FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Omezení oboru hodnot výrazu v AST z definičních oborů proměnných
Student:	Jan Pokorný
Vedoucí:	Ing. Štěpán Plachý
Studijní program:	Informatika
Studijní obor:	Teoretická informatika
Katedra:	Katedra teoretické informatiky
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

Seznamte se s reprezentací a implementací abstraktního syntaktického stromu výrazů ve vyhodnocovači v aplikaci Grades, vyvíjené ICT oddělením Faktulty informačních technologií ČVUT, psané v jazyce Java a běžící na adrese grades.fit.cvut.cz.
Navrhněte průchod abstraktním syntaktickým stromem výrazu, který na základě zadaného rozsahu možných hodnot jednotlivých proměnných co nejpřesněji omezí výsledný rozsah možných hodnot výrazu. Váš návrh by měl brát v úvahu různé datové typy výrazu.
Vámi navržený průchod do vyhodnocovače Grades implementujte a své řešení otestujte.

Seznam odborné literatury

Dodá vedoucí práce.

doc. Ing. Jan Janoušek, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 4. ledna 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Omezení oboru hodnot výrazu v AST z definičních oborů proměnných

Jan Pokorný

Katedra teoretické informatiky
Vedoucí práce: Ing. Štěpán Plachý

3. června 2020

Poděkování

Nejdříve bych chtěl poděkovat vedoucímu práce Ing. Štěpánu Plachému za jeho ochotu a čas, který mi věnoval při pravidelných konzultacích. Dále bych chtěl poděkovat rodině a skupině přátel scházející se ”Nad Pianem” za motivaci a psychickou podporu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 3. června 2020 ……

Odkaz na tuto práci

Pokorný, Jan. *Omezení oboru hodnot výrazu v AST z definičních oborů proměnných*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Tato bakalářská práce se zabývá omezením možných hodnot výrazu reprezentovaného pomocí abstraktního syntaktického stromu na základě omezení hodnot vstupních proměnných. Výrazy mohou obsahovat hodnoty různých datových typů. Práce představuje průchod abstraktním syntaktickým stromem, jehož výsledek obsahuje všechny hodnoty, které mohly vzniknout vyhodnocením výrazu s každým ohodnocením proměnných splňující omezení. Práce se zaobírá návrhem potřebných datových struktur a vyhodnocením vybraných funkcí. Navržený průchod je implementován do vyhodnocovače výrazů elektronické klasifikace Grades, který podporuje datové typy číslo, řetězec a boolean. Rozšíření vyhodnocovače umožní přídání funkcionality do aplikace Grades pro zjištění všech potencionálních klasifikací v jednotlivých předmětech a také pro automatizovanou kontrolu definice hodnocení.

Klíčová slova abstraktní syntaktický strom, vyhodnocovač výrazů, Grades FIT ČVUT, matematická optimalizace, určení možných výsledků výrazu

Abstract

Aim of this bachelor thesis is to restrict possible values of an expression represented by abstract syntax tree and based on restrictions of input variables. Expressions can contain values of different data types. This thesis introduces abstract syntax tree traversal, which results contain all values that could be acquired by evaluating expression with any of the variable values satisfying restrictions. The thesis discusses necessary data structures and the evaluation of selected functions. Designed traversal is implemented in expression evaluator in electronic classification application Grades, which supports data types number, string, and boolean. The extension of the expression evaluator will enable the addition of functionality to the Grades application for the identification of all possible classifications in individual subjects and also for the automated verification of the subject definition.

Keywords abstract syntax tree, expression evaluator, Grades FIT CTU, mathematical optimization, determination of possible expression evaluations

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Abstraktní syntaktický strom (AST) je datová struktura pro reprezentaci strukturalizovaného textu v strojově orientované formě, zejména zdrojového kódu programu. Je používán k analýze a optimalizaci programu, a následnému generování kódu v překladačích a řízení běhu programu v interpretech.

Tématem práce je navržení průchodu AST pro omezení oboru hodnot výrazu na základě omezení definičních oborů vstupních proměnných a rozšíření o tento průchod vyhodnocovač výrazů aplikace elektronické klasifikace Grades, vyvíjené ICT oddělením Faktulty informačních technologií ČVUT.

Práce je určena pro uživatele aplikace Grades, tedy pro studenty a učitele Fakulty informačních technologií ČVUT. Práce umožňuje přidání funkcionality do aplikace Grades pro zobrazení celého rozsahu možných ohodnocení, a tím studenty motivovat nebo lépe odhadovat potencionální průchodnosti předmětů a upozornění na špatnou definici hodnocení.

V teoretické části se práce věnuje vysvětlení základních pojmů, abstraktnímu syntaktickému stromu a matematické analýze. Následující kapitola se zabývá projektem Grades, zejména modulem vyhodnocovače výrazů, jeho jazyku a implementací AST včetně průchodu pro standardní vyhodnocení výrazu.

Třetí kapitola popisuje navržený průchod AST pro omezení možných hodnot výrazu. Dále se kapitola věnuje datovým strukturám pro uchovávání rozsahu hodnot a vyhodnocením vybraných funkcí. Poslední kapitola navazuje popisem implementace navrženého průchodu do vyhodnocovače výrazů v aplikaci Grades včetně konkrétních implementací datových struktur a dalších objektů použitých při rozsahovém vyhodnocení. Nakonec jsou popsány testovací metody na ověření nalezení všech možných hodnot, kterých může výraz nabývat, pomocí existujícího průchodu AST pro standardní vyhodnocení.

Cíl práce

Prvním cílem práce je navrhnutí průchodu abstraktním syntaktickým stromem, který na základě zadaného rozsahu možných hodnot jednotlivých vstupních proměnných co nejpřesněji omezí výsledný rozsah možných hodnot výrazů, přičemž bude brát v úvahu různé datové typy proměnných. Ve výsledném odhadu se musí nacházet všechna možná ohodnocení výrazu splňující omezení proměnných.

Posledním cílem je ověření korektnosti stávající implementace standardního vyhodnocování výrazů a jejího případné opravení.

Kapitola 1

Teoretická část

1.1 Základní pojmy

Definice 1.1.1 (Graf)**:** Nechť V je množina libovolných prvků a nechť $E = \{\{u, v\} | u, v \in V; u \neq v\}$. Uspořádanou dvojici (V, E) nazýváme *graf* G. Prvky V nazýváme vrcholy grafu G a prvky E hrany grafu G.

Definice 1.1.2 (Orientovaný graf)**:** Graf, jehož hrany jsou uspořádané dvojice vrcholů, nazýváme *orientovaným* grafem. O uspořádané dvojici $(u, v) \in E$ řekneme, že vede orientovaná hrana z u do v.

Definice 1.1.3 (Cesta)**:** *Cesta* délky $n > 0$ je graf G definován následovně:

$$G = (\{i \mid i \in \mathbb{N}, i \leq n\}, \{\{j, j+1\} \mid j \in \mathbb{N}, j < n\})$$

Délka cesty je počet hran tvořících cestu. O cestě s krajními vrcholy u a v mluvíme jako o cestě z u do v.

Definice 1.1.4 (Podgraf)**:** Graf H je *podgrafem* grafu G právě tehdy, když $V(H) \subseteq V(G)$ a $E(H) \subseteq E(G)$.

Definice 1.1.5 (Souvislý graf)**:** Graf $G = (V, E)$ je *souvislý* právě tehdy, když $\forall u, v \in V$ existuje podgraf G tvořící cestu z u do v.

