Curs 3

Cuprins

Formule și satisfiabilitate în logica propozițională

- Limbajul PL
  - variabile propoziționale: $Var = \{p, q, v, \ldots\}$
  - conectori logici: $\neg$ (unar), $\to$, $\wedge$, $\vee$, $\leftrightarrow$ (binari)

- Formulele PL

$$
\begin{array}{lcl}
var & ::= & p \mid q \mid v \mid \ldots \\
form & ::= & var \mid (\neg form) \mid form \wedge form \mid form \vee form \\
 & & \mid form \to form \mid form \leftrightarrow form
\end{array}
$$

Notăm cu *Form* mulțimea formulelor.

- O evaluare $e : Var \to \{0, 1\}$ este model al formulei φ dacă $e^+(\varphi) = 1$.
- Două formule sunt echivalente dacă au aceleași modele: $\varphi \sim \psi$ ddacă $e^+(\varphi) = e^+(\psi)$ or. $e : Var \to \{0, 1\}$
- O formulă φ este satisfiabilă dacă are un model.

FNC în logica propozițională

Orice formulă poate fi adusa la forma normală conjunctivă (FNC) prin urmatoarele transformări:

1. înlocuirea implicațiilor și echivalențelor

$$\varphi \to \psi \sim \neg\varphi \vee \psi,$$
$$\varphi \leftrightarrow \psi \sim (\neg\varphi \vee \psi) \wedge (\neg\psi \vee \varphi)$$

FNC în logica propozițională

Orice formulă poate fi adusa la forma normală conjunctivă (FNC) prin urmatoarele transformări:

1. înlocuirea implicațiilor și echivalențelor

$$\varphi \to \psi \sim \neg\varphi \vee \psi,$$
$$\varphi \leftrightarrow \psi \sim (\neg\varphi \vee \psi) \wedge (\neg\psi \vee \varphi)$$

2. regulile De Morgan

$$\neg(\varphi \vee \psi) \sim \neg\varphi \wedge \neg\psi,$$
$$\neg(\varphi \wedge \psi) \sim \neg\varphi \vee \neg\psi,$$

FNC în logica propozițională

Orice formulă poate fi adusa la forma normală conjunctivă (FNC) prin urmatoarele transformări:

1. înlocuirea implicațiilor și echivalențelor

$$\varphi \to \psi \sim \neg\varphi \vee \psi,$$
$$\varphi \leftrightarrow \psi \sim (\neg\varphi \vee \psi) \wedge (\neg\psi \vee \varphi)$$

2. regulile De Morgan

$$\neg(\varphi \vee \psi) \sim \neg\varphi \wedge \neg\psi,$$
$$\neg(\varphi \wedge \psi) \sim \neg\varphi \vee \neg\psi,$$

3. principiului dublei negații

$$\neg\neg\psi \sim \psi$$

FNC în logica propozițională

Orice formulă poate fi adusa la forma normală conjunctivă (FNC) prin urmatoarele transformări:

1. înlocuirea implicațiilor și echivalențelor

$$\varphi \to \psi \sim \neg\varphi \vee \psi,$$
$$\varphi \leftrightarrow \psi \sim (\neg\varphi \vee \psi) \wedge (\neg\psi \vee \varphi)$$

2. regulile De Morgan

$$\neg(\varphi \vee \psi) \sim \neg\varphi \wedge \neg\psi,$$
$$\neg(\varphi \wedge \psi) \sim \neg\varphi \vee \neg\psi,$$

3. principiului dublei negații

$$\neg\neg\psi \sim \psi$$

4. distributivitatea

$$\varphi \vee (\psi \wedge \chi) \sim (\varphi \vee \psi) \wedge (\varphi \vee \chi),$$
$$(\psi \wedge \chi) \vee \varphi \sim (\psi \vee \varphi) \wedge (\chi \vee \varphi)$$

Exercițiu: Determinați FNC pentru $\varphi := (p \leftrightarrow \neg q) \to p$

Regula rezoluției

- Un **literal** este o variabila sau negatia unei variabile.
- O **clauza** este o multime finita de literali.
- Regula rezoluției păstrează satisfiabilitatea.

Regula Rezolutiei

$$Rez \quad \frac{C_1 \cup \{p\}, C_2 \cup \{\neg p\}}{C_1 \cup C_2}$$

unde $\{p, \neg p\} \cap C_1 = \emptyset$ si $\{p, \neg p\} \cap C_2 = \emptyset$.

Procedura Davis-Putnam DPP (informal)

DPP

Intrare: o mulțime $\mathcal{C}$ de clauze

Se repetă următorii pași:

- se elimină clauzele triviale
- se alege o variabilă p
- se adaugă la mulțimea de clauze toți rezolvenții obținuti prin aplicarea *Rez* pe variabila p
- se șterg toate clauzele care conțin p sau $\neg p$

Ieșire: dacă la un pas s-a obținut $\square$, mulțimea $\mathcal{C}$ nu este satisfiabilă; altfel $\mathcal{C}$ este satisfiabilă.

Exercițiu: Să se cerceteze satisfiabilitatea formulei:

$$\varphi := (\neg v_1 \vee \neg v_2) \wedge ((v_3 \rightarrow \neg v_2) \vee v_1) \wedge (v_1 \rightarrow v_2) \wedge (v_2 \wedge v_3)$$

Clauze propoziționale definite. Rezoluția SLD

Clauze propoziționale definite

- O clauză propozițională definită este o formulă care poate avea una din formele:
  1. q
  2. $p_1 \wedge \ldots \wedge p_k \rightarrow q$

  unde $q, p_1, \ldots, p_n$ sunt variabile propoziționale

Clauze propoziționale definite

- O clauză propozițională definită este o formulă care poate avea una din formele:
  1. q (un fapt în Prolog `q.`)
  2. $p_1 \land \ldots \land p_k \to q$ (o regulă în Prolog `q :- p1,...,pk`)

  unde $q, p_1, \ldots, p_n$ sunt variabile propoziționale

Clauze propoziționale definite

- O clauză propozițională definită este o formulă care poate avea una din formele:
  1. q (un fapt în Prolog `q.`)
  2. $p_1 \wedge \ldots \wedge p_k \to q$ (o regulă în Prolog `q :- p1,...,pk`)

  unde $q, p_1, \ldots, p_n$ sunt variabile propoziționale
- Numim variabilele propoziționale atomi.

Clauze propoziționale definite

- O clauză propozițională definită este o formulă care poate avea una din formele:
  1. q (un fapt în Prolog `q.`)
  2. $p_1 \wedge \ldots \wedge p_k \to q$ (o regulă în Prolog `q :- p1,...,pk`)

  unde $q, p_1, \ldots, p_n$ sunt variabile propoziționale

- Numim variabilele propoziționale atomi.

Programare logică – cazul logicii propoziționale

- Un "program logic" este o listă $Cd_1, \ldots, Cd_n$ de clauze definite.

Clauze propoziționale definite

- O clauză propozițională definită este o formulă care poate avea una din formele:
  1. q (un fapt în Prolog `q.`)
  2. $p_1 \land \ldots \land p_k \to q$ (o regulă în Prolog `q :- p1,...,pk`)

  unde $q, p_1, \ldots, p_n$ sunt variabile propoziționale
- Numim variabilele propoziționale atomi.

Programare logică – cazul logicii propoziționale

- Un "program logic" este o listă $Cd_1, \ldots, Cd_n$ de clauze definite.
- O întrebare este o listă $q_1, \ldots, q_m$ de atomi.

Clauze propoziționale definite

- O clauză propozițională definită este o formulă care poate avea una din formele:
  1. q (un fapt în Prolog `q.`)
  2. $p_1 \wedge \ldots \wedge p_k \rightarrow q$ (o regulă în Prolog `q :- p1,...,pk`)

  unde $q, p_1, \ldots, p_n$ sunt variabile propoziționale
- Numim variabilele propoziționale atomi.

