Teorie - Programare Logică și Funcțională

Seria 36

2025

1 Algoritmul de unificare

O *substituţie* este o funcţie parţială de la variabile la termeni, adică $\sigma : V \to Trm_{\mathcal{L}}$. Un *unificator* pentru doi termeni t_1 şi t_2 este o substituţie θ astfel încât $\theta(t_1) = \theta(t_2)$. Un unificator ν pentru t_1 şi t_2 este un *cel mai general unificator* dacă pentru orice alt unificator ν' pentru t_1 şi t_2, există o substituţie μ astfel încât $\nu' = \nu; \mu$.

Algoritmul de unificare:

	Lista soluţie S	Lista de rezolvat R
Iniţial	$\emptyset$	$t_1 \doteq t'_1, \ldots, t_n \doteq t'_n$
SCOATE	S	$R', t \doteq t$
	S	R'
DESCOMPUNE	S	$R', f(t_1, \ldots, t_n) \doteq f(t'_1, \ldots, t'_n)$
	S	$R', t_1 \doteq t'_1, \ldots t_n \doteq t'_n$
REZOLVĂ	S	$R', x \doteq t$ sau $t \doteq x$, x nu apare în t
	$x \doteq t$, $S[x \leftarrow t]$	$R'[x \leftarrow t]$
Final	S	$\emptyset$

Algoritmul *se termină normal* dacă $R = \emptyset$ (în acest caz, în S are un unificator pentru termenii din lista iniţială R).

Algoritmul este oprit cu concluzia *inexistenţei unui unificator* dacă:

(i) În R există o ecuaţie de forma $f(t_1, \ldots, t_n) \doteq g(t'_1, \ldots, t'_k)$ cu $f \neq g$. Simbolurile de constantă se consideră simboluri de funcţie de aritate 0.

(ii) În R există o ecuaţie de forma $x \doteq t$ sau $t \doteq x$ şi variabila x apare în termenul t.

2 Deductie naturala

Sistemul de reguli al deducţiei naturale

$$\frac{\varphi \quad \psi}{\varphi \wedge \psi}\ (\wedge i) \qquad \frac{\varphi \wedge \psi}{\varphi}\ (\wedge e_1) \qquad \frac{\varphi \wedge \psi}{\psi}\ (\wedge e_2)$$

$$\frac{\begin{array}{|c|}\hline \varphi \\ \vdots \\ \psi \\ \hline \end{array}}{\varphi \to \psi}\ (\to i) \qquad \frac{\varphi \quad \varphi \to \psi}{\psi}\ (\to e)$$

$$\frac{\varphi}{\varphi \vee \psi}\ (\vee i_1) \qquad \frac{\psi}{\varphi \vee \psi}\ (\vee i_2) \qquad \frac{\varphi \vee \psi \qquad \begin{array}{|c|}\hline \varphi \\ \vdots \\ \chi \\ \hline \end{array} \quad \begin{array}{|c|}\hline \psi \\ \vdots \\ \chi \\ \hline \end{array}}{\chi}\ (\vee e)$$

$$\frac{\begin{array}{|c|}\hline \varphi \\ \vdots \\ \bot \\ \hline \end{array}}{\neg\varphi}\ (\neg i) \qquad \frac{\varphi \quad \neg\varphi}{\bot}\ (\neg e)$$

$$\frac{\varphi}{\neg\neg\varphi}\ (\neg\neg i) \qquad \frac{\neg\neg\varphi}{\varphi}\ (\neg\neg e)$$

$$\frac{}{\varphi \vee \neg\varphi}\ \textbf{TND} \qquad \frac{\bot}{\varphi}\ (\bot e)$$

TND (*tertium non datur*) este regulă derivată.

Atenţie! La acest sistem se adaugă regula de copiere
Regula de copiere:

- la un pas al unei demonstraţii poate fi copiată orice formulă demonstrată anterior.
- la un pas al unei demonstraţii nu pot fi copiate formule din interiorul cutiilor care sunt închise în acel moment.