Definice 1.1.6 (Strom)**:** Graf $G = (V, E)$ je *strom* právě tehdy, když $|E| = |V| - 1$ a G je souvislý.

Definice 1.1.7 (Stupeň vrcholu)**:** Pro vrchol v orientovaného grafu (V, E) definujeme:

- *výstupní stupeň* vrcholu v: $deg^-(v) = |\{e \in E \mid \exists u \in V : e = (v, u)\}|$
- *vstupní stupeň* vrcholu v: $deg^+(v) = |\{e \in E \mid \exists u \in V : e = (u, v)\}|$

Definice 1.1.8 (Zakořeněný strom)**:** *Zakořeněný strom* je orientovaný strom splňující:

- Existuje právě jeden vrchol v, pro který platí $deg^+(v) = 0$. Tento vrchol nazýváme *kořen* stromu.
- Pro všechny vrcholy u, krom kořene, platí $deg^+(u) = 1$.

Definice 1.1.9 (Vnitřní uzel)**:** *Vnitřní uzel* zakořeněného stromu nazýváme vrchol v, pro který platí: $deg^-(v) > 0$.

Definice 1.1.10 (List)**:** *List* zakořeněného stromu nazýváme vrchol v, pro který platí: $deg^-(v) = 0$.

Definice 1.1.11 (Potomek)**:** *Potomek* u vrcholu v v zakořeněném stromu (V, E) nazveme takový vrchol, pro který platí $(v, u) \in E$.

Definice 1.1.12 (Uspořádaný strom)**:** Zakořeněný strom, pro jehož každý vnitřní vrchol v platí, že jeho potomci tvoří uspořádanou n-tici, kde $n = \deg^+(v)$, nazveme *uspořádaným stromem*.

Definice 1.1.13 (Abeceda)**:** *Abeceda* je konečná neprázdná množina symbolů.

Abecedu obvykle značíme Σ.

Definice 1.1.14 (Řetězec)**:** *Řetězec* nad abecedou Σ je konečná posloupnost symbolů z Σ.

Množinu všech řetězců nad abecedou Σ značíme Σ^*.

Definice 1.1.15 (Formální jazyk)**:** Množinu $L \subseteq \Sigma^*$ nazýváme *formální jazyk* nad Σ.

Definice 1.1.16 (Gramatika)**:** *Gramatika* G je uspořádána čtveřice $G = (N, \Sigma, P, S)$, kde

- N je konečná množina neterminálních symbolů (neterminálů).
- Σ je konečná množina terminálních symbolů (terminálů).
- P je konečná množina přepisovacích pravidel ve tvaru $(N \cup \Sigma)^* N (N \cup \Sigma)^* \to (N \cup \Sigma)^*$.
- S je počáteční symbol, $S \in N$.

Definice 1.1.17 (Asymptotická složitost)**:** Nechť $f(n)$ a $g(n)$ jsou funkce. Řekneme, že $f(n)$ *roste asymptoticky nejvýše* tak rychle jak $g(n)$, zapisujeme $f(n) \in \mathcal{O}(g(n))$, právě tehdy když:

$$\exists c \in \mathbb{R}^+ : \exists n_0 \in \mathbb{N}^+ : \forall n \in \mathbb{N}^+ : n > n_0 \Rightarrow f(n) \leq g(n) \cdot c$$

Definice 1.1.18 (Částečné uspořádání)**:** Relaci $R \subset X \times X$ nazveme *částečným uspořádáním*, pokud platí:

- $\forall x \in X : xRx$ (reflexivita)
- $\forall x, y \in X : xRy \wedge yRx \Rightarrow x = y$ (antisymetrie)
- $\forall x, y, z \in X : xRy \wedge yRz \Rightarrow xRz$ (tranzitivita)

Definice 1.1.19 (Úplné uspořádání)**:** Řekneme, že částečné uspořádání $\preceq$ na množině X je *úplné uspořádání*, jestliže platí: $\forall a, b \in X : a \preceq b \vee b \preceq a$.

Definice 1.1.20 (Nejmenší a největší prvek)**:** Buď $\preceq$ částečné uspořádání na množině X a nechť $M \subseteq X$. Prvek $a \in M$ nazveme:

- *nejmenším prvkem* množiny M pokud: $\forall x \in M : a \preceq x$,
- *největším prvkem* množiny M pokud: $\forall x \in M : x \preceq a$.

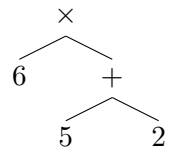
Věta 1.1.1: [1, s. 170] Buď E úplně uspořádaná množina. Každá neprázdná konečná podmnožina E obsahuje největší a nejmenší prvek.

1.2 Abstraktní syntaktický strom

Tato sekce se opírá o informace především z [2] a také z [3]. Abstraktní syntaktický strom (AST) je datová struktura (zakořeněný uspořádaný strom) používaná především pro reprezentování počítačového programu, ale i matematických výrazů nebo jiných dat popsatelných gramatikou. AST je užitečný k sémantické analýze, optimalizacím, interpretaci programu nebo pro generování kódu.

AST se skládá z vnitřních uzlů zvaných operátory, které mají za potomky, zvané operandy, další operátory nebo listy. Listy reprezentují zejména proměnné a literály. Operátory představují provedení operace nad operandy. Operátor nemusí být pouze matematický operátor ($+$, $-$, $\times$,$\div$,$>$, ...), ale i další konstrukce jazyka jako přiřazení výrazu do proměnné, příkaz (cyklus, podmíněný příkaz, volání funkce, ...), blok příkazů a další dle definované gramatiky. AST již neobsahuje pomocné syntaktické znaky jako „;“ používané k značení konce příkazu nebo závorky k pozměnění pořadí provádění operací, jelikož informaci, kterou drží ve vstupní reprezentaci, uchovává AST svou strukturou.

Průchod AST začíná v kořenovém vrcholu a rekurzivně navštěvuje své potomky ve specifikovaném pořadí. V každém vrcholu se provede operace dle typu a hodnoty vrcholu a dle výsledků průchodu operandů. Operací může být pro daný AST definováno více, jelikož může být požadováno projít AST vícekrát z různým cílem, ale v průběhu jednoho průchodu se provádějí operace jenom jednoho cíle. Rozlišování typu vrcholu záleží na implementaci, pro každý



Obrázek 1.1: Ukázka AST pro matematický výraz $6 \times (5 + 2)$

operátor lze vytvořit nový typ vrcholu nebo lze podobné operátory shlukovat stejným typem a operátory rozlišovat hodnotou vrcholu.

Obrázek 1.1 zobrazuje AST pro výraz $6 \times (5 + 2)$. Vrchol $\times$ je kořenem AST, společně s vrcholem $+$ jsou operátory. Vrcholy 6, 5 a 2 jsou listy. Lze si povšimnout, že reprezentovaný výraz obsahuje závorky určující dřívější provedení operace $5 + 2$, ale v AST se závorky nevyskytují.

1.3 Matematická analýza

Definice 1.3.1 (Potenční množina)**:** Buď X libovolná množina. *Potenční množinu* množiny X nazveme množinu $P(X)$, která obsahuje právě všechny možné podmnožiny X.

