

# Technika a styl programování v Prologu

- Styl programování v Prologu
  - některá pravidla správného stylu
  - správný vs. špatný styl
  - komentáře
- Ladění
- Efektivita

## Technika a styl programování v Prologu

### Styl programování v Prologu I.

- Cílem stylistických konvencí je
  - redukce nebezpečí programovacích chyb
  - psaní čitelných a srozumitelných programů, které se dobře ladí a modifikují
- Některá pravidla správného stylu
  - krátké klauzule
  - krátké procedury; dlouhé procedury pouze s uniformní strukturou (tabulka)
  - klauzule se základními (hraničními) případy psát před rekurzivními klauzulemi
  - vhodná jména procedur a proměnných
    - nepoužívat seznamy ([...]) nebo závorky ({...}, (...)) pro termy pevné arity
  - vstupní argumenty psát před výstupními
  - **struktura programu – jednotné konvence** v rámci celého programu, např.
    - mezery, prázdné řádky, odsazení
    - klauzule stejné procedury na jednom místě; prázdné řádky mezi klauzulemi; každý cíl na zvláštním řádku

### Správný styl programování

- konstrukce setříděného seznamu Seznam3 ze setříděných seznamů Seznam1, Seznam2: `merge( Seznam1, Seznam2, Seznam3 )`
- `merge( [2,4,7], [1,3,4,8], [1,2,3,4,4,7,8] )`
- `merge( [], Seznam, Seznam ) :-`  
`!. % prevence redundantních řešení`  
`merge( Seznam, [], Seznam ).`
- `merge( [X|Telo1], [Y|Telo2], [X|Telo3] ) :-`  
`X < Y, !,`  
`merge( Telo1, [Y|Telo2], Telo3 ).`
- `merge( Seznam1, [Y|Telo2], [Y|Telo3] ) :-`  
`merge( Seznam1, Telo2, Telo3 ).`

## Špatný styl programování

```
merge( S1, S2, S3 ) :-
  S1 = [], !, S3 = S2;           % první seznam je prázdný
  S2 = [], !, S3 = S1;           % druhý seznam je prázdný
  S1 = [X|T1],
  S2 = [Y|T2],
  ( X < Y, !,
    Z = X,                         % Z je hlava seznamu S3
    merge( T1, S2, T3 );
    Z = Y,
    merge( S1, T2, T3 ) ),
  S3 = [ Z | T3 ].
```

## Dokumentace a komentáře

- co program dělá, jak ho používat (jaký cíl spustit a jaké jsou očekávané výsledky), příklad použití
- které predikáty jsou hlavní (*top-level*)
- jak jsou hlavní koncepty (objekty) reprezentovány
- doba výpočtu a paměťové nároky
- jaké jsou limity programu
- zda jsou použity nějaké speciální rysy závislé na systému
- jaký je význam predikátů v programu, jaké jsou jejich argumenty, které jsou vstupní a které výstupní (pokud víme)
  - vstupní argumenty „+“, výstupní „-“      `merge( +Seznam1, +Seznam2, -Seznam3 )`
  - `JmenoPredikatu/Arit`      `merge/3`
- algoritmické a implementační podrobnosti

## Styl programování v Prologu II.

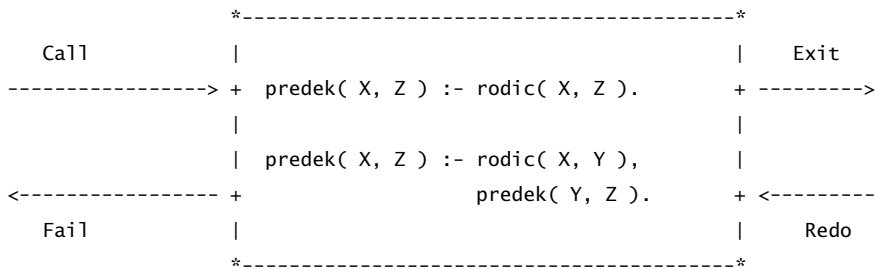
- **Středník „;“** může způsobit nesrozumitelnost klauzule
  - nedávat středník na konec řádku, používat závorky
  - v některých případech: rozdělení klauzule se středníkem do více klauzulí
- **Opatrné používání operátoru řezu**
  - preferovat použití zeleného řezu (nemění deklarativní sémantiku)
  - červený řez používat v jasně definovaných konstruktech
    - negace: `P, !, fail; true`      `\+ P`
    - alternativy: `Podminka, !, Ci11 ; Ci12`      `Podminka -> Ci11 ; Ci12`
- **Opatrné používání negace „\+“**
  - negace jako neúspěch: negace není ekvivalentní negaci v matematické logice
- **Pozor na `assert` a `retract`**: snižují transparentnost chování programu