3 Puncte fixe

O *mulţime parţial ordonată (mpo)* este o pereche $(M, \leq)$ unde $\leq \subseteq M \times M$ este o relaţie de ordine (i.e., reflexivă, antisimetrică, tranzitivă). O mulţime parţial ordonată $(C, \leq)$ este *completă (cpo)* dacă C are prim element $\bot$ ($\bot \leq x$ oricare $x \in C$) şi $\bigvee_n x_n$ există în C pentru orice lanţ $x_1 \leq x_2 \leq x_3 \leq \ldots$.

Fie $(C, \leq_C)$ o mulţime parţial ordonată. Un element $a \in C$ este *punct fix* al unei funcţii $f : C \to C$ dacă $f(a) = a$. Un element $lfp \in C$ este *cel mai mic punct fix* al unei funcţii $f : C \to C$ dacă este punct fix şi pentru orice alt punct fix $a \in C$ al lui f avem $lfp \leq_C a$.

Fie $(A, \leq_A)$ şi $(B, \leq_B)$ mulţimi parţial ordonate. O funcţie $f : A \to B$ este *monotonă (crescătoare)* dacă $a_1 \leq_A a_2$ implică $f(a_1) \leq_B f(a_2)$ oricare $a_1, a_2 \in A$.

O *clauză definită propoziţională* este o formulă care poate avea una din formele:

- q (clauză unitate)
- $p_1 \wedge \ldots \wedge p_k \to q$

unde $q, p_1, \ldots, p_n$ sunt variabile propoziţionale.

Fie S o mulţime de clauze definite propoziţionale. Fie A mulţimea variabilelor propoziţionale $p_1, p_2, \ldots$ care apar în S şi $Baza = \{p_i \mid p_i \in S\}$ mulţimea clauzelor unitate din S. Definim funcţia $f_S : \mathcal{P}(A) \to \mathcal{P}(A)$ prin

$$f_S(Y) = Y \cup Baza \cup \{a \in A \mid (s_1 \wedge \ldots \wedge s_n \to a) \text{ este în } S, s_1 \in Y, \ldots, s_n \in Y\}$$

Fie $(A, \leq_A)$ şi $(B, \leq_B)$ mulţimi parţial ordonate complete. O funcţie $f : A \to B$ este *continuă* dacă $f(\bigvee_n a_n) = \bigvee_n f(a_n)$ pentru orice lanţ $\{a_n\}_n$ din A. Observăm că orice funcţie continuă este crescătoare.

Pentru orice mulţime de clauze definite propoziţionale S, funcţia f_S este continuă.

Teorema 1 (Knaster-Tarski)**.** *Fie $(C, \leq)$ o mulţime parţial ordonată completă şi $\mathbf{F} : C \to C$ o funcţie continuă. Atunci*

$$a = \bigvee_n \mathbf{F}^n(\bot)$$

este cel mai mic punct fix al funcţiei $\mathbf{F}$.

4 Rezolutie SLD

- O *clauză definită* este o formulă de forma:
  - $P(t_1, \ldots, t_n)$ (formulă atomică), unde P este un simbol de predicat, iar $t_1, \ldots, t_n$ termeni
  - $P_1 \wedge \ldots \wedge P_n \to Q$, unde toate P_i, Q sunt formule atomice.
- O regulă din Prolog $\mathtt{Q} :- \mathtt{P_1}, \ldots, \mathtt{P_n}$ este o clauză $P_1 \wedge \ldots \wedge P_n \to Q$, iar un fapt din Prolog $\mathtt{P}(\mathtt{t_1}, \ldots, \mathtt{t_n})$ este o formulă atomică $P(t_1, \ldots, t_n)$.
- O clauză definită $P_1 \wedge \ldots \wedge P_n \to Q$ poate fi gândită ca formula $Q \vee \neg P_1 \vee \ldots \vee \neg P_n$.
- Pentru o mulţime de clauze definite T, *regula rezolutiei SLD* este