Definice 1.3.2 (Potenční funkce)**:** Buď funkce $f : X \to Y$. Funkci $g : P(X) \to P(Y)$ nazveme *potenční funkcí* funkce f, jestliže platí:

$$\forall U \in P(X) : g(U) = \bigcup_{u \in U} f(u)$$

Definice 1.3.3 (Prostá funkce)**:** Funkci f nazveme *prostou*, právě když platí:

$$\forall x, y \in D_f : f(x) = f(y) \Rightarrow x = y$$

Definice 1.3.4 (Monotonie)**:** Funkci $f : \mathbb{R} \to \mathbb{R}$ definovanou na intervalu I nazveme

- *rostoucí* na $I \iff \forall x, y \in I : x > y \Rightarrow f(x) \geq f(y)$,
- *klesající* na $I \iff \forall x, y \in I : x > y \Rightarrow f(x) \leq f(y)$.

Funkci rostoucí nebo klesající na intervalu I souhrnně nazýváme funkce *monotonní* na intervalu I.

Definice 1.3.5 (Norma)**:** Nechť V je vektorový prostor nad tělesem $\mathbb{R}$. Zobrazení $\|\cdot\| : V \to \mathbb{R}$ nazveme *normou*, jestliže platí:

- $\forall x \in V : \|x\| \geq 0, \|x\| = 0 \Rightarrow x = \theta$
- $\forall \alpha \in \mathbb{R}, \forall x \in V : \|\alpha \cdot x\| = |\alpha| \cdot \|x\|$

- $\forall x, y \in V : \|x + y\| \leq \|x\| + \|y\|$

Vektorový prostor vybavený následující normou $\|\cdot\|_2$

$$\|x\|_2 = \sqrt{\sum_{k=1}^{n} x_k^2}$$

nazýváme *euklidovský prostor*.

Definice 1.3.6 (Okolí bodu)**:** Nechť V je vektorový prostor vybavený normou $\|\cdot\|$. *Okolí bodu* $x \in V$ velikosti $\epsilon > 0$ definujeme následovně:

$$H_x(\epsilon) = \{y \in V \mid \|x - y\| < \epsilon\}$$

Definice 1.3.7 (Spojitost v bodě)**:** Řekneme, že funkce $f : \mathbb{R}^n \to \mathbb{R}$ je spojitá v bodě $a \in D_f$, jestliže platí:

$$\forall \epsilon > 0 : \exists \delta : \forall x \in D_f : x \in H_a(\delta) \Rightarrow |f(a) - f(x)| < \epsilon$$

Definice 1.3.8 (Spojitost na množině)**:** Řekneme, že funkce f je spojitá na množině $M \subseteq D_f$ právě tehdy, když f je spojitá v každém bodě množiny M vzhledem k množině M.

Definice 1.3.9 (Globální extrém)**:** Funkce $f : \mathbb{R}^n \to \mathbb{R}$ nabývá v bodě $a \in D_f$:

- *globálního maxima* jestliže $\forall x \in D_f : f(x) \leq f(a)$,
- *globálního minima* jestliže $\forall x \in D_f : f(x) \geq f(a)$.
- *lokálního maxima* jestliže $\exists \epsilon > 0 : \forall x \in \mathbb{R}^n : x \in H_a(\epsilon) \Rightarrow f(x) \leq f(a)$
- *lokálního minima* jestliže $\exists \epsilon > 0 : \forall x \in \mathbb{R}^n : x \in H_a(\epsilon) \Rightarrow f(x) \geq f(a)$

Definice 1.3.10 (Derivace)**:** Buď $f : \mathbb{R} \to \mathbb{R}$ a $x \in \mathbb{R}$. Pokud existuje limita

$$\lim_{h \to 0} \frac{f(x + h) - f(x)}{h}$$

nazveme ji *derivací* funkce f v bodě x, značíme $f'(x)$, v opačném případě řekneme, že funkce f nemá v bodě x derivaci.

Definice 1.3.11 (Parciální derivace)**:** Buď $f : \mathbb{R}^n \to \mathbb{R}$ a $a \in D_f$. Nechť $g(x_i) = f(a_1, a_2, ..., x_i, ..., a_n)$. Pokud existuje derivace $g'(a_j)$, nazýváme $g'(a_j)$ *parciální derivací* funkce f v bodě a a značíme $f_j(a)$.

Definice 1.3.12 (Omezená množina)**:** Řekneme, že množina $X \subset \mathbb{R}$ je *omezená*, jestliže platí:

$$\exists k > 0 : \forall x \in X : \|x\| < k$$

Definice 1.3.13 (Otevřená množina)**:** Množinu M nazveme *otevřenou*, právě když platí $\forall x \in M : \exists H_x : H_x \subset M$.

Definice 1.3.14 (Uzavřená množina)**:** Množinu M nazveme *uzavřenou*, právě když doplněk množiny M je otevřená množina.

Definice 1.3.15 (Hranice množiny)**:** *Hranicí* množiny M nazveme množinu:

$$\{x \in M \mid \forall H_x : \exists a, b : a \in M \wedge b \notin M\}$$

Definice 1.3.16 (Souvislá množina)**:** Množinu M na euklidovské prostoru nazveme *souvislou*, jestliže pro všechny otevřené množiny $A, B \subseteq M, A \cap B = \emptyset : M \subseteq A \cup B \Rightarrow A \cap M = \emptyset \vee B \cap M = \emptyset$.

Souvislou množinou na $\mathbb{R}$ je jednoprvková množina, prázdná množina nebo interval.

Věta 1.3.1: [4, věta č. 215] Buď $f : \mathbb{R}^n \to \mathbb{R}$ a $a \in D_f$. Pokud existuje derivace $f_i(a) \neq 0$, funkce f nemá v bodě a lokální extrém.

To však neznamená, že ostatní body musí nutně být lokálním extrémem, ale stačí se při hledání lokálních extrémů zajímat pouze o tyto body.

Věta 1.3.2: [5, věta č. 4.1] Nechť f je spojitá funkce na uzavřené a omezené množině M v euklidovském prostoru X. Pak existují body $x_{max}, x_{min} \in M$ tak, že

$$f(x_{max}) = \max_{x \in M} f(x)$$

$$f(x_{min}) = \min_{x \in M} f(x)$$

Věta 1.3.3: [5, věta č. 4.9] Každá spojitá funkce na euklidovském prostoru zobrazí souvislou množinu na souvislou množinu.

Speciálně obraz neprázdné souvislé množiny spojité funkce $f : \mathbb{R}^n \to \mathbb{R}$ je jednoprvková množina nebo interval.

Kapitola 2

Projekt Grades

Projekt Grades je webová aplikace určená pro klasifikaci studentů v jednotlivých předmětech na FIT ČVUT. Grades umožňuje učitelům definovat hodnocení předmětů a hodnotit studenty. Studenti si mohou zobrazit své hodnocení. Grades se skládají z několika modulů, pro cíle práce je důležitý zejména vyhodnocovač výrazů Expev[6], který tato práce rozšiřuje.

2.1 Vyhodnocovač výrazů Expev

Vyhodnocovač výrazů Expev je samostatná knihovna vyvíjená v jazyce Java. Pro vyhodnocení výrazů dostane Expev řetězec symbolizující výraz, který má být vyhodnocen, navíc může obdržet uživatelem dodefinované proměnné a funkce. Vstupní řetězec je převeden na tokeny (neterminály gramatiky popisující jazyk Expev) pomocí třídy `Lexer`, z kterých postupně třída `Parser` vytvoří AST. Nakonec je proveden průchod nad sestaveným AST, který nám vrátí výsledek vyhodnocení zadaného výrazu.