Programare logică – cazul logicii propoziționale

- Un "program logic" este o listă $Cd_1, \ldots, Cd_n$ de clauze definite.
- O întrebare este o listă $q_1, \ldots, q_m$ de atomi.
- Sarcina sistemului este să stabilească:

$$Cd_1, \ldots, Cd_n \models q_1 \wedge \ldots \wedge q_m.$$

Clauze propoziționale definite

- O clauză propozițională definită este o formulă care poate avea una din formele:
  1. q (un fapt în Prolog `q.`)
  2. $p_1 \land \ldots \land p_k \to q$ (o regulă în Prolog `q :- p1,...,pk`)

  unde $q, p_1, \ldots, p_n$ sunt variabile propoziționale
- Numim variabilele propoziționale atomi.

Programare logică – cazul logicii propoziționale

- Un "program logic" este o listă $Cd_1, \ldots, Cd_n$ de clauze definite.
- O întrebare este o listă $q_1, \ldots, q_m$ de atomi.
- Sarcina sistemului este să stabilească:

$$Cd_1, \ldots, Cd_n \models q_1 \land \ldots \land q_m.$$

Vom studia metode sintactice pentru a rezolva această problemă!

Clauze Horn propoziționale

Fie $q, p_1, \ldots, p_n, q_1, \ldots, q_m$ variabile propoziționale.

- Observăm că $((p_1 \wedge \ldots \wedge p_k) \to q) \sim (\neg p_1 \vee \ldots \vee \neg p_k \vee q)$
- O clauză propozițională definită este o formulă care poate avea una din formele:
  1. q
  2. $\neg p_1 \vee \ldots \vee \neg p_k \vee q$

Clauze Horn propoziționale

Fie $q, p_1, \ldots, p_n, q_1, \ldots, q_m$ variabile propoziționale.

- Observăm că $((p_1 \land \ldots \land p_k) \to q) \sim (\neg p_1 \lor \ldots \lor \neg p_k \lor q)$
- O clauză propozițională definită este o formulă care poate avea una din formele:
  1. q (un fapt în Prolog `q.`)
  2. $\neg p_1 \lor \ldots \lor \neg p_k \lor q$ (o regulă în Prolog `q :- p1,...,pk`)
- Observăm că $\Gamma \models q_1 \land \ldots \land q_m$ ddacă $\Gamma \cup \{\neg q_1 \lor \ldots \lor \neg q_m\}$ **nu** este satisfiabilă.

Clauze Horn propoziționale

Fie $q, p_1, \ldots, p_n, q_1, \ldots, q_m$ variabile propoziționale.

- Observăm că $((p_1 \wedge \ldots \wedge p_k) \to q) \sim (\neg p_1 \vee \ldots \vee \neg p_k \vee q)$
- O clauză propozițională definită este o formulă care poate avea una din formele:
  1. q (un fapt în Prolog `q.`)
  2. $\neg p_1 \vee \ldots \vee \neg p_k \vee q$ (o regulă în Prolog `q :- p1,...,pk`)
- Observăm că $\Gamma \models q_1 \wedge \ldots \wedge q_m$ ddacă $\Gamma \cup \{\neg q_1 \vee \ldots \vee \neg q_m\}$ **nu** este satisfiabilă.
- $\neg q_1 \vee \ldots \vee \neg q_m$ se numește clauză scop.
- O clauză Horn propozițională este o clauză propozițională definită sau o clauză scop. O clauză Horn propozițională poate fi definită ca o disjuncție de literali dintre care cel mult unul este variabilă (literal pozitv).

Rezoluția SLD (cazul propozițional)

Fie S o mulțime de clauze definite.

$$\text{SLD} \quad \boxed{\frac{\neg p_1 \vee \cdots \vee \neg q \vee \cdots \vee \neg p_n}{\neg p_1 \vee \cdots \vee \neg q_1 \vee \cdots \vee \neg q_m \vee \cdots \vee \neg p_n}}$$

unde $q \vee \neg q_1 \vee \cdots \vee \neg q_m$ este o clauză definită din S.

Rezoluția SLD

Fie S o mulțime de clauze definite și q o întrebare.

O derivare din S prin rezoluție SLD este o secvență

$$G_0 := \neg q, \quad G_1, \quad \ldots, \quad G_k, \ldots$$

în care G_{i+1} se obține din G_i prin regula SLD.

Dacă există un k cu $G_k = \square$ (clauza vidă), atunci derivarea se numește SLD-respingere.

Rezoluția SLD

Baza de cunoștințe KB:

```
oslo .
windy :- oslo.
norway :- oslo.
cold :- norway.
winter :- cold, windy.
```

Întrebarea:

```
-? winter.
```

Rezoluția SLD

Baza de cunoștințe KB:

```
oslo .
windy :- oslo.
norway :- oslo.
cold :- norway.
winter :- cold, windy.
```

Întrebarea:

```
-? winter.
```

- Formă clauzală:

  $$KB = \{\{oslo\}, \{\neg oslo, windy\}, \{\neg oslo, norway\}, \{\neg norway, cold\}, \{\neg cold, \neg windy, winter\}\}$$

- KB $\vdash$ *winter* dacă și numai dacă $KB \cup \{\neg winter\}$ **nu** este satisfiabilă.

Rezoluția SLD

Baza de cunoștințe KB:

```
oslo .
windy :- oslo.
norway :- oslo.
cold :- norway.
winter :- cold, windy.
```

Întrebarea:

```
-? winter.
```

- Formă clauzală:

  $$KB = \{\{oslo\}, \{\neg oslo, windy\}, \{\neg oslo, norway\}, \{\neg norway, cold\}, \{\neg cold, \neg windy, winter\}\}$$

- KB $\vdash$ *winter* dacă și numai dacă $KB \cup \{\neg winter\}$ **nu** este satisfiabilă.

- Satisfiabilitatea este verificată prin rezoluție

  SLD = Linear resolution with Selected literal for Definite clauses

Clause Horn propoziționale - rezoluția SLD

Exemplu

Demonstrăm KB $\vdash$ *winter* prin rezoluție SLD:

$\{\neg winter\}$

Clause Horn propoziționale - rezoluția SLD

Exemplu

Demonstrăm KB $\vdash$ *winter* prin rezoluție SLD:

$\{\neg winter\}$ $\{\neg cold, \neg windy, winter\}$

$\{\neg cold, \neg windy\}$

Clause Horn propoziționale - rezoluția SLD

Exemplu

Demonstrăm KB $\vdash$ *winter* prin rezoluție SLD:

$\{\neg winter\}$ $\{\neg cold, \neg windy, winter\}$

$\{\neg cold, \neg windy\}$ $\{\neg norway, cold\}$

$\{\neg norway, \neg windy\}$

Clause Horn propoziționale - rezoluția SLD

Exemplu

Demonstrăm KB $\vdash$ *winter* prin rezoluție SLD:

$\{\neg winter\}$ $\{\neg cold, \neg windy, winter\}$

$\{\neg cold, \neg windy\}$ $\{\neg norway, cold\}$

$\{\neg norway, \neg windy\}$ $\{\neg oslo, norway\}$

$\{\neg oslo, \neg windy\}$

Clause Horn propoziționale - rezoluția SLD

Exemplu

Demonstrăm KB $\vdash$ *winter* prin rezoluție SLD:

$\{\neg winter\}$ $\qquad$ $\{\neg cold, \neg windy, winter\}$

$\{\neg cold, \neg windy\}$ $\qquad$ $\{\neg norway, cold\}$

$\{\neg norway, \neg windy\}$ $\qquad$ $\{\neg oslo, norway\}$

$\{\neg oslo, \neg windy\}$ $\qquad$ $\{oslo\}$

$\{\neg windy\}$

Clause Horn propoziționale - rezoluția SLD

Exemplu

Demonstrăm KB $\vdash$ *winter* prin rezoluție SLD:

