

9 Jednoduchý deklarativní jazyk

Pokračujeme v tématu předchozí lekce, tj. budeme se zabývat matematickým dokazováním vlastností a správnosti algoritmů. Třebaže mnohým mohla přijít už Lekce 8 více než dost formální, není tomu úplně tak; nyní si ukážeme (ještě) přesnější přístup založený na myšlenkách funkcionálního programování.

$f(3)$	↪	if 3 then 3 * f(3 - 1) else 1 fi	↪	3 * f(3 - 1)	↪
$3 * f(2)$	↪	3 * (if 2 then 2 * f(2 - 1) else 1 fi)	↪	3 * (2 * f(2 - 1))	↪
$3 * (2 * f(1))$	↪	3 * (2 * (if 1 then 1 * f(1 - 1) else 1 fi))	↪	3 * (2 * (1 * f(1 - 1)))	↪
$3 * (2 * (1 * f(0)))$	↪	3 * (2 * (1 * (if 0 then 0 * f(0 - 1) else 1 fi)))	↪	3 * (2 * (1 * 1))	↪
$3 * (2 * 1)$	↪	3 * 2	↪	6	

□

Stručný přehled lekce

- * Zavedení jednoduchého deklarativního jazyka, jeho formální přínos.
- * Formalizace pojmu „výpočet“ z hlediska našeho jazyka.
- * Několik konkrétních zápisů algoritmů a jejich vyhodnocení / důkaz.

O „správnosti“ programů, podruhé

Vraťme se k otázce, jak se máme přesvědčit, že program funguje „správně“? □

- V některých případech, jak už jsme zmínili, je třeba mít **naprostou jistotu**, že program funguje tak jak má, například v řídicích systémech, na nichž závisí lidské životy. □

V takovém případě je jedinou „dostatečně spolehlivou“ možností podat formální **matematický důkaz** chování algoritmu. □

- A co tedy důkazy vlastností symbolicky zapsaných (procedurálních) algoritmů z Lekce 8? Všimli jste si, co v nich bylo problematickým bodem? □

Náš procedurální zápis algoritmu totiž přesně nedefinuje, co je to „**elementární krok**“ výpočtu – to je sice většinou docela zřejmé, někdy však může hlavní problém nastat právě zde. Sice by bylo možné použít k definici některý z přesných teoretických modelů výpočtu jako je *Turingův stroj* (nebo třeba i některý vhodný z reálných programovacích jazyků), avšak pak by se formální důkazy staly velmi složitými. □

- Vhodnějším řešením (pro potřeby formálního dokazování) se jeví příklon k „**funkcionálnímu**“ zápisu algoritmů pomocí matematicky zcela přesných *deklarací*.

9.1 Popis jednoduchého deklarativního jazyka

Definice 9.1. Deklarativní programovací jazyk (pro přednášky FI:IB000).

- Necht' $Var = \{x, y, z, \dots\}$ je spočetná množina *proměnných*. □
- Necht' $Num = \{0, 1, \dots, 52, \dots, 397, \dots\}$ je množina všech dekadických zápisů *přirozených* čísel. □
- Necht' $Fun = \{f, g, h, \dots\}$ je spočetná množina *funkčních symbolů*. Ke každému $f \in Fun$ je přiřazeno číslo $a \in \mathbb{N}$, které nazýváme *arita* f . Dále předpokládáme, že pro každé $a \in \mathbb{N}$ existuje nekonečně mnoho $f \in Fun$ s aritou a . □
- Množina **výrazů** Exp je (*induktivně*) definována následující abstraktní syntaktickou rovnicí:

$$\begin{aligned} E ::= & x \mid \mathbf{n} \\ & \mid E_1 + E_2 \mid E_1 - E_2 \mid E_1 * E_2 \mid E_1 \div E_2 \mid (E_1) \\ & \mid f(E_1, \dots, E_a) \\ & \mid \text{if } E_1 \text{ then } E_2 \text{ else } E_3 \text{ fi} \end{aligned}$$

V uvedené rovnici je $x \in Var$, $\mathbf{n} \in Num$, $f \in Fun$ a $a \in \mathbb{N}$ je arita f .