## Ladění

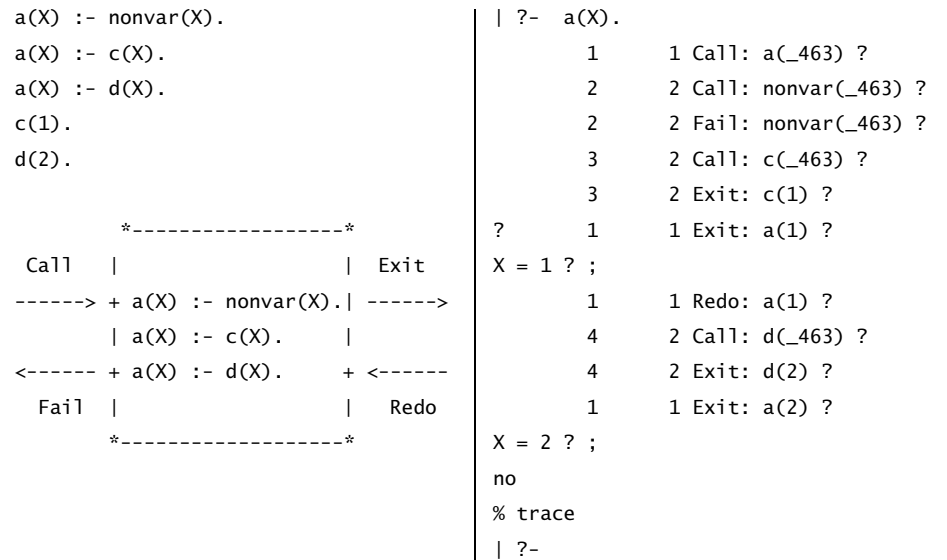
- Přepínače na trasování: `trace/0`, `notrace/0`
- Trasování specifického predikátu: `spy/1`, `nosp/1`
  - `spy( merge/3 )`
- `debug/0`, `nodebug/0`: pro trasování pouze predikátů zadaných `spy/1`
- Libovolná část programu může být spuštěna zadáním vhodného dotazu: **trasování cíle**
  - vstupní informace: jméno predikátu, hodnoty argumentů při volání
  - výstupní informace
    - při úspěchu hodnoty argumentů splňující cíl
    - při neúspěchu indikace chyby
  - nové vyvolání přes `;`: stejný cíl je volán při backtrackingu

# Krabičkový (4-branový) model

- Vizualizace řídicího toku (backtrackingu) na úrovni predikátu
  - Call: volání cíle
  - Exit: úspěšné ukončení volání cíle
  - Fail: volání cíle neuspělo
  - Redo: jeden z následujících cílů neuspěl a systém backtrackuje, aby našel alternativy k předchozímu řešení



# Příklad: trasování



## Efektivita

- Čas výpočtu, paměťové nároky, a také časové nároky na vývoj programu
  - u Prologu můžeme častěji narazit na problémy s časem výpočtu a paměti
  - Prologovské aplikace redukují čas na vývoj
  - vhodnost pro symbolické, nenumerické výpočty se strukturovanými objekty a relacemi mezi nimi
- Pro zvýšení efektivity je nutno se zabývat **procedurálními aspekty**
  - zlepšení efektivity při prohledávání**
    - odstranění zbytečného backtrackingu
    - zrušení provádění zbytečných alternativ co nejdříve
  - návrh **vhodnějších datových struktur**, které umožní efektivnější operace s objekty