$$\text{SLD} \quad \boxed{\frac{\neg P_1 \vee \cdots \vee \neg P_i \vee \cdots \vee \neg P_n}{(\neg P_1 \vee \cdots \vee \neg Q_1 \vee \cdots \vee \neg Q_m \vee \cdots \vee \neg P_n)\theta}}$$

  unde $Q \vee \neg Q_1 \vee \cdots \vee \neg Q_m$ este o clauză definită din T (în care toate variabilele au fost redenumite) şi θ este c.g.u pentru P_i şi Q.
- Fie T o mulţime de clauze definite şi $P_1 \wedge \ldots \wedge P_m$ o ţintă, unde P_i sunt formule atomice. O derivare din T prin rezoluţie SLD este o secvenţă $G_0 := \neg P_1 \vee \ldots \vee \neg P_m$, G_1, ..., G_k, ... în care G_{i+1} se obţine din G_i prin regula SLD. Dacă există un k cu $G_k = \square$ (clauza vidă), atunci derivarea se numeşte *SLD-respingere.*

Teorema 2 (Completitudinea SLD-rezoluţiei)**.** *Sunt echivalente:*

(i) există o SLD-respingere a lui $P_1 \wedge \ldots \wedge P_m$ din T,

(ii) $T \vDash P_1 \wedge \cdots \wedge P_m$.

Fie T o mulţime de clauze definite şi o ţintă $G_0 = \neg P_1 \vee \ldots \vee \neg P_m$. Un *arbore SLD* este definit astfel:

- Fiecare nod al arborelui este o ţintă (posibil vidă)
- Rădăcina este G_0
- Dacă arborele are un nod G_i, iar G_{i+1} se obţine din G_i folosind regula SLD folosind o clauză $C_i \in T$, atunci nodul G_i are copilul G_{i+1}. Muchia dintre G_i şi G_{i+1} este etichetată cu C_i.

Dacă un arbore SLD cu rădăcina G_0 are o frunză $\square$ (clauza vidă), atunci există o SLD-respingere a lui G_0 din T.

5 Lambda-calcul

Sintaxa λ-calculului:

$$t = x \mid \lambda x.t \mid t\ t$$

λ-termeni. Fie $Var = \{x, y, z, ...\}$ o mulțime infinită de variabile. Mulțimea λ-termenilor ΛT este definită inductiv, astfel:

[Variabilă] $Var \subseteq \Lambda T$
[Aplicare] dacă $t_1, t_2 \in \Lambda T$ atunci $(t_1\ t_2) \in \Lambda T$
[Abstractizare] dacă $x \in Var$ și $t \in \Lambda T$, atunci $(\lambda x.t) \in \Lambda T$

Convenții de scriere.

- în scrierea λ-termenilor vom elimina parantezele exterioare și vom scrie, de exemplu, xy în loc de (xy), sau $\lambda x.xy$ în loc de $(\lambda x.(xy))$;
- aplicarea este asociativă la stânga: $t_1t_2t_3$ este $(t_1t_2)t_3$;
- corpul abstractizării este extins la dreapta: $\lambda x.t_1t_2$ este $\lambda x.(t_1t_2)$;
- scriem $\lambda xyz.t$, în loc de $\lambda x.\lambda y.\lambda z.t$.

5.1 Variabile libere și legate

Variabile libere și legate. Pentru un termen $\lambda x.t$ spunem că:

- aparițiile variabilei x în t sunt legate *(bound)*;
- λx este legătura *(binder)*, iar t este domeniul *(scope)* legării;
- o apariție a unei variabile este liberă *(free)* dacă apare într-o poziție în care nu este legată.

Un termen fără variabile libere se numește **închis** *(closed)*.