Poznámka: Taková specifikace syntaxe je **abstraktní** v tom smyslu, že se nezabývá tím, jak výrazy jednoznačně zapsat do řádku jako posloupnost symbolů. Je na nás, abychom napsali dostatečně mnoho závorek a případně stanovili prioritu operátorů tak, aby bylo zcela jasné, jak daný výraz podle uvedené rovnice vznikl. □

(Ve smyslu Lekce 6 tato induktivní definice **není jednoznačná**. To nám však nebude v Definici 9.2 vadit.)

Pro lepší pochopení uvádíme několik příkladů výrazů (*Exp*) z definice.

- **254**
- **$2 + 3 * 4$** □
- **$f(2) \div g(5)$**
- **$f(2 + x, g(y, 3 * y))$** □
- **if $x - 1$ then $(2 + f(y))$ else $g(x, x)$ fi**

(Vyhodnocení podmínky v „if“ testuje nenulovost argumentu.)

Definice: *Deklarace* (v jazyce Definice 9.1) je konečný **system rovnic** tvaru

$$\begin{aligned} f_1(x_1, \dots, x_{a_1}) &= E_1 \\ &\vdots \\ f_n(x_1, \dots, x_{a_n}) &= E_n \end{aligned},$$

kde pro každé $1 \leq i \leq n$ platí, že $f_i \in Fun$ je funkce arity a_i , že $x_1, \dots, x_{a_i} \in Var$ jsou proměnné a E_i je výraz, v němž se mohou vyskytovat pouze proměnné x_1, \dots, x_{a_i} a funkční symboly f_1, \dots, f_n . \square

Opět uvádíme pro osvětlení několik příkladů deklarací z naší definice.

- $f(x) = \text{if } x \text{ then } x * f(x - 1) \text{ else } 1 \text{ fi}$ \square

- $f(x) = g(x - 1, x)$
• $g(x, y) = \text{if } x \text{ then } f(y) \text{ else } 3 \text{ fi}$

(Jak uvidíme formálně později, konvencí našich výpočtů je neužívat záporná čísla, místo toho $0 - 1$ „=“ 0.) \square

- $g(x, y) = \text{if } x - y \text{ then } x \text{ else } y \text{ fi}$ \square

- $f(x) = f(x)$ \square

(Nezapisuje toto náhodou „nekonečnou smyčku“?)

9.2 Formalizace pojmu „výpočet“

Za výpočet (nad Δ) budeme považovat posloupnost úprav výrazů, které jsou „postaveny“ na naši uvažované deklaraci Δ . To je formálně podchyceno v následujících dvou definicích. Srovnajte si Definici 9.2 také s neformálním pojetím algoritmů jako konečných posloupností elementárních kroků v Oddíle 1.3.

Definice: Buď Δ deklarace. Symbolem $Exp(\Delta)$ označíme množinu všech výrazů E , které splňují zároveň tyto dvě podmínky:

- E neobsahuje žádnou proměnnou.
- Jestliže E obsahuje funkční symbol f , pak f byl v Δ deklarován. \square

Dívejte se na množinu $Exp(\Delta)$ jako na soubor „platných vstupů“ (jako u programu) pro deklaraci Δ , nad kterými bude prováděn výpočet.

Fakt: Množinu $Exp(\Delta)$ lze definovat také induktivně:

$$E ::= \mathbf{n} \mid (E_1) \mid E_1 + E_2 \mid E_1 - E_2 \mid E_1 * E_2 \mid E_1 \div E_2 \\ \mid f(E_1, \dots, E_a) \mid \text{if } E_1 \text{ then } E_2 \text{ else } E_3 \text{ fi}$$

V uvedené zápise je $\mathbf{n} \in Num$, $f \in Fun$ je funkční symbol deklarovaný v Δ a $a \in \mathbb{N}$ je arita tohoto f .

Definice 9.2. Výpočet a krok výpočtu v našem deklarativním jazyce. Funkci „krok výpočtu“ $\mapsto : Exp(\Delta) \rightarrow Exp(\Delta)$ definujeme induktivně takto; místo $\mapsto (E) = F$ budeme psát $\mathbf{E} \mapsto \mathbf{F}$.