## Zlepšení efektivity: základní techniky

- Optimalizace posledního volání (LCO) a akumulátory**
- Rozdílové seznamy** při spojování seznamů
- Caching:** uložení vypočítaných výsledků do programové databáze
- Indexace** podle prvního argumentu
  - např. v SICStus Prologu
  - při volání predikátu s prvním nainstanciováním argumentem se používá hašovací tabulka zpřístupňující pouze odpovídající klauzule
  - zamestnanec( Prijmeni, KrestniJmeno, Oddeleni, ...)
  - seznamy( [], ... ) :- ... .
  - seznamy( [HT], ... ) :- ... .
- Determinismus:**
  - rozhodnout, které klauzule mají uspět vícekrát, ověřit požadovaný determinismus

# Teorie logického programování

## Predikátová logika 1.řádu

- PROLOG: PROgramming in LOGic, část predikátové logiky 1.řádu
  - fakta:  $\text{rodic}(\text{petr}, \text{petrik}), \forall X a(X)$
  - klauzule:  $\forall X \forall Y \text{rodic}(X, Y) \Rightarrow \text{predek}(X, Y)$
- Predikátová logika I. řádu (PL1)
  - soubory objektů: lidé, čísla, body prostoru, ...
  - syntaxe PL1, sémantika PL1, pravdivost a dokazatelnost
- Rezoluce ve výrokové logice, v PL1
  - dokazovací metoda
- Rezoluce v logickém programování
- Backtracking, řez, negace vs. rezoluce

## Predikátová logika I. řádu (PL1)

Abeceda  $\mathcal{A}$  jazyka  $\mathcal{L}$  PL1 se skládá ze symbolů:

- **proměnné**  $X, Y, \dots$  označují libovolný objekt z daného oboru
- **funkční symboly**  $f, g, \dots$  označují operace (příklad:  $+, \times$ )
  - **arita** = počet argumentů,  $n$ -ární symbol, značíme  $f/n$
  - nulární funkční symboly – **konstanty**: označují význačné objekty (příklad:  $0, 1, \dots$ )
- **predikátové symboly**  $p, q, \dots$  pro vyjádření vlastností a vztahů mezi objekty
  - **arita** = počet argumentů,  $n$ -ární symbol, značíme  $p/n$      příklad:  $<, \in$
- **logické spojky**  $\wedge, \vee, \neg, \Rightarrow, \equiv$
- **kvantifikátory**  $\forall, \exists$ 
  - logika I. řádu používá **kvantifikaci pouze pro individua** (odlišnost od logik vyššího řádu)
  - v logice 1. řádu nelze:  $\forall \mathbb{R} : \forall A \subseteq \mathbb{R}, \forall f : \mathbb{R} \rightarrow \mathbb{R}$
- **závorky**:  $), ($

## Jazyky PL1

- Specifikace jazyka  $\mathcal{L}$  je definována funkčními a predikátovými symboly  
symboly tedy určují oblast, kterou jazyk popisuje
- **Jazyky s rovností**: obsahují predikátový symbol pro rovnost „=“

### Příklady

- jazyk teorie uspořádání
  - jazyk  $s =$ , binární predikátový symbol  $<$
- jazyk teorie množin
  - jazyk  $s =$ , binární predikátový symbol  $\in$
- jazyk elementární aritmetiky
  - jazyk  $s =$ , nulární funkční symbol  $0$  pro nulu,  
unární funkční symbol  $s$  pro operaci následníka,  
binární funkční symboly pro sčítání  $+$  a násobení  $\times$