2.2 Jazyk Expev

Pro variabilní definice hodnocení Grades využívá vlastní programovací jazyk pro popis výrazů se syntaxí podobnou jazyku C. Jazyk je silně typovaný, všechny proměnné a výrazy mohou být pouze datových typů `Number`, `String` a `Boolean`. Datový typ `Number` je číslo v pohyblivé řádové čárce s dvojnásobnou přesností – `Double` [7], bez hodnoty `NaN`. Instance všech datových typů mohou nabývat oproti očekávaných hodnot i hodnoty `null`. Jazyk podporuje proměnné, operátory a funkce, nemá však žádné konstrukce pro iterace nebo tvoření vlastních funkcí. Podmíněný příkaz je pouze ve formě ternárního operátoru. Hodnotu proměnné lze určit pouze při deklaraci a není možné ji měnit. Není rozlišována velikost písmen názvů funkcí, datových typů a proměnných.

```
Number fahrenheit = toNumber(prefix(input, length(input) - 2));
Boolean valid = endsWith(input, '°F') && fahrenheit != NULL;
Number celsius = (fahrenheit - 32)/1.8;
String result = concat(toString(celsius), ' °C');
valid ? result : 'Nespravný vstup.'
```

Obrázek 2.1: Ukázka programu v jazyce Expev

Podporované funkce a operátory jsou uvedeny v příloze B včetně krátkého popisu. Operátory jsou seřazeny od největší po nejmenší priority, funkce jsou seskupeny dle podobnosti. Pokud parametry končí třemi tečkami, znamená to, že funkce je variadická a může následovat libovolný počet argumentů stejného datového typu jaký je poslední uvedený. Parametr nebo návratový typ „výraz“, znamená libovolný podporovaný datový typ, avšak v rámci jednoho použití je každý „výraz“ stejného datového typu. Pokud vyhodnocení funkce pro dané argumenty nemá dobrý smysl je výsledkem hodnota `null`, např. pro funkce `mod` a `div`, pokud je dělitel roven 0 nebo pro funkci `pow(řetězec, číslo)` a záporný druhý argument.

Prostředí pro definování hodnocení rozděluje proměnné na 3 typy dle způsobu zadání – ručně, vzorcem a agregovaně. Vzorce využívají jazyku hodnocení, agregace pouze jednu funkci. Prostředí umožňuje omezit možné hodnoty proměnných typu řetězec a číslo. Ručně zadávané řetězce mohou být omezeny výčtem možných hodnot, u číselných proměnných mohou být specifikované orientační minimální a maximální hodnoty (a minimální požadovanou hodnotu – pro splnění požadavků předmětu). Tyto rozdělení a omezení se vyskytují pouze v prostředí pro definování hodnocení Grades, nikoliv ve vyhodnocovači Expev. Prostředí dále rozšiřuje jazyk Expev o vlastní funkce.[8]

Na obrázku 2.1 je výpis kódu algoritmu jazyka Expev, který obdrží v proměnné `input` teplotu ve stupních Fahrenheita a vrátí teplotu ve stupních Celsia. Očekávaný formát proměnné `input` je číslo následované jednotkou „°F“, v takovém případě je výsledek v obdobném formátu (až na jednotku – „°C“). V opačném případě je vrácen řetězec „Nesprávný vstup.“.

2.3 AST v Expev

Každý vrchol AST je v Expev reprezentován třídou `Node`. `Node` je abstraktní třída, která definuje a deklaruje společné proměnné a metody napříč třídami pro jednotlivé druhy uzlů AST. Samotný průchod AST se provádí pomocí metody `evaluateNode`, která má vstupně výstupní parametr slovník z názvu proměnné na její hodnotu. Na začátku tento slovník obsahuje definice vstupních proměnných a postupně se do něj přidávají hodnoty lokálně definovaných

proměnných, ale hodnoty proměnných se neupravují. Výsledkem volání metody `evaluateNode` je hodnota podléhající datovému typu `DataType`(`String`, `Boolean`, `Number`). Jednotlivé třídy, které dědí z třídy Node, jsou následující:

- `BlockNode` agreguje posloupnost deklarací proměnných ukončenou výrazem. Pro vyhodnocování se postupně vyhodnotí vrcholy deklarací, následně ukončujícího výrazu, jehož hodnota je vrácena jako výsledek vyhodnocení tohoto vrcholu.
- `DeclarationNode` deklaruje proměnou jejíž hodnota je definována výrazem. Při vyhodnocení se nejdříve vyhodnotí vrchol reprezentující výraz, jehož výsledek se přidá jako asociovaná hodnota k názvu proměnné do slovníku.
- `FunctionNode` reprezentuje volání funkce, jejíž argumenty jsou potomci vrcholu. Nejdříve se vyhodnotí všechny operandy a následně vrátí výsledek aplikování funkce na hodnoty. Funkce je reprezentována třídou `Function`. Počet potomků vrcholu odpovídá počtu parametrů každé funkce. Počet parametrů funkce může být buďto variabilní, anebo fixní.
- `LiteralNode` je list obsahující konstantu, jejíž hodnota je výsledkem vyhodnocení tohoto vrcholu.
- `OperatorNode` je speciální případ `FunctionNode`, ze kterého dědí, reprezentuje operátory. [1]
- `VariableNode` je list reprezentující použití proměnné, vyhodnotí se vrácením hodnoty ze slovníku pro jméno proměnné, které si vrchol udržuje.

[1] Operátor má oproti funkci rozdílnou syntaxi, ale ne sémantiku.

Kapitola 3

Návrh výpočetního průchodu

Od výpočetního průchodu pro omezení hodnot výrazu je očekáváno následující chování: Průchod obdrží výraz ve formě AST a množinu proměnných s jejich možnými hodnotami. Výsledek průchodu je konzervativní, to znamená, že výsledkem je množina hodnot, která obsahuje všechny ohodnocení, které by vznikly ohodnocením výrazu standardním způsobem pro všechna možná ohodnocení proměnných, které splňují zadané omezení. Výsledek může obsahovat prvky navíc, ale cílem je, aby byl co nejpřesnější. Pro reálné použití je požadováno, aby výpočet průchodu netrval neúměrně dlouho vzhledem k velikosti AST, proto je potřeba umožnit omezení počtu kroků provedených v každém vrcholu na úkor přesnosti pomocí parametru průchodu. Toto omezení musí zachovat konzervativnost výsledku.

AST, nad kterým je prováděn průchod, musí být schopen reprezentovat výraz, tedy obsahuje vrcholy pro konstanty, proměnné (listy) a vnitřní vrcholy, které reprezentují provedení operace (funkce) nad operandy, dále nazývané funkční vrcholy. Od funkčních vrcholů se očekává, že reprezentované funkce nemodifikují hodnoty proměnných a jsou bez jakýkoliv dalších vedlejších efektů. Jazyk Expev navíc umožňuje, aby výraz obsahoval deklarace lokálních proměnných, navržený průchod toto zohledňuje pomocí vrcholů deklarace a bloku. Další možné rozšíření jazyka popisující výrazy, už zohledněné není.

Možné hodnoty proměnných a výsledků ohodnocení výrazů jsou uchovávané v množinových datových strukturách (MDS), více jsou popsané v sekci 3.1. Průchod si udržuje deklarované proměnné s odpovídajícími MDS.

Dle typu vrcholů se výpočetní průchod provede následovně:

- Proměnná – vrátí se MDS pro proměnnou.
- Konstanta – vyhodnotí se na MDS obsahující pouze hodnotu konstanty.
- Funkce – pro každý typ vrcholu, tzn. každou funkci jazyka, je třeba, aby byla vytvořena potenční funkce. Parametry potenční funkce jsou MDS

odpovídající datovému typu parametru u původní funkce. Pro vyhodnocení vrcholu se nejdříve vyhodnotí všechny možné hodnoty potomků (provede se nad nimi průchod), na obdržené MDS se aplikuje potenční funkce vrcholu.