$\{\neg winter\}$	$\{\neg cold, \neg windy, winter\}$
$\{\neg cold, \neg windy\}$	$\{\neg norway, cold\}$
$\{\neg norway, \neg windy\}$	$\{\neg oslo, norway\}$
$\{\neg oslo, \neg windy\}$	$\{oslo\}$
$\{\neg windy\}$	$\{\neg oslo, windy\}$
$\{\neg oslo\}$	

Clause Horn propoziționale - rezoluția SLD

Exemplu

Demonstrăm KB $\vdash$ *winter* prin rezoluție SLD:

$\{\neg winter\}$	$\{\neg cold, \neg windy, winter\}$
$\{\neg cold, \neg windy\}$	$\{\neg norway, cold\}$
$\{\neg norway, \neg windy\}$	$\{\neg oslo, norway\}$
$\{\neg oslo, \neg windy\}$	$\{oslo\}$
$\{\neg windy\}$	$\{\neg oslo, windy\}$
$\{\neg oslo\}$	$\{oslo\}$
$\square$	

Termeni compuși `f(t1,..., tn)`

- Termenii sunt unitățile de bază prin care Prolog reprezintă datele.
- Sunt de 3 tipuri:
  - Constante: `23, sansa, 'Jon Snow'`
  - Variabile: `X, Stark, _house`
  - Termeni compuși:
    - predicate
    - termeni prin care reprezentăm datele

Exemplu

- `born(john, date(20,3,1977))`
  - `born/2` și `date/3` sunt *functori*
  - `born/2` este un predicat
  - `date/3` definește date compuse

Sintaxa Prolog

Atenție!

- În sintaxa Prolog
  - termenii compuși sunt predicate: `father(eddard, jon_snow)`
  - operatorii sunt funcții: `+`, `*`, `mod`
- Sintaxa Prolog nu face diferență între simboluri de funcții și simboluri de predicate!
- Dar este important când ne uităm la teoria corespunzătoare programului în logică să facem acestă distincție.

Logica de ordinul I - limbaj și formule

Limbaje de ordinul I

Un limbaj $\mathcal{L}$ de ordinul I este format din:

- o mulțime numărabilă de variabile $V = \{x_n \mid n \in \mathbb{N}\}$
- conectorii $\neg, \rightarrow, \wedge, \vee$
- paranteze
- cuantificatorul universal $\forall$ și cuantificatorul existențial $\exists$
- o mulțime $\mathbf{R}$ de simboluri de relații
- o mulțime $\mathbf{F}$ de simboluri de funcții
- o mulțime $\mathbf{C}$ de simboluri de constante
- o funcție aritate $ar : \mathbf{F} \cup \mathbf{R} \rightarrow \mathbb{N}^*$

Logica de ordinul I

- $\mathcal{L}$ este unic determinat de $\tau = (\mathbf{R}, \mathbf{F}, \mathbf{C}, ari)$
- τ se numește signatura (vocabularul, alfabetul) lui $\mathcal{L}$

Exemplu

Un limbaj $\mathcal{L}$ de ordinul I în care:

- $\mathbf{R} = \{P, R\}$
- $\mathbf{F} = \{f\}$
- $\mathbf{C} = \{c\}$
- $ari(P) = 1$, $ari(R) = 2$, $ari(f) = 2$

Logica de ordinul I

Termenii lui $\mathcal{L}$ sunt definiți inductiv astfel:

- orice variabilă este un termen;
- orice simbol de constantă este un termen;
- dacă $f \in \mathbf{F}$, $ar(f) = n$ și $t_1, \ldots, t_n$ sunt termeni, atunci $f(t_1, \ldots, t_n)$ este termen.

Notăm cu $Trm_{\mathcal{L}}$ mulțimea termenilor lui $\mathcal{L}$.

Exemplu

$$c, \quad x_1, \quad f(x_1, c), \quad f(f(x_2, x_2), c)$$

Formulele atomice ale lui $\mathcal{L}$ sunt definite astfel:

- dacă $R \in \mathbf{R}$, $ar(R) = n$ și $t_1, \ldots, t_n$ sunt termeni, atunci $R(t_1, \ldots, t_n)$ este formulă atomică.

Exemplu

$$P(f(x_1, c)), \quad R(c, x_3)$$

Logica de ordinul I

Formulele lui $\mathcal{L}$ sunt definite astfel:

- orice formulă atomică este o formulă
- dacă φ este o formulă, atunci $\neg\varphi$ este o formulă
- dacă φ și ψ sunt formule, atunci $\varphi \vee \psi$, $\varphi \wedge \psi$, $\varphi \rightarrow \psi$ sunt formule
- dacă φ este o formulă și x este o variabilă, atunci $\forall x\, \varphi$, $\exists x\, \varphi$ sunt formule

Logica de ordinul I

Formulele lui $\mathcal{L}$ sunt definite astfel:

- orice formulă atomică este o formulă
- dacă φ este o formulă, atunci $\neg\varphi$ este o formulă
- dacă φ și ψ sunt formule, atunci $\varphi \vee \psi$, $\varphi \wedge \psi$, $\varphi \rightarrow \psi$ sunt formule
- dacă φ este o formulă și x este o variabilă, atunci $\forall x\, \varphi$, $\exists x\, \varphi$ sunt formule

Exemplu

$$P(f(x_1, c)), \quad P(x_1) \vee P(c), \quad \forall x_1\, P(x_1), \quad \forall x_2\, R(x_2, x_1)$$

Logica de ordinul I

Exemplu

Fie limbajul $\mathcal{L}_1$ cu $\mathbf{R} = \{<\}$, $\mathbf{F} = \{s, +\}$, $\mathbf{C} = \{0\}$ și $ari(s) = 1$, $ari(+) = ari(<) = 2$.

Exemplu

Fie limbajul $\mathcal{L}_1$ cu $\mathbf{R} = \{<\}$, $\mathbf{F} = \{s, +\}$, $\mathbf{C} = \{0\}$ și $ari(s) = 1$, $ari(+) = ari(<) = 2$.

Exemple de termeni:

Logica de ordinul I

Exemplu

Fie limbajul $\mathcal{L}_1$ cu $\mathbf{R} = \{<\}$, $\mathbf{F} = \{s, +\}$, $\mathbf{C} = \{0\}$ și $ari(s) = 1$, $ari(+) = ari(<) = 2$.

Exemple de termeni:

0, x, $s(0)$, $s(s(0))$, $s(x)$, $s(s(x))$, $\ldots$,

Exemplu

Fie limbajul $\mathcal{L}_1$ cu $\mathbf{R} = \{<\}$, $\mathbf{F} = \{s, +\}$, $\mathbf{C} = \{0\}$ și $ari(s) = 1$, $ari(+) = ari(<) = 2$.

Exemple de termeni:

0, x, $s(0)$, $s(s(0))$, $s(x)$, $s(s(x))$, $\ldots$,
$+(0,0)$, $+(s(s(0)), +(0, s(0)))$, $+(x, s(0))$, $+(x, s(x))$, $\ldots$,

Exemplu

Fie limbajul $\mathcal{L}_1$ cu $\mathbf{R} = \{<\}$, $\mathbf{F} = \{s, +\}$, $\mathbf{C} = \{0\}$ și $ari(s) = 1$, $ari(+) = ari(<) = 2$.