# Term, atomická formule, formule

- **Term** nad abecedou  $\mathcal{A}$ 
  - každá proměnná z  $\mathcal{A}$  je term
  - je-li  $f/n$  z  $\mathcal{A}$  a  $t_1, \dots, t_n$  jsou termy, pak  $f(t_1, \dots, t_n)$  je také term
  - každý term vznikne konečným počtem užití přechozích kroků  $f(X, g(X, 0))$
- **Atomická formule (atom)** nad abecedou  $\mathcal{A}$ 
  - je-li  $p/n$  z  $\mathcal{A}$  a  $t_1, \dots, t_n$  jsou termy, pak  $p(t_1, \dots, t_n)$  je atomická formule  $f(X) < g(X, 0)$
- **Formule** nad abecedou  $\mathcal{A}$ 
  - každá atomická formule je formule
  - jsou-li  $F$  a  $G$  formule, pak také  $(\neg F), (F \wedge G), (F \vee G), (F \Rightarrow G), (F \equiv G)$  jsou formule
  - je-li  $X$  proměnná a  $F$  formule, pak také  $(\forall X F)$  a  $(\exists X F)$  jsou formule
  - každá formule vznikne konečným počtem užití přechozích kroků  $(\exists X ((f(X) = 0) \wedge p(0)))$

# Sémantika formulí

- **Ohodnocení proměnné**  $\varphi(X)$ : každé proměnné  $X$  je přiřazen prvek  $|I|$
- **Hodnota termu**  $\varphi(t)$ : každému termu je přiřazen prvek univerza
  - příklad: necht'  $\varphi(X) := 0$
  - $\varphi(\text{plus}(s(\text{zero}), X)) = \varphi(s(\text{zero})) + \varphi(X) = (1 + \varphi(\text{zero})) + 0 = (1 + 0) + 0 = 1$
- Každá **dobře utvořená formule** označuje **pravdivostní hodnotu** (PRAVDA, NEPRAVDA) v závislosti na své struktuře a interpretaci
  - Pravdivá formule**  $I \models_{\varphi} Q$ : formule  $Q$  označena PRAVDA
  - Nepravdivá formule**  $I \not\models_{\varphi} Q$ : formule  $Q$  označena NEPRAVDA
  - příklad:  $p/1$  predikátový symbol, tj.  $p \subseteq |I|$   $p := \{\langle 1 \rangle, \langle 3 \rangle, \langle 5 \rangle, \dots\}$
  - $I \models p(\text{zero}) \wedge p(s(\text{zero}))$  iff  $I \models p(\text{zero})$  a  $I \models p(s(\text{zero}))$
  - iff  $\langle \varphi(\text{zero}) \rangle \in p$  a  $\langle \varphi(s(\text{zero})) \rangle \in p$
  - iff  $\langle \varphi(\text{zero}) \rangle \in p$  a  $\langle (1 + \varphi(\text{zero})) \rangle \in p$
  - iff  $\langle 0 \rangle \in p$  a  $\langle 1 \rangle \in p$
  - $\langle 1 \rangle \in p$  ale  $\langle 0 \rangle \notin p$ , tedy formule je nepravdivá v  $I$

# Interpretace

- **Interpretace**  $I$  jazyka  $\mathcal{L}$  nad abecedou  $\mathcal{A}$  je dána
  - neprázdnou množinou  $\mathcal{D}$  (také značíme  $|I|$ , nazývá se **univerzum**) a
  - zobrazením, které
    - každé konstantě  $c \in \mathcal{A}$  přiřadí nějaký **prvek**  $\mathcal{D}$
    - každému funkčnímu symbolu  $f/n \in \mathcal{A}$  přiřadí  $n$ -ární **operaci** nad  $\mathcal{D}$
    - každému predikátovému symbolu  $p/n \in \mathcal{A}$  přiřadí  $n$ -ární **relaci** nad  $\mathcal{D}$
- **Příklad: uspořádání na  $\mathbb{R}$** 
  - jazyk: predikátový symbol  $\text{mensi}/2$
  - interpretace: univerzum  $\mathbb{R}$ ; zobrazení:  $\text{mensi}(x, y) := x < y$
- **Příklad: elementární aritmetika nad množinou  $\mathbb{N}$  (včetně 0)**
  - jazyk: konstanta  $\text{zero}$ , funční symboly  $s/1$ ,  $\text{plus}/2$
  - interpretace:
    - univerzum  $\mathbb{N}$ ; zobrazení:  $\text{zero} := 0$ ,  $s(x) := 1 + x$ ,  $\text{plus}(x, y) := x + y$