- $\mathbf{n} \mapsto \mathbf{n}$ pro každé $\mathbf{n} \in Num$. \square
- Pro $\mathbf{E} \equiv (E_1)$ definujeme krok výpočtu takto:
 - * Jestliže $E_1 \mapsto F \in Num$, pak $(E_1) \mapsto F$.
 - * Jestliže $E_1 \mapsto F \notin Num$, pak $(E_1) \mapsto (F)$. \square
- Pro $\mathbf{E} \equiv E_1 + E_2$ definujeme krok výpočtu takto:
 - * Jestliže $E_1, E_2 \in Num$, pak $E_1 + E_2 \mapsto \mathbf{z}$, kde \mathbf{z} je dekadický zápis čísla $E_1 + E_2$.
 - * Jestliže $E_1 \notin Num$, pak $E_1 + E_2 \mapsto F + E_2$, kde $E_1 \mapsto F$.
 - * Jestliže $E_1 \in Num$ a $E_2 \notin Num$, pak $E_1 + E_2 \mapsto E_1 + F$, kde $E_2 \mapsto F$.

- Pro $E \equiv E_1 - E_2$ definujeme krok výpočtu takto:
 - * Jestliže $E_1, E_2 \in Num$, pak $E_1 - E_2 \mapsto \mathbf{z}$, kde \mathbf{z} je dekadický zápis čísla $\max\{0, E_1 - E_2\}$. (Pozor na **nezápornost** výsledku odčítání!)
 - * Jestliže $E_1 \notin Num$, pak $E_1 - E_2 \mapsto F - E_2$, kde $E_1 \mapsto F$.
 - * Jestliže $E_1 \in Num$ a $E_2 \notin Num$, pak $E_1 - E_2 \mapsto E_1 - F$, kde $E_2 \mapsto F$.□
- Pro $E \equiv E_1 * E_2$ definujeme krok výpočtu takto:
 - * Jestliže $E_1, E_2 \in Num$, pak $E_1 * E_2 \mapsto \mathbf{z}$, kde \mathbf{z} je dekadický zápis čísla $E_1 * E_2$.
 - * Jestliže $E_1 \notin Num$, pak $E_1 * E_2 \mapsto F * E_2$, kde $E_1 \mapsto F$.
 - * Jestliže $E_1 \in Num$ a $E_2 \notin Num$, pak $E_1 * E_2 \mapsto E_1 * F$, kde $E_2 \mapsto F$.□
- Pro $E \equiv E_1 \div E_2$ definujeme krok výpočtu takto:
 - * Jestliže $E_1, E_2 \in Num$, pak $E_1 \div E_2 \mapsto \mathbf{z}$, kde \mathbf{z} je dekadický zápis (celé části) čísla $\lfloor E_1/E_2 \rfloor$. Pokud $E_2 \equiv 0$, je $\mathbf{z} \equiv 0$ (**dělení nulou!**).
 - * Jestliže $E_1 \notin Num$, pak $E_1 \div E_2 \mapsto F \div E_2$, kde $E_1 \mapsto F$.
 - * Jestliže $E_1 \in Num$ a $E_2 \notin Num$, pak $E_1 \div E_2 \mapsto E_1 \div F$, kde $E_2 \mapsto F$.

- Pro $E \equiv \text{if } E_1 \text{ then } E_2 \text{ else } E_3 \text{ fi}$ definujeme krok výpočtu takto:
 - * Jestliže $E_1 \in Num$ a $E_1 \equiv 0$, pak $\text{if } E_1 \text{ then } E_2 \text{ else } E_3 \text{ fi} \mapsto (E_3)$.
 - * Jestliže $E_1 \in Num$ a $E_1 \not\equiv 0$, pak $\text{if } E_1 \text{ then } E_2 \text{ else } E_3 \text{ fi} \mapsto (E_2)$.
 - * Jestliže $E_1 \notin Num$, pak
 $\text{if } E_1 \text{ then } E_2 \text{ else } E_3 \text{ fi} \mapsto \text{if } F \text{ then } E_2 \text{ else } E_3 \text{ fi}$, kde $E_1 \mapsto F$. \square
- Pro $E \equiv f(E_1, \dots, E_k)$ definujeme krok výpočtu takto:
 - * Jestliže $E_1, \dots, E_k \in Num$, pak $f(E_1, \dots, E_k) \mapsto (E_f(x_1 \upharpoonright E_1, \dots, x_k \upharpoonright E_k))$.
 - * Jinak $f(E_1, \dots, E_k) \mapsto f(E_1, \dots, E_{i-1}, F, E_{i+1}, \dots, E_k)$, kde i je nejmenší index pro který platí $E_i \notin Num$ a $E_i \mapsto F$. \square

V zápise jednotlivých bodů vždy platí, že $E_1, E_2, \dots \in Exp(\Delta)$. Znak \equiv zde znamená „textovou“ rovnost na množině výrazů Exp .