Exemple de termeni:

0, x, $s(0)$, $s(s(0))$, $s(x)$, $s(s(x))$, $\ldots$,
$+(0, 0)$, $+(s(s(0)), +(0, s(0)))$, $+(x, s(0))$, $+(x, s(x))$, $\ldots$,

Exemple de formule atomice:

$<(0, 0)$, $<(x, 0)$, $<(s(s(x)), s(0))$, $\ldots$

Logica de ordinul I

Exemplu

Fie limbajul $\mathcal{L}_1$ cu $\mathbf{R} = \{<\}$, $\mathbf{F} = \{s, +\}$, $\mathbf{C} = \{0\}$ și
$ari(s) = 1$, $ari(+) = ari(<) = 2$.

Exemple de termeni:

0, x, $s(0)$, $s(s(0))$, $s(x)$, $s(s(x))$, ...,
$+(0,0)$, $+(s(s(0)), +(0, s(0)))$, $+(x, s(0))$, $+(x, s(x))$, ...,

Exemple de formule atomice:

$< (0,0)$, $< (x,0)$, $< (s(s(x)), s(0))$, ...

Exemple de formule:

$\forall x \forall y < (x, +(x,y))$
$\forall x < (x, s(x))$

Pentru a stabili dacă o formulă este adevărată, trebuie să definim semantica logicii de ordinul I!

Logica de ordinul I - sintaxa

Limbaj de ordinul I $\mathcal{L}$

- unic determinat de $\tau = (\mathbf{R}, \mathbf{F}, \mathbf{C}, ari)$

Termenii lui $\mathcal{L}$, notați $Trm_{\mathcal{L}}$, sunt definiți inductiv astfel:

- orice variabilă este un termen;
- orice simbol de constantă este un termen;
- dacă $f \in \mathbf{F}$, $ar(f) = n$ și $t_1, \ldots, t_n$ sunt termeni, atunci $f(t_1, \ldots, t_n)$ este termen.

Formulele atomice ale lui $\mathcal{L}$ sunt definite astfel:

- dacă $R \in \mathbf{R}$, $ar(R) = n$ și $t_1, \ldots, t_n$ sunt termeni, atunci $R(t_1, \ldots, t_n)$ este formulă atomică.

Formulele lui $\mathcal{L}$ sunt definite astfel:

- orice formulă atomică este o formulă
- dacă φ este o formulă, atunci $\neg\varphi$ este o formulă
- dacă φ și ψ sunt formule, atunci $\varphi \vee \psi$, $\varphi \wedge \psi$, $\varphi \rightarrow \psi$ sunt formule
- dacă φ este o formulă și x este o variabilă, atunci $\forall x\, \varphi$, $\exists x\, \varphi$ sunt formule

Date compuse în Prolog

Tipuri de date definite recursiv

Tipuri de date definite recursiv

- Cum definim arborii binari în Prolog?

Tipuri de date definite recursiv

- Cum definim arborii binari în Prolog? Soluție posibilă:

Tipuri de date definite recursiv

- Cum definim arborii binari în Prolog? Soluție posibilă:
  - `void` este arbore

Tipuri de date definite recursiv

- Cum definim arborii binari în Prolog? Soluție posibilă:
  - `void` este arbore
  - `tree(X,A1,A2)` este arbore, unde `X` este un element, iar `A1` și `A2` sunt arbori

Tipuri de date definite recursiv

- Cum definim arborii binari în Prolog? Soluție posibilă:
  - `void` este arbore
  - `tree(X,A1,A2)` este arbore, unde `X` este un element, iar `A1` și `A2` sunt arbori
- Cum arată un arbore?

tree(a, tree(b, tree(d, void, void), void), tree(c, void, tree(e, void, void)))

Tipuri de date definite recursiv

- Cum definim arborii binari în Prolog? Soluție posibilă:
  - `void` este arbore
  - `tree(X,A1,A2)` este arbore, unde `X` este un element, iar `A1` și `A2` sunt arbori
- Cum arată un arbore?

  tree(a, tree(b, tree(d, void, void), void), tree(c, void, tree(e, void, void)))

- `tree(X,A1,A2)` este un termen compus dar nu este un predicat; el corespunde unui termen din logica de ordinul I.

Termeni compuși în Prolog

Termeni compuși în Prolog

- Cum dăm un "nume" arborelui de mai sus?

Termeni compuși în Prolog

- Cum dăm un "nume" arborelui de mai sus? Definim un predicat:

```
def(arb, tree(a, tree(b,
                      tree(d,void,void),
                      void),
                 tree(c, void,
                      tree(e,void,void)))).
?- def(arb,T).
T = tree(a, tree(b, tree(d, void, void), void), tree(c,
void, tree(e, void, void))).
```

Termeni compuși în Prolog

- Cum dăm un "nume" arborelui de mai sus? Definim un predicat:

```
def(arb, tree(a, tree(b,
                      tree(d,void,void),
                      void),
                 tree(c, void,
                      tree(e,void,void)))).
?- def(arb,T).
T = tree(a, tree(b, tree(d, void, void), void), tree(c,
void, tree(e, void, void))).
```

Deoarece în Prolog nu avem declarații explicite de date, pentru a defini arborii vom scrie un predicat care este adevărat atunci când argumentul său este un arbore.

Predicate și date compuse în Prolog

Scrieți un predicat care verifică că un termen este arbore binar.

Predicate și date compuse în Prolog

Scrieți un predicat care verifică că un termen este arbore binar.

```
binary_tree(void).
binary_tree(tree(Element,Left,Right)) :-
                              binary_tree(Left),
                              binary_tree(Right).
```

Predicate și date compuse în Prolog

Scrieți un predicat care verifică că un termen este arbore binar.

```
binary_tree(void).
binary_tree(tree(Element,Left,Right)) :-
                              binary_tree(Left),
                              binary_tree(Right).
```

Eventual putem defini și un predicat pentru elemente:

```
binary_tree(void).
binary_tree(tree(Element,Left,Right)) :-
                              binary_tree(Left),
                              binary_tree(Right),
                              element_binary_tree(Element).

element_binary_tree(X):- integer(X). /* de exemplu */
```

Predicate și date compuse în Prolog

Scrieți un predicat care verifică că un termen este arbore binar.

```
binary_tree(void).
binary_tree(tree(Element,Left,Right)) :-
                             binary_tree(Left),
                             binary_tree(Right).
```

Eventual putem defini și un predicat pentru elemente:

```
binary_tree(void).
binary_tree(tree(Element,Left,Right)) :-
                             binary_tree(Left),
                             binary_tree(Right),
                             element_binary_tree(Element).

element_binary_tree(X):- integer(X). /* de exemplu */

test:- def(arb,T), binary_tree(T).
```

Arbori binari în Prolog

Exercițiu

Scrieți un predicat care verifică dacă un element aparține unui arbore.

Arbori binari în Prolog

Exercițiu

Scrieți un predicat care verifică dacă un element aparține unui arbore.

```
tree_member(X,tree(X,Left,Right)).

tree_member(X,tree(_,Left,Right)) :- tree_member(X,Left).

tree_member(X,tree(_,Left,Right)) :- tree_member(X,Right).
```

Cum găsește Prolog răspunsul

```
tree\_member(X,tree(X,Left,Right)).
tree\_member(X,tree(\_,Left,Right)) :- tree\_member(X,Left).
tree\_member(X,tree(\_,Left,Right)) :- tree\_member(X,Right).

?- tree_member(a, tree(a,void,void)).
true .

?- def(arb,T), tree_member(b, T).
T = tree(a, tree(b, tree(d, void, void), void),
            tree(c, void, tree(e, void, void))) .

4 ?- def(arb,T), tree_member(v, T).
false.
```

Pentru a răspunde la întrebare se caută o potrivire (unificator) pentru între scop, parcurgând baza de cunoștințe. Raspunsul este true/false sau substituția care realizează unificarea.

Vom prezenta algoritmul de unificare!