# Model

- Interpretace se nazývá **modelem** formule, je-li v ní tato formule pravdivá
  - interpretace množiny  $\mathbb{N}$  s obvyklými operacemi je modelem formule  $(1 + s(0) = s(s(0)))$
  - interpretace, která se liší přiřazením  $s/1$ :  $s(x) := x$  není modelem této formule
- **Teorie  $\mathcal{T}$**  jazyka  $\mathcal{L}$  je množina formulí jazyka  $\mathcal{L}$ , tzv. **axiomů**
  - $\neg s(X) = 0$  je jeden z axiomů teorie elementární aritmetiky
- **Model teorie**: libovolná interpretace, která je modelem všech jejích axiomů
  - všechny axiomy teorie musí být v této interpretaci pravdivé
- **Pravdivá formule v teorii**  $\mathcal{T} \models F$ : pravdivá v každém z modelů teorie  $\mathcal{T}$ 
  - říkáme také formule **platí v teorii** nebo je **splněna v teorii**
  - formule  $1 + s(0) = s(s(0))$  je pravdivá v teorii elementárních čísel
- **Logicky pravdivá formule**  $\models F$ : libovolná interpretace je jejím modelem
  - nebo-li  $F$  je pravdivá v každém modelu libovolné teorie
  - formule  $G \vee \neg G$  je logicky pravdivá, formule  $1 + s(0) = s(s(0))$  není logicky pravdivá

## Zkoumání pravdivosti formulí

- Zjištění pravdivosti provádíme důkazem

**Důkaz:** libovolná posloupnost  $F_1, \dots, F_n$  formulí jazyka  $\mathcal{L}$ , v níž každé  $F_i$  je buď axiom teorie jazyka  $\mathcal{L}$  nebo lze  $F_i$  odvodit z předchozích  $F_j$  ( $j < i$ ) použitím určitých **odvozovacích pravidel**

- Odvozovací pravidla – příklady

- **pravidlo modus ponens:** z formulí  $F$  a  $F \Rightarrow G$  lze odvodit  $G$
- **rezoluční princip:** z formulí  $F \vee A$ ,  $G \vee \neg A$  odvodit  $F \vee G$

- $F$  je **dokazatelná z formulí**  $A_1, \dots, A_n$

$$A_1, \dots, A_n \vdash F$$

existuje-li důkaz  $F$  z  $A_1, \dots, A_n$

- Dokazatelné formule v teorii  $\mathcal{T}$  nazýváme **teorémy** teorie  $\mathcal{T}$

## Korektnost a úplnost

- **Uzavřená formule:** neobsahuje volnou proměnnou (bez kvantifikace)

- $\forall Y ((0 < Y) \wedge (\exists X (X < Y)))$  je uzavřená formule
- $(\exists X (X < Y))$  není uzavřená formule

- Množina odvozovacích pravidel se nazývá **korektní**, jestliže pro každou množinu uzavřených formulí  $\mathcal{P}$  a každou uzavřenou formuli  $F$  platí:

$$\text{jestliže } \mathcal{P} \vdash F \text{ pak } \mathcal{P} \models F \quad (\text{jestliže je něco dokazatelné, pak to platí})$$

Odvozovací pravidla jsou **úplná**, jestliže

$$\text{jestliže } \mathcal{P} \models F \text{ pak } \mathcal{P} \vdash F \quad (\text{jestliže něco platí, pak je to dokazatelné})$$

- PL1: úplná a korektní dokazatelnost, tj.

pro teorii  $\mathcal{T}$  s množinou axiomů  $\mathcal{A}$  platí:  $\mathcal{T} \models F$  **právě když**  $\mathcal{A} \vdash F$