Mulțimea variabilelor libere FV(t). Pentru un λ-termen t, mulțimea variabilelor libere este definită astfel:

[Variabilă] $FV(x) = \{x\}$

[Aplicare] $FV(t_1t_2) = FV(t_1) \cup FV(t_2)$

[Abstractizare] $FV(\lambda x.t) = FV(t) - \{x\}$

5.2 Substituții

Fie t un λ-termen și $x \in Var$. Pentru un λ-termen u vom nota prin $[u/x]t$ rezultatul înlocuirii tuturor aparițiilor libere ale lui x cu u în t.

[Variabilă] $[u/x]x = u$

[Variabilă] $[u/x]y = y$ dacă $x \neq y$

[Aplicare] $[u/x](t_1t_2) = [u/x]t_1[u/x]t_2$

[Abstractizare] $[u/x]\lambda x.t = \lambda x.t$

[Abstractizare] $[u/x]\lambda y.t = \lambda y.[u/x]t$ unde $x \neq y$ și $y \notin FV(u)$

[Abstractizare] $[u/x]\lambda y.t = \lambda z.[u/x]([z/y]t)$ unde $x \neq y$ și $y \in FV(u)$

Variabilele legate pot fi redenumite.

5.3 α-conversie (α-echivalență)

α-conversia $=_\alpha$ este relația binară care satisface următoarele proprietăți:

[Reflexivitate] $t =_\alpha t$

[Simetrie] $t_1 =_\alpha t_2$ implică $t_2 =_\alpha t_1$

[Tranzitivitate] $t_1 =_\alpha t_2$ și $t_2 =_\alpha t_3$ implică $t_1 =_\alpha t_3$

[Redenumire] $\lambda x.t =_\alpha \lambda y.[y/x]t$ dacă $y \notin FV(t)$

[Compatibilitate] $t_1 =_\alpha t_2$ implică $tt_1 =_\alpha tt_2$, $t_1t =_\alpha t_2t$ și $\lambda x.t_1 =_\alpha \lambda x.t_2$

Compatibilitatea cu substituția.

$$t_1 =_\alpha t_2 \text{ și } u_1 =_\alpha u_2 \text{ implică } [u_1/x]t_1 =_\alpha [u_2/x]t_2$$

5.4 β-reducția

β-reducția este o relație definită pe mulțimea α-termenilor, $\beta \subseteq \Lambda T \times \Lambda T$, unde

[Aplicarea] $(\lambda x.y)u \to_\beta [u/x]t$

[Compatibilitatea] $t_1 \to_\beta t_2$ implică $tt_1 \to_\beta tt_2$, $t_1t \to_\beta t_2t$ și $\lambda x.t_1 \to_\beta \lambda x.t_2$

Închiderea reflexivă și tranzitivă a acestei relații se notează $\to_\beta^* \subseteq \Lambda T \times \Lambda T$, iar $t_1 \to_\beta^* t_2$ dacă există $n \geq 0$ și $u_0, ..., u_n$ asfel încât $t_1 =_\alpha u_0 \to_\beta u_1 \to_\beta ... \to_\beta u_n =_\alpha t_2$.

Un termen poate fi β-redus în mai multe moduri, iar proprietatea de confluență ne asigură că vom ajunge întotdeauna la același rezultat. (forma normală este unică, modulo α-echivalență)