Při nejednoznačnosti vždy aplikujeme **první použitelné** pravidlo naší definice zleva.

Reflexivní a tranzitivní uzávěr relace \mapsto značíme \mapsto^* („výpočet“).

Tato rozsáhlá a důležitá definice si zaslouží několik podstatných připomínek.

- Za prvé si dobře povšimněte některých „aritmetických“ aspektů výpočtu.
 - Výsledek odečítání je vždy nezáporný, neboli všechna záporná čísla jsou nahrazena nulou. □
 - Výsledek dělení je vždy celočíselný, počítáme jeho dolní celou část. □
 - **Dělení nulou** je definováno (není chybou), výsledkem je opět nula. □
- Další připomínka se týká pořadí vyhodnocování ve výrazu — to je přesně dáno pořadím pravidel v Definici 9.2, neboli vždy aplikujeme první pravidlo, které aktuálně lze použít na výraz E , a to na prvním možném místě zleva. □ Mimo jiné je takto „definována“ nejvyšší priorita vyhodnocení uzávorkovaného výrazu. □
- Uvědomte si dobře, že definice výpočetního kroku \mapsto je (poněkud skrytě) **rekurzivní**. Třeba krok $(2 * 1) \mapsto 2$ je ve skutečnosti jediným krokem, přestože „vyvolá“ použití dvou pravidel v sobě – vyhodnocení součinu i odstranění závorek. □
- Ještě si uveďme, že naše definice připouští jistý **nedeterminismus**: Je možné mít v deklaraci Δ zadaných více rovnic pro tutéž funkci $f()$, pak se však z \mapsto stává pouhá relace. My se touto možností nebudeme zabývat.

9.3 Příklady výpočtů a důkazů

Příklad 9.3. Ukážeme si několik ilustrativních „výpočtů“ nad různými deklaracemi.

Uvažme deklaraci $f(x) = \text{if } x \text{ then } x * f(x - 1) \text{ else } 1 \text{ fi}$. Pak třeba $f(3) \mapsto^* 6$, neboť

$$\begin{array}{llll} f(3) & \mapsto \text{if } 3 \text{ then } 3 * f(3 - 1) \text{ else } 1 \text{ fi} & \mapsto 3 * f(3 - 1) & \mapsto \\ 3 * f(2) & \mapsto 3 * (\text{if } 2 \text{ then } 2 * f(2 - 1) \text{ else } 1 \text{ fi}) & \mapsto 3 * (2 * f(2 - 1)) & \mapsto \\ 3 * (2 * f(1)) & \mapsto 3 * (2 * (\text{if } 1 \text{ then } 1 * f(1 - 1) \text{ else } 1 \text{ fi})) & \mapsto 3 * (2 * (1 * f(1 - 1))) & \mapsto \\ 3 * (2 * (1 * f(0))) & \mapsto 3 * (2 * (1 * (\text{if } 0 \text{ then } 0 * f(0 - 1) \text{ else } 1 \text{ fi}))) & \mapsto 3 * (2 * (1 * 1)) & \mapsto \\ 3 * (2 * 1) & \mapsto 3 * 2 & \mapsto 6. \square & \end{array}$$

Uvažme deklaraci $f(x) = g(x - 1, x)$ a $g(x, y) = \text{if } x \text{ then } f(y) \text{ else } 3 \text{ fi}$. Pak například

$$f(3) \mapsto g(3 - 1, 3) \mapsto g(2, 3) \mapsto \text{if } 2 \text{ then } f(3) \text{ else } 3 \text{ fi} \mapsto f(3). \square$$

Uvažme deklaraci $f(x) = f(x)$. Pak pro každé $n \in \text{Num}$ platí

$$f(n) \mapsto f(n)$$

a podobně $f(f(n)) \mapsto f(f(n))$. Ale $f(f(2 + 3)) \mapsto f(f(5)) \mapsto f(f(5))$. \square \square

Poznámka: Všimněte si, že při našich úpravách výrazů si dovoluujeme vynechávat zbytečné „vnější“ závorky kolem celého výrazu.