Termeni, substituții, unificatori

Termeni

Alfabet:

- $\mathcal{F}$ o multime de simboluri de functii de aritate cunoscuta
- $\mathcal{V}$ o multime numarabila de variabile
- $\mathcal{F}$ si $\mathcal{V}$ sunt disjuncte

Termeni

Alfabet:

- $\mathcal{F}$ o multime de simboluri de functii de aritate cunoscuta
- $\mathcal{V}$ o multime numarabila de variabile
- $\mathcal{F}$ si $\mathcal{V}$ sunt disjuncte

Termeni peste $\mathcal{F}$ si $\mathcal{V}$:

$$t \ ::= \ x \mid f(t_1, \ldots, t_n)$$

unde

- $n \geq 0$
- x este o variabila
- f este un simbol de functie de aritate n

Termeni

Notatii:

- constante: simboluri de functii de aritate 0
- $x, y, z, \ldots$ pentru variabile
- $a, b, c, \ldots$ pentru constante
- $f, g, h, \ldots$ pentru simboluri de functii arbitrare
- $s, t, u, \ldots$ pentru termeni
- $var(t)$ multimea variabilelor care apa in t
- ecuatii $s \doteq t$ pentru o pereche de termeni
- $Trm_{\mathcal{F},\mathcal{V}}$ multimea termenilor peste $\mathcal{F}$ si $\mathcal{V}$

Termeni

Exemplu

- $f(x, g(x, a), y)$ este un termen, unde f are aritate 3, g are aritate 2, a este o constanta
- $var(f(x, g(x, a), y)) = \{x, y\}$

Substituții

Definiție

O subtituție σ este o funcție (parțială) de la variabile la termeni, adică

$$\sigma : \mathcal{V} \to Trm_{\mathcal{F},\mathcal{V}}$$

Exemplu

În notația uzuală, $\sigma = \{x/a,\ y/g(w), z/b\}$. Substitutia σ este identitate pe restul variabilelor.

Notatie alternativa $\sigma = \{x \mapsto a, y \mapsto g(w), z \mapsto b\}$.

Substituții

- Substituțiile sunt o modalitate de a înlocui variabilele cu alți termeni.
- Substituțiile se aplică simultan pe toate variabilele.

Aplicarea unei substitutii σ unui termen t:

$$\sigma(t) = \begin{cases} \sigma(x), \text{ daca } t = x \\ f(\sigma(t_1), \ldots, \sigma(t_n)), \text{ daca } t = f(t_1, \ldots, t_n) \end{cases} .$$

Substituții

- Substituțiile sunt o modalitate de a înlocui variabilele cu alți termeni.
- Substituțiile se aplică simultan pe toate variabilele.

Aplicarea unei substitutii σ unui termen t:

$$\sigma(t) = \begin{cases} \sigma(x), \text{ daca } t = x \\ f(\sigma(t_1), \ldots, \sigma(t_n)), \text{ daca } t = f(t_1, \ldots, t_n) \end{cases}.$$

Exemplu

- $\sigma = \{x \mapsto f(x, y), y \mapsto g(a)\}$
- $t = f(x, g(f(x, f(y, z))))$
- $\sigma(t) = f(f(x, y), g(f(f(x, y), f(g(a), z))))$

Substituții

Două substituții σ_1 și σ_2 se pot compune

$$\sigma_1 ; \sigma_2$$

(aplicăm întâi σ_1, apoi σ_2).

Substituții

Două substituții σ_1 și σ_2 se pot compune

$$\sigma_1 ; \sigma_2$$

(aplicăm întâi σ_1, apoi σ_2).

Exemplu

- $t = h(u, v, x, y, z)$

Substituții

Două substituții σ_1 și σ_2 se pot compune

$$\sigma_1 ; \sigma_2$$

(aplicăm întâi σ_1, apoi σ_2).

Exemplu

- $t = h(u, v, x, y, z)$
- $\tau = \{x \mapsto f(y),\ y \mapsto f(a),\ z \mapsto u\}$
- $\sigma = \{y \mapsto g(a),\ u \mapsto z,\ v \mapsto f(f(a))\}$

Substituții

Două substituții σ_1 și σ_2 se pot compune

$$\sigma_1 ; \sigma_2$$

(aplicăm întâi σ_1, apoi σ_2).

Exemplu

- $t = h(u, v, x, y, z)$
- $\tau = \{x \mapsto f(y),\ y \mapsto f(a),\ z \mapsto u\}$
- $\sigma = \{y \mapsto g(a),\ u \mapsto z,\ v \mapsto f(f(a))\}$
- $(\tau ; \sigma)(t) = \sigma(\tau(t)) = \sigma(h(u, v, f(y), f(a), u)) =$
  $= h(z, f(f(a)), f(g(a)), f(a), z)$

Substituții

Două substituții σ_1 și σ_2 se pot compune

$$\sigma_1 ; \sigma_2$$

(aplicăm întâi σ_1, apoi σ_2).

Exemplu

- $t = h(u, v, x, y, z)$
- $\tau = \{x \mapsto f(y),\ y \mapsto f(a),\ z \mapsto u\}$
- $\sigma = \{y \mapsto g(a),\ u \mapsto z,\ v \mapsto f(f(a))\}$
- $(\tau;\sigma)(t) = \sigma(\tau(t)) = \sigma(h(u, v, f(y), f(a), u)) =$
  $= h(z, f(f(a)), f(g(a)), f(a), z)$
- $(\sigma;\tau)(t) = \tau(\sigma(t)) = \tau(h(z, f(f(a)), x, g(a), z))$
  $= h(u, f(f(a)), f(y), g(a), u)$

Unificare

- Doi termeni t_1 și t_2 se unifică dacă există o substituție σ astfel încât
$$\sigma(t_1) = \sigma(t_2).$$
- În acest caz, σ se numește unificatorul termenilor t_1 și t_2.

Unificare

- Doi termeni t_1 și t_2 se unifică dacă există o substituție σ astfel încât
$$\sigma(t_1) = \sigma(t_2).$$
- În acest caz, σ se numește unificatorul termenilor t_1 și t_2.
- În programarea logică, unificatorii sunt ingredientele de bază în execuția unui program.

Unificare

- Doi termeni t_1 și t_2 se unifică dacă există o substituție σ astfel încât
$$\sigma(t_1) = \sigma(t_2).$$
- În acest caz, σ se numește unificatorul termenilor t_1 și t_2.
- În programarea logică, unificatorii sunt ingredientele de bază în execuția unui program.
- Un unificator σ pentru t_1 și t_2 este un cel mai general unificator (cgu,mgu) dacă pentru orice alt unificator σ' pentru t_1 și t_2, există o substituție τ astfel încât
$$\sigma' = \sigma; \tau.$$

Unificator

Exemplu

- $t = x + (y * y) = +(x, *(y, y))$
- $t' = x + (y * x) = +(x, *(y, x))$

Unificator

Exemplu

- $t = x + (y * y) = +(x, *(y, y))$
- $t' = x + (y * x) = +(x, *(y, x))$
- $\sigma = \{x/y, y/y\}$
  - $\sigma(t) = y + (y * y)$
  - $\sigma(t') = y + (y * y)$
  - σ este cgu

Unificator

Exemplu

- $t = x + (y * y) = +(x, *(y, y))$
- $t' = x + (y * x) = +(x, *(y, x))$
- $\sigma = \{x/y, y/y\}$
  - $\sigma(t) = y + (y * y)$
  - $\sigma(t') = y + (y * y)$
  - σ este cgu
- $\sigma' = \{x/0, y/0\}$
  - $\sigma'(t) = 0 + (0 * 0)$
  - $\sigma'(t') = 0 + (0 * 0)$