6 Exemplu `foldr`

```
Pasul 0: scriem functia in forma recursiva

doubleOddLTn :: [Int] -> Int -> [Int]
doubleOddLTn [] _ = []
doubleOddLTn (x:xs) n
    | odd x && x < n    = (2 * x) : doubleOddLTn xs n
    | otherwise         = doubleOddLTn xs n

Scopul este sa aducem aceasta functie in forma proprietatii de universalitate

DACA
    g [] = init
    g (x:xs) = f x (g xs)
ATUNCI
    g = foldr f init

Pasul 1: redenumim functia doubleOddLTn cu g

g [] _ = []
g (x:xs) n
    | odd x && x < n    = (2 * x) : g xs n
    | otherwise         = g xs n

Pasul 2: schimbam din guards in if-then-else

g [] _ = []
g (x:xs) n = if (odd x && x < n) then (2 * x) : g xs n else g xs n

Pasul 3: aplicam currying si uncurrying

g [] = \_ -> []
g (x:xs) = \n -> if (odd x && x < n) then (2 * x) : g xs n else g xs n

Pasul 4: aplicam proprietatea de universalitate
    (am gasit deja ca init = \_ -> [])
g (x:xs) = f x (g xs)
g (x:xs) = \n -> if (odd x && x < n) then (2 * x) : g xs n else g xs n

    deci

f x (g xs) = \n -> if (odd x && x < n) then (2 * x) : g xs n else g xs n

Pasul 5: redenumim g xs = u (APROAPE DE FIECARE DATA)

f x u = \n -> if (odd x && x < n) then (2 * x) : u n else u n

Pasul 6: extragem functia f

f = \x -> \u -> \n -> if (odd x && x < n) then (2 * x) : u n else u n

sau

f = \x u n -> if (odd x && x < n) then (2 * x) : u n else u n

in acest moment, avem f si init, deci putem scrie g = foldr f init

doubleOddLTnFoldr :: [Int] -> Int -> [Int]
doubleOddLTnFoldr = foldr (\x u n -> if (odd x && x < n) then (2 * x) : u n else u n) (\_ -> [])
```

Semantica unei variabile

(Id) $\langle x\ ,\ \sigma\rangle \to \langle i\ ,\ \sigma\rangle \quad$ *dacă* $i = \sigma(x)$

Semantica adunării a două expresii aritmetice

(Add) $\langle i_1 + i_2\ ,\ \sigma\rangle \to \langle i\ ,\ \sigma\rangle \quad$ *dacă* $i = i_1 + i_2$

$$\frac{\langle a_1\ ,\ \sigma\rangle \to \langle a_1'\ ,\ \sigma\rangle}{\langle a_1 + a_2\ ,\ \sigma\rangle \to \langle a_1' + a_2\ ,\ \sigma\rangle} \qquad \frac{\langle a_2\ ,\ \sigma\rangle \to \langle a_2'\ ,\ \sigma\rangle}{\langle a_1 + a_2\ ,\ \sigma\rangle \to \langle a_1 + a_2'\ ,\ \sigma\rangle}$$

Observatie: ordinea de evaluare a argumentelor este nespecificată.

Semantica operatorului de comparatie

(Leq-false) $\langle i_1 =< i_2\ ,\ \sigma\rangle \to \langle \texttt{false}\ ,\ \sigma\rangle \quad$ *dacă* $i_1 > i_2$

(Leq-true) $\langle i_1 =< i_2\ ,\ \sigma\rangle \to \langle \texttt{true}\ ,\ \sigma\rangle \quad$ *dacă* $i_1 \leq i_2$

$$\frac{\langle a_1\ ,\ \sigma\rangle \to \langle a_1'\ ,\ \sigma\rangle}{\langle a_1 =< a_2\ ,\ \sigma\rangle \to \langle a_1' =< a_2\ ,\ \sigma\rangle} \qquad \frac{\langle a_2\ ,\ \sigma\rangle \to \langle a_2'\ ,\ \sigma\rangle}{\langle a_1 =< a_2\ ,\ \sigma\rangle \to \langle a_1 =< a_2'\ ,\ \sigma\rangle}$$

Semantica negatiei

(!-false) $\langle \texttt{not(true)}\ ,\ \sigma\rangle \to \langle \texttt{false}\ ,\ \sigma\rangle$