- Deklarace – vyhodnotí se vrchol pro získání množiny možných hodnot proměnné a uloží se společně s identifikátorem do průchodem udržovaných proměnných.

- Blok – vyhodnotí se všichni potomci vrcholu, vrátí se výsledek posledního z nich.

Aby bylo zajištěno, že celkové ohodnocení je konzervativní, musí být konzervativní vyhodnocení každého vrcholu. Vyhodnocení vrcholu pro proměnnou a konstantu je konzervativní, jelikož se vyhodnotí přesně na hodnoty, které proměnná může nabývat nebo přímo na hodnotu konstanty. Konzervativnost vyhodnocení funkčních vrcholů není zaručená a závisí na konzervativnosti každé potenční funkce. Vyhodnocení vrcholu deklarace proměnné je konzervativní, jelikož se proměnné přiřadí její konzervativní odhad. Vyhodnocení všech potomků vrcholu bloku je konzervativní, tedy i vyhodnocení vrcholu bloku je konzervativní, jelikož nijak nemodifikuje možné hodnoty.

Omezení počtu kroků při vyhodnocování funkčního vrcholu lze provést kontrolováním, kolik unikátních variací argumentů bylo vyhodnoceno. Pokud počet vyhodnocení překročí mez určenou parametrem průchodu, tak se zastaví vyhodnocování vrcholu a jako výsledek vyhodnocení se vrátí celý obor hodnot funkce (nebo nejmenší reprezentovatelná nadmnožina). Obor hodnot funkce jistě obsahuje všechna možná ohodnocení funkce při daných omezení argumentů a tedy je konzervativní. Obecně nestačí kontrolovat počet unikátních výsledků, protože existují funkce, které vyhodnotí různé argumenty na stejný výsledek (např. konstantní funkce). Ostatní vrcholy není nutné omezovat, protože jejich složitost vyhodnocení je konstantní (nepočítá se počet vyhodnocení potomků).

3.1 Návrh datových struktur

Datové struktury, reprezentující možné hodnoty proměnných a vyhodnocení vrcholu, vyžadují operace podobné množinám, proto budou dále nazývány množinové datové struktury, zkráceně MDS. Od MDS je požadováno přidávání prvků určeného datového typu a následné postupné získání všech vložených prvků bez duplicit. Jelikož se každý vrchol AST vyhodnocuje pouze jednou a vyhodnocení vrcholu a proměnné se neupravuje, je postačující umožnění vložení všech prvků do MDS pouze před zobrazením prvků. Získání prvků musí být umožněno vícekrát z důvodu možného několikanásobného výskytu proměnné.

Omezení hodnot každého datového typu lze provést buďto výčtem všech možných hodnot, anebo seznamem libovolných konstrukcí, kde každá konstrukce popisuje podmnožinu možných hodnot, jejichž sjednocení obsahuje všechny možné hodnoty a jejich průnik je prázdná množina.

Pro datové typy řetězec, číslo a boolean mohou vypadat MDS následovně:

- Boolean může nabývat pouze 2 hodnot – `true` a `false`, je tedy ideální použití výčtu pro uchovávání možných hodnot.
- Řetězec je nad konečnou abecedou, ale nemá omezení na délku. Vhodné řešení závisí na způsobu omezení vstupních proměnných a požadavcích na omezení výsledku. Pokud vstupní proměnné jsou omezeny výčtem hodnot nebo nemají žádné omezení a obdobné je očekávání i od výsledku, je omezení výčtem stále vhodné, i když množina všech řetězců není konečná. Další možnou reprezentací pro řetězce jsou regulární výrazy.
- Číslo může nabývat různých hodnot dle implementace. Pro tuto práci bude uvažováno pouze čísel s konečnou přesností, jelikož aplikace, do které bude průchod implementován tuto reprezentaci využívá. Omezení čísel může být provedeno pomocí konstrukcí podobné intervalu obsahující všechna čísla od minimálního po maximální prvek na daném datovém typu (je povolen interval nulové délky).

Pro nekonečné množiny reprezentované výčtem, je třeba zavést speciální reprezentaci stavu, kdy MDS obsahuje všechny možné hodnoty. Operace na funkčním vrcholu nemusí být definována, např. $\frac{0}{0}$. Z toho důvodu je třeba u každé MDS mít speciální značení pro nedefinovanou hodnotu.

3.2 Vyhodnocení vybraných funkcí

V první části této sekce bylo uvedeno, že pro každou funkci v standardním vyhodnocení je potřeba vytvořit potenční funkci, která popisuje všechna možná ohodnocení původní funkce. Vytvoření těchto funkcí lze usnadnit seskupením na základě stejného způsobu ohodnocení. Pro každou takovou skupinu se vytvoří pouze jedna potenční funkce, která vybere hodnoty pro vyhodnocení původní funkcí a ze získaných hodnot zkonstruuje odpovídající MDS. Dané skupiny funkcí se mohou lišit dle využité MDS. Následující rozdělení funkcí je vytvořeno na základě MDS uvedených v předchozí sekci a pro vybrané funkce z vyhodnocovače Expev uvedené v příloze B.

3.2.1 Číselné funkce

Tato podsekce se zabývá funkcemi, které splňují předpis $f : \mathbb{R}^n \to \mathbb{R}$. Množina vymezená v každé složce intervalem je souvislá, věta 1.3.3 říká, že pokud je

funkce spojitá, tak obraz této množiny bude také souvislá množina – na $\mathbb{R}$ interval nebo jednoprvková množina. Pro funkce, které jsou spojité na podmnožinách definičního oboru, lze zobrazované souvislé množiny rozdělit tak, že na každé z nich je funkce spojitá a tedy obraz každé z nich je souvislá množina. Výsledek potenční funkce je sjednocení výsledných množin.

Pro získání výsledného intervalu spojité funkce na souvislé uzavřené a omezené množině stačí nalézt minimum a maximum možných hodnot. Věta 1.3.2 říká, že takto zvolené množině existuje minimum i maximum a věta 1.3.1 napovídá kde lokální extrémy funkce hledat. Maximum a minimum na dané množině se může nacházet buďto v lokálním extrému funkce nebo na hranici množiny.

Při použití číselných datových typů s konečnou přesností se stačí zaměřit pouze na uzavřené a omezené množiny. Otevřenou množinu vymezenou intervaly lze nahradit její největší uzavřenou podmnožinou. Množina čísel daného datového typu je konečná a tvoří úplné uspořádaní, tedy má největší a nejmenší prvek a je omezená. Pokud datový typ umožňuje reprezentovat hodnotu $\pm\infty$, je třeba se věnovat pozornost tomu, kdy je výraz nedefinovaný.

Funkce monotonní na konečném počtu intervalů

Pro vyhodnocení stačí danou funkci rozdělit na monotonní intervaly dle průběhu funkce, provést průnik s intervaly v argumentu, pomocí čehož se vyřadí argumenty mimo definiční obor funkce a ostatním intervalům se přiřadí monotonie. Dle monotonie funkce na intervalu je vyhodnocení následující.

- Funkce f je na intervalu $[x, y]$ rostoucí: $f([x, y]) = [f(x), f(y)]$.
- Funkce f je na intervalu $[x, y]$ klesající: $f([x, y]) = [f(y), f(x)]$.