Unificator

Exemplu

- $t = x + (y * y) = +(x, *(y, y))$
- $t' = x + (y * x) = +(x, *(y, x))$
- $\sigma = \{x/y, y/y\}$
  - $\sigma(t) = y + (y * y)$
  - $\sigma(t') = y + (y * y)$
  - σ este cgu
- $\sigma' = \{x/0, y/0\}$
  - $\sigma'(t) = 0 + (0 * 0)$
  - $\sigma'(t') = 0 + (0 * 0)$
  - $\sigma' = \sigma; \{y/0\}$

Unificator

Exemplu

- $t = x + (y * y) = +(x, *(y, y))$
- $t' = x + (y * x) = +(x, *(y, x))$
- $\sigma = \{x/y, y/y\}$
  - $\sigma(t) = y + (y * y)$
  - $\sigma(t') = y + (y * y)$
  - σ este cgu
- $\sigma' = \{x/0, y/0\}$
  - $\sigma'(t) = 0 + (0 * 0)$
  - $\sigma'(t') = 0 + (0 * 0)$
  - $\sigma' = \sigma; \{y/0\}$
  - σ' este unificator, dar nu este cgu

Algoritmul de unificare

Algoritmul de unificare

- Pentru o mulțime finită de perechi $\{(t_1, t'_1), \ldots, (t_n, t'_n)\}$, algoritmul de unificare stabilește dacă există un unificator σ astfel încât

$$\sigma(t_1) = \sigma(t'_1), \ldots, \sigma(t_n) = \sigma(t'_n)$$

  În plus, unificatorul determinat de algoritm este cgu.

- Algoritmul lucrează cu două liste:
  - Lista soluție: S
  - Lista de rezolvat: R

Algoritmul de unificare

- Pentru o mulțime finită de perechi $\{(t_1, t_1'), \ldots, (t_n, t_n')\}$, algoritmul de unificare stabilește dacă există un unificator σ astfel încât

$$\sigma(t_1) = \sigma(t_1'), \ldots, \sigma(t_n) = \sigma(t_n')$$

  În plus, unificatorul determinat de algoritm este cgu.

- Algoritmul lucrează cu două liste:
  - Lista soluție: S
  - Lista de rezolvat: R

- Inițial:
  - Lista soluție: $S = \emptyset$
  - Lista de rezolvat: $R = \{t_1 \doteq t_1', \ldots, t_n \doteq t_n'\}$

- $\doteq$ este un simbol nou, folosit pentru a desemna perechile de termeni care vor fi unificați.

Algoritmul de unificare

- Pentru o mulțime finită de perechi $\{t_1 \doteq t'_1, \ldots, t_n \doteq t'_n\}$, algoritmul de unificare stabilește dacă există un unificator σ astfel încât

$$\sigma(t_1) = \sigma(t'_1), \ldots, \sigma(t_n) = \sigma(t'_n)$$

- Pentru o mulțime finită de termeni $\{t_1, \ldots, t_n\}$, $n \geq 2$, găsim un unificator aplicând algoritmul de unificare pentru lista de perechi $\{t_1 \doteq t_2, \ldots, t_{n-1} \doteq t_n\}$.

- Pentru o țintă în Prolog

  ?- p$(t_1, \ldots, t_n)$ = p$(t'_1, \ldots, t'_n)$

  algoritmul unifică lista de perechi $\{t_1 \doteq t'_1, \ldots, t_n \doteq t'_n\}$.

Algoritmul de unificare

Algoritmul constă în aplicarea regulilor de mai jos:

Algoritmul de unificare

Algoritmul constă în aplicarea regulilor de mai jos:

- SCOATE
  - orice ecuație de forma $t \doteq t$ din R este eliminată.

Algoritmul de unificare

Algoritmul constă în aplicarea regulilor de mai jos:

- SCOATE
  - orice ecuație de forma $t \doteq t$ din R este eliminată.
- DESCOMPUNE
  - orice ecuație de forma $f(t_1, \ldots, t_n) \doteq f(t'_1, \ldots, t'_n)$ din R este înlocuită cu ecuațiile $t_1 \doteq t'_1, \ldots, t_n \doteq t'_n$.

Algoritmul de unificare

Algoritmul constă în aplicarea regulilor de mai jos:

- SCOATE
  - orice ecuație de forma $t \doteq t$ din R este eliminată.
- DESCOMPUNE
  - orice ecuație de forma $f(t_1, \ldots, t_n) \doteq f(t'_1, \ldots, t'_n)$ din R este înlocuită cu ecuațiile $t_1 \doteq t'_1, \ldots, t_n \doteq t'_n$.
- REZOLVĂ
  - orice ecuație de forma $x \doteq t$ sau $t \doteq x$ din R, unde variabila x nu apare în termenul t, este mutată sub forma $x \doteq t$ în S.
    În toate celelalte ecuații (din R și S), x este înlocuit cu t.

Algoritmul de unificare

Algoritmul se termină normal dacă $R = \emptyset$. În acest caz, S conține cgu.

Algoritmul de unificare

Algoritmul se termină normal dacă $R = \emptyset$. În acest caz, S conține cgu.

Algoritmul este oprit cu concluzia inexistenței unui cgu dacă:

1. În R există o ecuație de forma
$$f(t_1, \ldots, t_n) \doteq g(t'_1, \ldots, t'_k) \text{ cu } f \neq g.$$
2. În R există o ecuație de forma $x \doteq t$ sau $t \doteq x$ și variabila x apare în termenul t.

Algoritmul de unificare - schemă

	Lista soluție S	Lista de rezolvat R
Inițial	$\emptyset$	$t_1 \doteq t'_1, \ldots, t_n \doteq t'_n$
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
	$x \doteq t, S[x/t]$	$R'[x/t]$
Final	S	$\emptyset$

$S[x/t]$: în toate ecuațiile din S, x este înlocuit cu t

Exemplu

Exemplu

- Ecuațiile $\{g(y) \doteq x,\ f(x, h(x), y) \doteq f(g(z), w, z)\}$ au cgu?

Exemplu

Exemplu

- Ecuațiile $\{g(y) \doteq x,\ f(x, h(x), y) \doteq f(g(z), w, z)\}$ au cgu?

S	R	
$\emptyset$	$g(y) \doteq x,\ f(x, h(x), y) \doteq f(g(z), w, z)$	

Exemplu

Exemplu

- Ecuațiile $\{g(y) \doteq x,\ f(x, h(x), y) \doteq f(g(z), w, z)\}$ au cgu?

S	R	
$\emptyset$	$g(y) \doteq x,\ f(x, h(x), y) \doteq f(g(z), w, z)$	REZOLVĂ

Exemplu

Exemplu

- Ecuațiile $\{g(y) \doteq x,\ f(x, h(x), y) \doteq f(g(z), w, z)\}$ au cgu?

S	R	
$\emptyset$	$g(y) \doteq x,\ f(x, h(x), y) \doteq f(g(z), w, z)$	REZOLVĂ
$x \doteq g(y)$	$f(g(y), h(g(y)), y) \doteq f(g(z), w, z)$	

Exemplu

Exemplu

- Ecuațiile $\{g(y) \doteq x,\ f(x, h(x), y) \doteq f(g(z), w, z)\}$ au cgu?

S	R	
$\emptyset$	$g(y) \doteq x,\ f(x, h(x), y) \doteq f(g(z), w, z)$	REZOLVĂ
$x \doteq g(y)$	$f(g(y), h(g(y)), y) \doteq f(g(z), w, z)$	DESCOMPUNE

Exemplu

Exemplu

- Ecuațiile $\{g(y) \doteq x,\ f(x, h(x), y) \doteq f(g(z), w, z)\}$ au cgu?