Důkaz správnosti programu

Příklad 9.4. Pro ukázkou uvažme deklaraci Δ obsahující pouze rovnici

$$f(x) = \text{if } x \text{ then } x * f(x - 1) \text{ else } 1 \text{ fi.}$$

Věta. Pro každé $n \in \mathbb{N}$ platí $f(\mathbf{n}) \mapsto^* \mathbf{m}$, kde $\mathbf{m} \equiv n!$. \square

Důkaz provedeme indukcí podle n :

- **Báze** $n = 0$. Platí $f(\mathbf{0}) \mapsto \text{if } \mathbf{0} \text{ then } \mathbf{0} * f(\mathbf{0} - 1) \text{ else } 1 \text{ fi} \mapsto 1$ a také $0! = 1$. \square
- **Indukční krok.** Necht' $n + 1 \equiv \mathbf{k}$. Pak

$$f(\mathbf{k}) \mapsto \text{if } \mathbf{k} \text{ then } \mathbf{k} * f(\mathbf{k} - 1) \text{ else } 1 \text{ fi} \mapsto \mathbf{k} * f(\mathbf{k} - 1) \mapsto \mathbf{k} * f(\mathbf{w}),$$

kde $\mathbf{w} \equiv k - 1 = n$. \square Podle I.P. platí $f(\mathbf{w}) \mapsto^* \mathbf{u}$, kde $\mathbf{u} \equiv n!$. Proto $\mathbf{k} * f(\mathbf{w}) \mapsto^* \mathbf{k} * \mathbf{u} \mapsto \mathbf{v}$, kde $\mathbf{v} \equiv (n + 1) \cdot n! = (n + 1)!$. \square

\square

Vidíte, jak „hutný“ a formálně zcela přesný zápis důkazu naše formalizace umožňuje?

Důkazy „neukončenosti“ výpočtů

Věta 9.5. *Bud' Δ deklarace. Pro každé $i \in \mathbb{N}$ definujeme relaci $\mapsto^i \subseteq \text{Exp}(\Delta) \times \text{Exp}(\Delta)$ předpisem $\mapsto^i = \underbrace{\mapsto \circ \dots \circ \mapsto}_i$. Dále definitoricky kládeme $\mapsto^0 = \{(E, E) \mid E \in \text{Exp}(\Delta)\}$. Pak platí $\mapsto^* = \bigcup_{i=0}^{\infty} \mapsto^i$. \square*

Podle předchozí věty platí, že $E \mapsto^* F$ právě když $E \mapsto^i F$ pro nějaké $i \in \mathbb{N}$. Navíc musí existovat **nejmenší** i s touto vlastností. Toto pozorování bývá velmi užitečné v důkazech „neukončenosti“ výpočtů. \square

Příklad 9.6. *Uvažme deklaraci $f(x) = f(x)$.*

Věta. Pro každé $\mathbf{n} \in \text{Num}$ platí, že neexistuje žádné $\mathbf{m} \in \text{Num}$ takové, že $f(\mathbf{n}) \mapsto^* \mathbf{m}$. \square

Důkaz sporem: Předpokládejme, že existují $\mathbf{n}, \mathbf{m} \in \text{Num}$ takové, že $f(\mathbf{n}) \mapsto^* \mathbf{m}$. Pak existuje **nejmenší** $i \in \mathbb{N}$ takové, že $f(\mathbf{n}) \mapsto^i \mathbf{m}$. Jelikož výrazy $f(\mathbf{n})$ a \mathbf{m} jsou různé, platí $i > 0$. Jelikož $\mapsto^i = \mapsto^{i-1} \circ \mapsto$ a $f(\mathbf{n}) \mapsto f(\mathbf{n})$, platí $f(\mathbf{n}) \mapsto^{i-1} \mathbf{m}$, což je spor s minimalitou i . \square