theorien

Eine Theorie T wird *kanonisierbar* genannt, wenn es einen “Kanonisierer” (*canonizer*), d.h. eine berechenbare Funktion σ gibt mit

- $\models_T a = b$ g.d.w. $\sigma(a) \doteq \sigma(b)$
- $\text{vars}(\sigma(a)) \subset \text{vars}(a)$
- $\sigma(b) \doteq b$ für jeden Unterterm b von $\sigma(a)$
- $\sigma(\sigma(a)) \doteq \sigma(a)$

Aus der Definition folgt $\models_T \sigma(a) = a$

Ein Term a ist *in kanonischer Form*, wenn gilt $\sigma(a) \doteq a$.

Beispiel: Kanonisierer für lineare Arithmetik]:

$$\sigma_A(y + x + 3 + x) \doteq 2x + y + 3$$

Kanonisierer liefert eine geordnete Summe von “Monomialen”. (Ordnung z.B. durch alphabetische Anordnung der Variablen)

Ein Kanonisierer σ liefert eine Prozedur, um einfache Gleichheiten $a = b$ zu entscheiden (Lösung des “Wortproblems” für T).

Lösbare Theorien

Eine Theorie T wird *lösbar* genannt, wenn es eine berechenbare Prozedur $solve(a = b)$ gibt mit

- $solve(a = b) \doteq bot$ g.d.w. die Gleichung $a = b$ in T unerfüllbar ist.
- Andernfalls liefert $solve(a = b)$ eine (funktionale) Lösungsmenge S , so dass
 - $dom(S) \subseteq vars(a = b)$
 - $solve(a = b)$ erhält $a = b$ in T (“T -preserves”]

Beispiel: Theorie der Listen

Signatur: $\Sigma_L := \{cons(-, -), car(-), cdr(-)\}$

Theorie L definiert durch die Axiome (implizit universell quantifiziert):

$$car(cons(x, y)) = x$$

$$cdr(cons(x, y)) = y$$

$$cons(car(x), cdr(x)) = x$$

$$car(x) \neq x$$

$$cdr(x) \neq x$$

$$car(cdr(x)) \neq x$$

...

Kanonisierer: $\sigma_L(a)$ ist die Normalform, die von dem (terminierenden und konfluenten) Termersetzungssystem, das sich aus den Axiomen ergibt, geliefert wird.

Beispiel für Solver über Listen

Lösen von $x = \mathit{cons}(\mathit{car}(x), y)$ in L :

Variablenmenge $\{x, y\}$

$\{x = \mathit{cons}(\mathit{car}(x), y)\}; \quad \emptyset$

$\{\mathit{cons}(z1, z2) = \mathit{cons}(z1, y)\}; \quad \{x = \mathit{cons}(z1, z2)\}$

$\{z1 = z1, z2 = y\}; \quad \{x = \mathit{cons}(z1, z2)\}$

$\{z2 = y\}; \quad \{x = \mathit{cons}(z1, z2)\}$

$\emptyset; \quad \{x = \mathit{cons}(z1, z2), y = z2\}$

Boolsche Terme

(Theorie der *booleans*)

Signatur: $\Sigma_B := \{true, false, ITE(, -, -)\}$

Kanonisierer σ_B liefert für eine vorgegebene Ordnung auf den Variablen äquivalente reduzierte OBDDs (*ordered binary decision diagrams*)

Solver $solve_B$:

$$solve_B(a = b) = solve(a \Leftrightarrow b)$$

$$solve(true) = \{\}$$

$$solve(false) = \perp$$

$$solve(ITE(x, p, n)) = \{x = (p \wedge (n \Rightarrow \delta))\}$$

$$\cup solve_B(p \vee n)$$

hierbei ist δ eine neue Variable; \circ_B ist Vereinigung (\cup nach Substitution (s. Beispiel))

Beispiel: Solver für boolesche Terme

$$\begin{aligned} & \text{solve}_B(x \wedge y = \neg x) \\ &= \text{solve}_B(\text{ITE}(x, \text{ITE}(y, \text{false}, \text{true}), \text{false})) \\ &= \{x = \text{ITE}(y, \text{false}, \text{true})\} \\ &\quad \circ_B \text{solve}_B(\text{ITE}(y, \text{false}, \text{true})) \\ &= \{x = \text{true}, y = \text{false}\} \end{aligned}$$