Do této skupiny patří funkce `uminus`, `uplus`, `abs`, `sqrt`, `log`, `log10`, `asin`, `acos`, `atan`, `sinh`, `cosh` a `tanh`.

Periodické funkce

Pokud lze periodu funkce rozdělit na monotonní intervaly, lze postupovat obdobně jako u monotonních funkcí, kdy každý interval v argumentu je rozdělen na monotonní intervaly a z vyhodnocení krajních bodů jsou získány krajní body výsledného intervalu. Pokud libovolný interval v argumentu je minimálně stejně dlouhý jako perioda funkce, lze jako výsledek vyhodnocení vrátit celý obor hodnot funkce bez nutnosti počítání obrazů funkce jednotlivých intervalů, jelikož všechny hodnoty, které může periodická funkce nabývat, se nachází na jedné periodě. Tuto optimalizaci lze použít i pro funkce, jejichž perioda není složena z monotonních intervalů.

Periodickými funkcemi jsou `sin`, `cos` a `tan`.

Funkce monotonní pro každý parametr

Spojité funkce s n parametry splňující $\forall i \in \mathbb{N}, 1 \leq i \leq n, \forall a_1, a_2, ..., a_n \in \mathbb{R}$:

$$\forall x, y \in \mathbb{R} : x > y \Rightarrow f(a_1, ..., a_{i-1}, x, a_{i+1}, ..., a_n) \geq f(a_1, ..., a_{i-1}, y, a_{i+1}, ..., a_n)$$
$$\vee$$
$$\forall x, y \in \mathbb{R} : x > y \Rightarrow f(a_1, ..., a_{i-1}, x, a_{i+1}, ..., a_n) \leq f(a_1, ..., a_{i-1}, y, a_{i+1}, ..., a_n)$$

Pro argumenty ze zadaných intervalů se největší hodnota vzhledem k i-tému argumentu nachází v jeho maximu, pokud je funkce pro i-tý argument rostoucí, jinak v jeho minimu. Obdobně se nejmenší hodnota funkce vzhledem k i-tému argumentu nachází v minimu, respektive v maximu pokud je funkce klesající v pro i-tý argument. Celkové maximum funkce se získá vyhodnocením bodu, kde vzhledem ke každému argumentu funkce nabývá maxima, podobně pro minimum. Pro každou souvislou množinu určenou intervaly je potřeba zjistit minimum a maximum zvlášť.

Do této skupiny patří funkce `min`, `max`, `avg` a `sum`, které jsou rostoucí pro každý parametr a vyhodnotí se následovně:

$$f([x_1, y_1], [x_2, y_2], ..., [x_n, y_n]) = [f(x_1, x_2, ...x_n), f(y_1, y_2, ...y_n)]$$

Dále sem patří funkce `minus`, která je v prvním argumentu rostoucí a v druhém klesající. Její vyhodnocení je následující:

$$[x_1, y_1] - [x_2, y_2] = [x_1 - x_2, y_1 - y_2]$$

Zaokrouhlení na celá čísla

Obor hodnot těchto funkcí jsou pouze celá čísla, zároveň zaokrouhlení celého čísla je to samé číslo. V obrazu intervalu se nachází všechna celá čísla zevnitř intervalu a zaokrouhlení krajních bodů.

Do této skupiny patří funkce `ceil`, `floor`, `round(číslo)`.

Ostatní spojité funkce

Vyhodnocení ostatních spojitých funkcí je pro každou funkci provedeno zvlášť kvůli jejich malému počtu a odlišnosti od ostatních funkcí.

- `product`: Funkce může mít libovolný počet argumentů, ale díky asociativitě lze funkci vyhodnocovat postupně.

  $$[x_1, y_1] \cdot [x_2, y_2] = [\min(x_1x_2, x_1y_2, y_1x_2, y_1y_2), \max(x_1x_2, x_1y_2, y_1x_2, y_1y_2)]$$

- `div`: Funkce se vyhodnotí na každém kvadrantu zvlášť. Pokud je dělitel roven 0, tak je výsledek nedefinovaný.

– 1. kvadrant ($\forall x_1 \geq 0, x_2 > 0$):

$$\frac{[x_1, y_1]}{[x_2, y_2]} = \left[\frac{x_1}{y_2}, \frac{y_1}{x_2}\right]$$

– 2. kvadrant ($\forall x_1 \leq 0, x_2 > 0$):

$$\frac{[x_1, y_1]}{[x_2, y_2]} = \left[\frac{x_1}{x_2}, \frac{y_1}{y_2}\right]$$

– 3. kvadrant ($\forall x_1 \geq 0, x_2 < 0$):

$$\frac{[x_1, y_1]}{[x_2, y_2]} = \left[\frac{y_1}{y_2}, \frac{x_1}{x_2}\right]$$

– 4. kvadrant ($\forall x_1 \leq 0, x_2 < 0$):

$$\frac{[x_1, y_1]}{[x_2, y_2]} = \left[\frac{y_1}{x_2}, \frac{x_1}{y_2}\right]$$

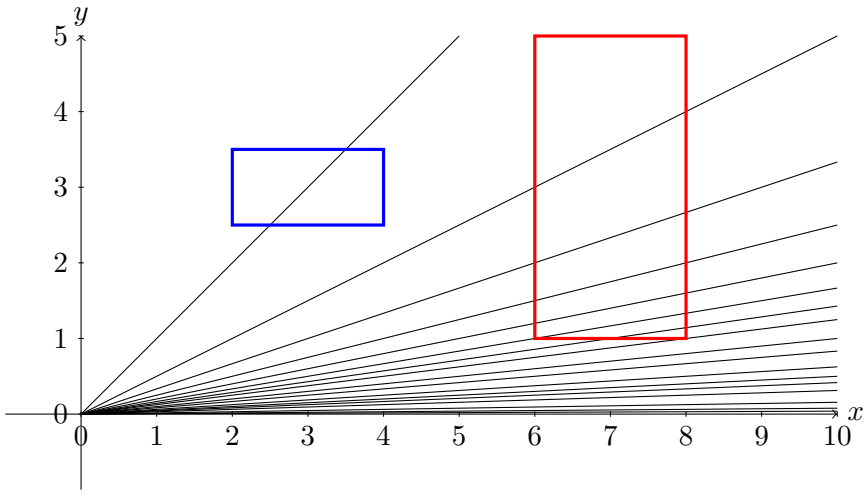
Ostatní číselné funkce

Tyto funkce mají unikátní chování vzhledem k vybraným funkcím a je třeba vyhodnotit každou zvlášť.

- `round`: Obor hodnot této funkce jsou čísla s konečným desetinným rozvojem. Součástí možných výsledků jsou všechna čísla z prvního argumentu, jejichž počet číslic desetinného rozvoje je menší než maximum z druhého argumentu. Navíc se ve výsledku můžou nacházet čísla vzniklá zaokrouhlením krajních bodů na různý počet desetinných míst.

- `pow`: Pro záporný základ je funkce nespojitá, je nutné vyhodnotit funkci na intervalu pro každý celočíselný exponent zvlášť. Pro nezáporný základ se funkce chová spojitě, tedy postačuje nalezení minima a maxima na každé souvislé množině omezené argumenty.