S	R	
$\emptyset$	$g(y) \doteq x,\ f(x, h(x), y) \doteq f(g(z), w, z)$	REZOLVĂ
$x \doteq g(y)$	$f(g(y), h(g(y)), y) \doteq f(g(z), w, z)$	DESCOMPUNE
$x \doteq g(y)$	$g(y) \doteq g(z),\ h(g(y)) \doteq w,\ y \doteq z$	

Exemplu

Exemplu

- Ecuațiile $\{g(y) \doteq x,\ f(x, h(x), y) \doteq f(g(z), w, z)\}$ au cgu?

S	R	
$\emptyset$	$g(y) \doteq x,\ f(x, h(x), y) \doteq f(g(z), w, z)$	REZOLVĂ
$x \doteq g(y)$	$f(g(y), h(g(y)), y) \doteq f(g(z), w, z)$	DESCOMPUNE
$x \doteq g(y)$	$g(y) \doteq g(z),\ h(g(y)) \doteq w,\ y \doteq z$	REZOLVĂ

Exemplu

Exemplu

- Ecuațiile $\{g(y) \doteq x,\ f(x, h(x), y) \doteq f(g(z), w, z)\}$ au cgu?

S	R	
$\emptyset$	$g(y) \doteq x,\ f(x, h(x), y) \doteq f(g(z), w, z)$	REZOLVĂ
$x \doteq g(y)$	$f(g(y), h(g(y)), y) \doteq f(g(z), w, z)$	DESCOMPUNE
$x \doteq g(y)$	$g(y) \doteq g(z),\ h(g(y)) \doteq w,\ y \doteq z$	REZOLVĂ
$w \doteq h(g(y)),$ $x \doteq g(y)$	$g(y) \doteq g(z),\ y \doteq z$	REZOLVĂ

Exemplu

Exemplu

- Ecuațiile $\{g(y) \doteq x,\ f(x, h(x), y) \doteq f(g(z), w, z)\}$ au cgu?

S	R	
$\emptyset$	$g(y) \doteq x,\ f(x, h(x), y) \doteq f(g(z), w, z)$	REZOLVĂ
$x \doteq g(y)$	$f(g(y), h(g(y)), y) \doteq f(g(z), w, z)$	DESCOMPUNE
$x \doteq g(y)$	$g(y) \doteq g(z),\ h(g(y)) \doteq w,\ y \doteq z$	REZOLVĂ
$w \doteq h(g(y))$, $x \doteq g(y)$	$g(y) \doteq g(z),\ y \doteq z$	REZOLVĂ
$y \doteq z, x \doteq g(z)$, $w \doteq h(g(z))$	$g(z) \doteq g(z)$	SCOATE

Exemplu

Exemplu

- Ecuațiile $\{g(y) \doteq x,\ f(x, h(x), y) \doteq f(g(z), w, z)\}$ au cgu?

S	R	
$\emptyset$	$g(y) \doteq x,\ f(x, h(x), y) \doteq f(g(z), w, z)$	REZOLVĂ
$x \doteq g(y)$	$f(g(y), h(g(y)), y) \doteq f(g(z), w, z)$	DESCOMPUNE
$x \doteq g(y)$	$g(y) \doteq g(z),\ h(g(y)) \doteq w,\ y \doteq z$	REZOLVĂ
$w \doteq h(g(y))$, $x \doteq g(y)$	$g(y) \doteq g(z),\ y \doteq z$	REZOLVĂ
$y \doteq z, x \doteq g(z)$, $w \doteq h(g(z))$	$g(z) \doteq g(z)$	SCOATE
$y \doteq z, x \doteq g(z)$, $w \doteq h(g(z))$	$\emptyset$	

- $\sigma = \{y \mapsto z,\ x \mapsto g(z),\ w \mapsto h(g(z))\}$ este cgu.

Exemplu

Exemplu

- Ecuațiile $\{g(y) \doteq x,\ f(x, h(y), y) \doteq f(g(z), b, z)\}$ au cgu?

Exemplu

Exemplu

- Ecuațiile $\{g(y) \doteq x,\ f(x, h(y), y) \doteq f(g(z), b, z)\}$ au cgu?

S	R	
$\emptyset$	$g(y) \doteq x,\ f(x, h(y), y) \doteq f(g(z), b, z)$	REZOLVĂ
$x \doteq g(y)$	$f(g(y), h(y), y) \doteq f(g(z), b, z)$	DESCOMPUNE
$x \doteq g(y)$	$g(y) \doteq g(z),\ h(y) \doteq b,\ y \doteq z$	- EȘEC -

Exemplu

Exemplu

- Ecuațiile $\{g(y) \doteq x,\ f(x, h(y), y) \doteq f(g(z), b, z)\}$ au cgu?

S	R	
$\emptyset$	$g(y) \doteq x,\ f(x, h(y), y) \doteq f(g(z), b, z)$	REZOLVĂ
$x \doteq g(y)$	$f(g(y), h(y), y) \doteq f(g(z), b, z)$	DESCOMPUNE
$x \doteq g(y)$	$g(y) \doteq g(z),\ h(y) \doteq b,\ y \doteq z$	- EȘEC -

- *h* și *b* sunt simboluri de operații diferite!
- Nu există unificator pentru acesti termeni.

Exemplu

Exemplu

- Ecuațiile $\{g(y) \doteq x,\ f(x, h(x), y) \doteq f(y, w, z)\}$ au cgu?

Exemplu

Exemplu

- Ecuațiile $\{g(y) \doteq x,\ f(x, h(x), y) \doteq f(y, w, z)\}$ au cgu?

S	R	
$\emptyset$	$g(y) \doteq x,\ f(x, h(x), y) \doteq f(y, w, z)$	REZOLVĂ
$x \doteq g(y)$	$f(g(y), h(g(y)), y) \doteq f(y, w, z)$	DESCOMPUNE
$x \doteq g(y)$	$g(y) \doteq y,\ h(g(y)) \doteq w,\ y \doteq z$	- EȘEC -

Exemplu

Exemplu

- Ecuațiile $\{g(y) \doteq x,\ f(x, h(x), y) \doteq f(y, w, z)\}$ au cgu?

S	R	
$\emptyset$	$g(y) \doteq x,\ f(x, h(x), y) \doteq f(y, w, z)$	REZOLVĂ
$x \doteq g(y)$	$f(g(y), h(g(y)), y) \doteq f(y, w, z)$	DESCOMPUNE
$x \doteq g(y)$	$g(y) \doteq y,\ h(g(y)) \doteq w,\ y \doteq z$	- EȘEC -

- În ecuația $g(y) \doteq y$, variabila y apare în termenul $g(y)$.
- Nu există unificator pentru aceste ecuații.

Terminarea algoritmului

Propoziție

Algoritmul de unificare se termină.

Terminarea algoritmului

Propoziție

Algoritmul de unificare se termină.

Demonstrație

- Notăm cu
  - N_1: numărul variabilelor care apar în R
  - N_2: numărul aparițiilor simbolurilor care apar în R
- Este suficient să arătăm că perechea (N_1, N_2) descrește strict în ordine lexicografică la execuția unui pas al algoritmului:

  dacă la execuția unui pas (N_1, N_2) se schimbă în (N_1', N_2'), atunci
  $$(N_1, N_2) \geq_{lex} (N_1', N_2')$$

Demonstrație (cont.)

Fiecare regulă a algoritmului modifică N_1 și N_2 astfel:

	N_1	N_2
SCOATE	$\geq$	$>$
DESCOMPUNE	$=$	$>$
REZOLVĂ	$>$	

- N_1: numărul variabilelor care apar în R
- N_2: numărul aparițiilor simbolurilor care apar în R

$\square$

Unificare în Prolog

- Ce se întâmplă dacă încercăm să unificăm X cu ceva care conține X?
  Exemplu: `?- X = f(X).`

- Conform teoriei, acești termeni nu se pot unifica.