(!-true) $\langle \texttt{not(false)}\ ,\ \sigma\rangle \to \langle \texttt{true}\ ,\ \sigma\rangle$

$$\frac{\langle a\ ,\ \sigma\rangle \to \langle a'\ ,\ \sigma\rangle}{\langle \texttt{not}\ (a)\ ,\ \sigma\rangle \to \langle \texttt{not}\ (a')\ ,\ \sigma\rangle}$$

Semantica si-ului

(And-false) $\langle \texttt{and}\ (\texttt{false},\ b_2)\ ,\ \sigma\rangle \to \langle \texttt{false}\ ,\ \sigma\rangle$

(And-true) $\langle \texttt{and}\ (\texttt{true},\ b_2)\ ,\ \sigma\rangle \to \langle b_2\ ,\ \sigma\rangle$

$$\frac{\langle b_1\ ,\ \sigma\rangle \to \langle b_1'\ ,\ \sigma\rangle}{\langle \texttt{and}\ (b_1\ ,\ b_2)\ ,\ \sigma\rangle \to \langle \texttt{and}\ (b_1'\ ,\ b_2)\ ,\ \sigma\rangle}$$

Semantica blocurilor

(BLOCK) $\langle \{\ s\ \},\ \sigma\rangle \to \langle s\ ,\ \sigma\rangle$

Semantica compunerii secventiale

(NEXT-STMT) $\langle \texttt{skip};\ s_2\ ,\ \sigma\rangle \to \langle s_2\ ,\ \sigma\rangle$

$$\frac{\langle s_1\ ,\ \sigma\rangle \to \langle s_1'\ ,\ \sigma'\rangle}{\langle s_1\ ;\ s_2\ ,\ \sigma\rangle \to \to \langle s_1'\ ;\ s_2\ ,\ \sigma'\rangle}$$

Semantica atribuirii

(ASGN) $\langle x = i\ ,\ \sigma\rangle \to \langle \texttt{skip}\ ,\ \sigma'\rangle \quad \textit{dacă}\ \sigma' = \sigma_{x \leftarrow i}$

$$\frac{\langle a\ ,\ \sigma\rangle \to \langle a'\ ,\ \sigma\rangle}{\langle x = a\ ,\ \sigma\rangle \to \langle x = a'\ ,\ \sigma\rangle}$$

Semantica lui `if`

(IF-TRUE) $\langle \texttt{if}\ (\texttt{true}, bl_1, bl_2)\ ,\ \sigma\rangle \to \langle bl_1\ ,\ \sigma\rangle$

(IF-FALSE) $\langle \texttt{if}\ (\texttt{false}, bl_1, bl_2)\ ,\ \sigma\rangle \to \langle bl_2\ ,\ \sigma\rangle$

$$\frac{\langle b\ ,\ \sigma\rangle \to \langle b'\ ,\ \sigma\rangle}{\langle \texttt{if}\ (b, bl_1, bl_2)\ ,\ \sigma\rangle \to \langle \texttt{if}\ (b', bl_1, bl_2)\ ,\ \sigma\rangle}$$

Semantica lui `while`

(WHILE) $\langle \texttt{while}\ (b, bl)\ ,\ \sigma\rangle \to \langle \texttt{if}\ (b, bl\ \texttt{;while}\ (b, bl), \texttt{skip})\ ,\ \sigma\rangle$

Semantica programelor

(PGM)
$$\frac{\langle a_1\ ,\ \sigma_1\rangle \to \langle a_2\ ,\ \sigma_2\rangle}{\langle (\texttt{skip}, a_1)\ ,\ \sigma_1\rangle \to \langle (\texttt{skip}, a_2)\ ,\ \sigma_2\rangle}$$

$$\frac{\langle s_1\ ,\ \sigma_1\rangle \to \langle s_2\ ,\ \sigma_2\rangle}{\langle (s_1, a)\ ,\ \sigma_1\rangle \to \langle (s_2, a)\ ,\ \sigma_2\rangle}$$