- `mod`: Zbytek čísla x po dělení y je roven 0, pokud x je násobkem y. Souřadnice těchto bodů tvoří přímky $p_k : y \cdot k = x; k \in \mathbb{Z}$. Přímky rozdělují rovinu $\mathbb{R}^2$ na souvislé množiny, s výjimkou $k = 0$, lze pozorovat na obrázku 3.1. Na souvislých množinách je funkce ve směru x rostoucí a ve směru y je v 1. a 3. kvadrantu klesající a v 2. a 4. kvadrantu rostoucí. Na množinách ohraničených přímkami p_k stačí zjistit y souřadnici průsečíku y_p přímek se vzdálenějšími stranami obdélníku vymezeného intervaly pro x a y. Možné hodnoty na dané množině vytyčené přímkami p_k je interval $[0, y_p)$ (respektive $(-y_p, 0]$ pro x < 0). Pro krajní části obdélníků je



Obrázek 3.1: Vizualizace funkce mod

Na zobrazených přímkách platí rovnost $x \mod y = 0$. Vyznačené obdélníky ukazují dva z možných případů pro vyhodnocení vzhledem k přímkám. Levý obdélník je vymezen $x \in [2, 4]$ a $y \in [2.5, 3.5]$, pravý obdélník $x \in [6, 8]$ a $y \in [1, 5]$.

třeba zjistit hodnotu funkce v jeho rozích. Díky opakujícím se hodnotám funkce stačí vyhodnotit pouze souvislou množiny kolem přímky p_k s nejnižším indexem v absolutní hodnotě.

Na obrázku 3.1 lze vidět první kvadrant s přímkami p_k, kde je výsledek funkce roven 0. Pro modře ohraničenou množinu omezenou $x \in [2, 4], y \in [2.5, 3.5]$ je třeba vyhodnotit funkci v levém horním rohu a pravém spodním rohu obdélníku $mod(2, 3.5) = 2, mod(4, 2.) = 1.5$, a jelikož přímka p_1 protíná vrchní stranu obdélníku je množina nalevo od přímky shora omezená 4. Množina možných hodnot funkce na modře ohraničené množině je $[0, 1.5] \cup [2, 4)$. Červeně ohraničená množina ukazuje, že stačí vyhodnotit pouze první 2 množiny oddělené přímkami, jelikož jejich sjednocení je interval.

3.2.2 Funkce pro výčtové typy

Funkce v této skupině má parametry v datovém typu, pro který je MDS formou výčtu. Vybrané funkce jsou vyhodnoceny pro všechny možné instance argumentů, jelikož se při omezení argumentů mohou v jejich každé instanci vyhodnotit vždy na unikátní hodnotu nezávisle na počtu argumentů a počtu možných hodnot argumentů.

Příkladem je funkce `concat` s variadickým počtem parametrů – řetězců. Nechť každý argument je omezen na řetězce ve tvaru číslo, následované oddě-

lovacím symbolem stejným pro všechny řetězce. Každé dvě ohodnocení takto zkonstruovaných argumentů, které se liší alespoň v jednom argumentu, jsou rozdílné. Pokud jsou dvě volání funkce rozdílné v i-tém argumentu, tak se musí lišit i ve výsledném řetězci mezi i-tým a předcházejícím oddělovačem (respektive začátkem řetězce).

Celkový počet vyhodnocení se rovná $\prod_{i=1}^{n} k_i$, kde k_i je počet možných hodnot v i-tém argumentu n je počet parametrů funkce. Pro některé funkce může být počet argumentů nutných k vyhodnocení řádově menší, např. funkce `and` a `or`, proto nejsou součástí této skupiny a vyhodnocují se zvlášť.

Do této skupiny patří funkce `eq`, `neq`, `neg`, `toString(boolean)`, `toNumber`, `toBoolean(řetězec)`, `concat`, `length`, `contains`, `startsWith`, `endsWith`, `matches`, `toLowerCase`, `toUpperCase`, `findSubstring`.

3.2.3 Funkce kombinující výčtové typy a čísla v argumentu

Pro funkce v této skupině se předpokládá, že pro získání všech možných vyhodnocení funkce stačí, když číselné parametry budou nabývat pouze celých čísel. Možné číselné hodnoty lze u některých funkcí omezit navíc dle hodnot ostatních parametrů, např. délkou řetězce. Stejně jak u funkcí pouze s výčtovými parametry je třeba vyhodnotit všechny možné varianty argumentů.

Do této kategorie patří funkce `substring`, `prefix`, `suffix`, pro které mohou být parametry omezeny zdola 0 a shora délkou řetězce, dále sem patří funkce `pow(řetězec, číslo)`, kterou lze omezit pouze zdola pomocí 0.

3.2.4 Ostatní funkce

Relace úplného uspořádání

Funkce mají za parametry 2 čísla, vrací boolean a reprezentují úplné uspořádání. Pokud existují dva prvky z odpovídajících množin, které jsou v relaci, tak nejmenší prvek dle uspořádání z první množiny je v relaci s největším prvkem dle uspořádání druhé množiny. Analogicky pokud existují dva prvky, které nejsou v relaci. Pro získání všech ohodnocení stačí porovnat největší číslo s nejmenším a nejmenší číslo s největším z odpovídacích množin.

Do této kategorie patří funkce `gt`, `gte`, `lt`, `lte`.

Ostatní funkce s unikátním vyhodnocením

- `rand`: Výsledek může být libovolná hodnota z intervalu $[0, 1)$, je tedy tento interval výsledkem rozsahového vyhodnocení.

- `if`: Pokud první argument může nabývat hodnoty `true` přidá se druhý argument do množiny možných výsledků, pokud může nabývat hodnoty `false`, přidá se třetí argument.

- `and`: Aby výsledek obsahoval `true`, musí i každý argument obsahovat `true`. Pro hodnotu `false` je postačující aby pouze jeden argument mohl nabývat `false`.

- `or`: Stejně jako funkce `and`, až na prohození `true` a `false`.

- `toBoolean(číslo)`: Jestli je povolený argument 0, výsledek obsahuje `false`, jakýkoliv jiný argument znamená možnost výsledku `true`.

- `toString(číslo)`: Převést každé číslo z intervalu na řetězce reprezentované výčtem nemusí být dle reprezentace čísel možné. Pokud tomu tak je a možné argumenty obsahují interval čísel, využije se hodnoty pro reprezentace všech řetězců se ztrátou přesnosti vyhodnocení. V opačném případě se vyhodnotí každé číslo zvlášť.

- `eq(číslo, číslo)`: Hodnota `true` je ve výsledku, pokud existuje průnik množin argumentů. Funkce se může vyhodnotit na `false`, pokud alespoň jedna z množin má alespoň dva prvky, nebo obě množiny mají pouze jeden prvek, který je odlišný.

- `neq(číslo, číslo)`: Negace výsledků funkce `eq`.

3.3 Výhody a nevýhody řešení

Výhody řešení pomocí průchodu AST jsou v požadovaných vlastnostech, řešení je konzervativní a dokáže pracovat s různými datovými typy. Vyhodnocení však nemusí být v některých případech přesné, jak by mohlo. Například pro proměnou x, která je omezená množinu čísel $X = \{1, -1\}$, se výraz $x + x$ vyhodnotí na množinu $\{-2, 0, 2\}$. Množina možných výsledků obsahuje navíc číslo 0, protože není zohledněn vícenásobný výskyt proměnné ve výrazu. Průchod nad každým operátorem pracuje pouze s možnými hodnotami operandů, ale nebere v potaz jejich závislosti. V některých případech lze výraz upravit do tvaru, pro který bude vyhodnocení přesné. Uvedený výraz $x+x$ lze převést na $2 \cdot x$, jehož vyhodnocení neobsahuje žádné prvky navíc, obsahuje pouze -2 a 2.