- Totuși, multe implementări ale Prolog-ului sar peste această verificare din motive de eficiență.

```
?- X = f(X).
X = f(X).
```

Unificare în Prolog

- Ce se întâmplă dacă încercăm să unificăm X cu ceva care conține X? Exemplu: `?- X = f(X).`
- Conform teoriei, acești termeni nu se pot unifica.
- Totuși, multe implementări ale Prolog-ului sar peste această verificare din motive de eficiență.

```
?- X = f(X).
X = f(X).
```

- Putem folosi `unify_with_occurs_check/2`

```
?- unify_with_occurs_check(X,f(X)).
false.
```

Corectitudinea algoritmului (opțional)

Corectitudinea algoritmului

Lema 1

Mulțimea unificatorilor pentru ecuațiile din $R \cup S$ nu se modifică prin aplicarea celor trei reguli ale algoritmului de unificare.

Corectitudinea algoritmului

Lema 1

Mulțimea unificatorilor pentru ecuațiile din $R \cup S$ nu se modifică prin aplicarea celor trei reguli ale algoritmului de unificare.

Demonstrație

Analizăm fiecare regulă:

- SCOATE: evident

Corectitudinea algoritmului

Lema 1

Mulțimea unificatorilor pentru ecuațiile din $R \cup S$ nu se modifică prin aplicarea celor trei reguli ale algoritmului de unificare.

Demonstrație

Analizăm fiecare regulă:

- SCOATE: evident
- DESCOMPUNE: Trebuie să arătăm că

$$\begin{array}{ccc} \nu \text{ unificator pt.} & \Leftrightarrow & \nu \text{ unificator pt.} \\ f(t_1, \ldots, t_n) \doteq f(t'_1, \ldots, t'_n) & & t_i \doteq t'_i, \text{ or. } i = 1, \ldots, n. \end{array}$$

Corectitudinea algoritmului

Lema 1

Mulțimea unificatorilor pentru ecuațiile din $R \cup S$ nu se modifică prin aplicarea celor trei reguli ale algoritmului de unificare.

Demonstrație

Analizăm fiecare regulă:

- SCOATE: evident
- DESCOMPUNE: Trebuie să arătăm că

$$\begin{array}{ccc} \nu \text{ unificator pt.} & \Leftrightarrow & \nu \text{ unificator pt.} \\ f(t_1, \ldots, t_n) \doteq f(t'_1, \ldots, t'_n) & & t_i \doteq t'_i, \text{ or. } i = 1, \ldots, n. \end{array}$$

$$\begin{aligned} & \nu \text{ unif. pt. } f(t_1, \ldots, t_n) \doteq f(t'_1, \ldots, t'_n) \\ & \Leftrightarrow \nu(f(t_1, \ldots, t_n)) = \nu(f(t'_1, \ldots, t'_n)) \\ & \Leftrightarrow f(\nu(t_1), \ldots, \nu(t_n)) = f(\nu(t'_1), \ldots, \nu(t'_n)) \\ & \Leftrightarrow \nu(t_i) = \nu(t'_i), \text{or. } i = 1, \ldots, n \\ & \Leftrightarrow \nu \text{ unificator pt. } t_i \doteq t'_i, \text{or. } i = 1, \ldots, n \end{aligned}$$

Demonstrație (cont.)

- REZOLVĂ:
  - Se observă că orice unificator ν pentru ecuațiile din $R \cup S$, atât înainte cât și după aplicarea regulii REZOLVĂ, trebuie să satisfacă:

$$\nu(x) = \nu(t).$$

  - Dacă μ este unificator pentru $x \doteq t$ observăm că:

$$(x \leftarrow t); \mu = \mu$$

    unde $(x \leftarrow t)(x) = t$ și $(x \leftarrow t)(y) = y$ pentru orice $y \neq x \in V$.
    - $((x \leftarrow t); \mu)(x) = \mu(t) = \mu(x)$
    - $((x \leftarrow t); \mu)(y) = \mu(y)$, pentru orice $y \neq x$
  - Deci,

μ este un unificator pentru ecuațiile din $R \cup S$ înainte de REZOLVĂ

$$\Leftrightarrow$$

μ este un unificator pentru ecuațiile din $R \cup S$ după REZOLVĂ $\square$

Corectitudinea algoritmului

- Pres. că algoritmul de unificare se termină cu $R = \emptyset$.
- Fie $x_i \doteq t_i$, $i = 1, \ldots, k$, ecuațiile din S.
- Variabilele care apar în partea stângă a ecuațiilor din S sunt distincte două câte două și nu mai apar în termenii $t1$, ..., t_k.
- Definim substituția:

$$\nu(x_i) = t_i \text{ pentru orice } i = 1, \ldots, k.$$

- Observăm că $\nu(t_i) = t_i = \nu(x_i)$ oricare $i = 1, \ldots, k$, deci ν este un unificator pentru $R \cup S$.

Corectitudinea algoritmului

Lema 2

ν definit mai sus cf. algoritmului de unificare este cgu pentru $R \cup S$.

Corectitudinea algoritmului

Lema 2

ν definit mai sus cf. algoritmului de unificare este cgu pentru $R \cup S$.

Demonstrație

La ultimul pas $R = \emptyset$ și $\nu(x_i) = t_i$ oricare $i = 1, \ldots, k$

- Fie μ un alt unificator pentru S. Avem
  - $\mu(\nu(x_i)) = \mu(t_i) = \mu(x_i)$, or. $i = 1, \ldots, k$,
  - $\mu(\nu(y)) = \mu(y)$, or. $y \neq x$.

  Deci $\nu; \mu = \mu$. În concluzie, ν este cgu deoarece oricare alt unificator se poate scrie ca o compunere a lui ν cu o substituție. $\square$

- Din Lema 1 rezultă că ν este unificator pentru problema inițială $\{u_1 \doteq u_2, \ldots, u_{n-1} \doteq u_n\}$, deci

$$\nu(u_1) = \cdots = \nu(u_n).$$

Complexitatea algoritmului

- Problema de unificare

  $R = \{x_1 \doteq f(x_0, x_0), x_2 \doteq f(x_1, x_1), \ldots, x_n \doteq f(x_{n-1}, x_{n-1})\}$

  are cgu $S = \{x_1 \leftarrow f(x_0, x_0), x_2 \leftarrow f(f(x_0, x_0), f(x_0, x_0)), \ldots\}$.

Complexitatea algoritmului

- Problema de unificare

  $R = \{x_1 \doteq f(x_0, x_0), x_2 \doteq f(x_1, x_1), \ldots, x_n \doteq f(x_{n-1}, x_{n-1})\}$

  are cgu $S = \{x_1 \leftarrow f(x_0, x_0), x_2 \leftarrow f(f(x_0, x_0), f(x_0, x_0)), \ldots\}$.

- La pasul Elimină, pentru a verifica că o variabilă x_i nu apare în membrul drept al ecuației (occur check) facem 2^i comparații.

Complexitatea algoritmului

- Problema de unificare

  $R = \{x_1 \doteq f(x_0, x_0), x_2 \doteq f(x_1, x_1), \ldots, x_n \doteq f(x_{n-1}, x_{n-1})\}$

  are cgu $S = \{x_1 \leftarrow f(x_0, x_0), x_2 \leftarrow f(f(x_0, x_0), f(x_0, x_0)), \ldots\}$.

- La pasul Elimină, pentru a verifica că o variabilă x_i nu apare în membrul drept al ecuației (occur check) facem 2^i comparații.

- Algoritmul de unificare prezentat anterior este exponențial. Complexitatea poate fi îmbunătățită printr-o reprezentare eficientă a termenilor.

  K. Knight, Unification: A Multidisciplinary Survey, ACM Computing Surveys, Vol. 21, No. 1, 1989.

Pe săptămâna viitoare!