Další vlastnosti průchodu se odvíjejí od použitých MDS pro reprezentaci možných hodnot a od typů vrcholů AST a jejich vyhodnocování. Pro různé MDS, byť stejného datového typu, může být rozdílná přesnost výsledku a doby trvání průchodu AST. Výčet hodnot nemusí být schopen reprezentovat obor hodnot funkce, takže se musí použít reprezentace pro všechny možné hodnoty universa. S vhodnou datovou strukturou lze tuto nepřesnost eliminovat. Je tedy důležité při návrhu průchodu pro daný účel vhodně zvolit implementaci MDS.

3.3.1 Porovnání s existujcími řešeními

Analýza datových toků

Analýza datových toků (anglicky data-flow analysis) se zabývá aproximací možných provedeních programů. Pro zjištění možných hodnot proměnných a výrazů se používají algoritmy *Meet Over all Paths*(MOP) a *Maximal Fixed Point* (MFP). V těcto algoritmech je program reprezentován jako orientovaný graf, kde vrcholy reprezentují příkazy nebo seskupení příkazů a hrana mezi vrcholy znamená, že sekvence příkazů se mohou provést neprodleně za sebou. Algoritmus MOP se snaží vyhodnotit výrazy pro všechny možné cesty v sestrojeném grafu. Při provádění algoritmu MFP se postupně upravují hodnoty přiřazené vrcholům grafu dle aplikování přechodové funkce na možné hodnoty předchůdců vrcholu. Výsledkem obou algoritmů je konzervativní odhad. [9]

Jazyk Expev svými konstrukty neumožňuje žádné větvení řídícího toku programu, popsaný graf libovolného programu v tomto jazyku je cesta. Zároveň není třeba řešit konvergenci algoritmu jako celku, ale stačí zajistit konvergenci vyhodnocení každého vrcholu zvlášť.

Matematická optimalizace

Matematická optimalizace se věnuje minimalizaci a maximalizaci funkce za daných podmínek pro proměnné. Existuje mnoho algoritmů pro numerické hledání extrémů. Algoritmy začnou počátečním odhadem, který iterativně zlepšují dokud nezkonverguje k hodnotě. Např. gradientní sestup postupuje vždy ve směru největšího poklesu funkce $(-\nabla f)$ pro nalezení minima. Výsledek gradientního sestupu i dalších algoritmů nemusí být globální extrém. Naopak např. simplexový algoritmus najde globální extrém, ale lze jej použít jen na funkce ve specifickém tvaru. [10]

Výrazy jazyka Expev neobsahují pouze čísla, ale i další datové typy, pro které metody matematické optimalizace nejsou vhodné. Navržené vyhodnocení funkcí neprobíhá iterativním způsobem, ale byly analyticky určeny body, kde se má funkce vyhodnotit pro získání extrémů, což přispívá k rychlosti řešení.

Kapitola 4

Realizace

Tato kapitola se zabývá implementací navrženého průchodu v kapitole 3 do modulu Expev aplikace Grades. Nejdříve jsou popsány použité datové struktury, které implementují MDS popsané v sekci 3.1. Následuje popis implementace navrženého průchodu a jeho ukázka. Nakonec se kapitola věnuje otestování implementace.

4.1 Použité datové struktury

V této sekci jsou popsány datové struktury, které byly použity pro reprezentaci množin hodnot, kterých můžou datové typy jazyka Expev nabývat.

4.1.1 Interval

Třída reprezentující uzavřený interval definovaný jeho krajními body. Oproti standardní matematické definici je možné mít interval, jehož krajní body jsou stejné, v tomto případě interval reprezentuje pouze jediný prvek. Od intervalu je potřeba, aby mohl udržovat hodnoty, kterých může nabývat datový typ `Number` jazyka Expev. Z toho důvodu jsou krajní body intervalu datového typu `double` stejné jako vnitřní reprezentace `Number`. Ze stejného důvodu je i povoleno, aby krajními body intervalu mohli být pozitivní i negativní nekonečno, přičemž to znamená, že jsou povolené i intervaly $[+\infty, +\infty]$ a $[-\infty, -\infty]$. Avšak hodnota `NaN` není povolena jako krajní bod intervalu.

Třída umožňuje nalezení průniku intervalů, zkonstruování nejmenšího intervalu obsahující dva intervaly, porovnání intervalu a další. Porovnání intervalů nejdříve porovná levou a pak pravou mez.

Z třídy `Interval` dědí třída `MonotonyInterval`, která slouží k vyjádření monotonie funkce na daném intervalu.

4.1.2 Container

`Container`, dále též kontejner, je generický interface pro MDS popsané v sekci 3.1, který deklaruje následující metody:

- `add`: Přidá prvek do kontejneru, který není `null`.
- `setCanBeNull`: Nastaví, zda kontejner obsahuje hodnotu `null`.
- `canBeNull`: Zjistí, zda kontejner může obsahovat hodnotu `null`.
- `setAllPossible`: Nastaví, že kontejner obsahuje všechny prvky, kterých může příslušný datový typ nabývat.
- `isAllPossible`: Zjistí, zda kontejner obsahuje všechny prvky, kterých může příslušný datový typ nabývat.
- `contains`: Zjistí, zda se daný prvek nachází v kontejneru.
- `createNew`: Vytvoří novou instanci stejné třídy, která neobsahuje žádné prvky.

Přístup k prvkům je definován přes interface `Iterable`, který `Container` rozšiřuje. Od iterace prvků je očekáváno, že žádný prvek uveden vícekrát. Metody `setCanBeNull` a `canBeNull` jsou implementovány třídou `Nullable`, z které jednotlivé implementace vyjma `NullContainer` dědí.

4.1.3 IntervalContainer

`IntervalContainer` je interface rozšiřující `Container` pro `Interval` a slouží k uchovávání množiny čísel. Od iterace intervalů se očekává, že každá dvojice intervalů bude mít prázdný průnik. Pro toto rozhraní bylo vytvořeno několik implementací s různými vlastnostmi.

IntervalBounds

Třída `IntervalBounds` uchovává pouze minimální a maximální mez vložených intervalů, reprezentováno taktéž třídou `Interval`. Pro první přidání se meze nastaví dle hranice intervalů, další vložení pouze rozšiřují dané hranice. Iterace všech prvků je jenom získání jednoho intervalu určující meze. Díky jedinému uchovávanému intervalu jsou všechny operace nad datovou strukturou prováděny v konstantním čase. Cenou za rychlost je nepřesnost datové struktury. Datová struktura může obsahovat čísla, které nebyly obsaženy v přidaných intervalech.

IntervalTree

Třída `IntervalTree` uchovává intervaly v kolekci `TreeSet`. Při vkládání intervalu je nejdříve zjištěn jeho následovník a předchůdce. Pokud mezi nimi existuje průnik, odstraní se daný interval a sjednotí se s vkládaným intervalem. Tato operace se opakuje, dokud existuje průnik s následovníkem nebo předchůdcem, teprve potom se vloží interval do kolekce. Nalezení předchůdce a následovníka a odstranění prvku z kolekce `TreeSet` má časovou složitost $\mathcal{O}(\log n)$ [7]. V jednom vložení může mít vkládaný interval průnik se všemi intervaly v kolekci, v nejhorším případě je tedy složitost přidání intervalu do kontejneru $\mathcal{O}(n \log n)$. Interval vložený do kolekce z ní může být odstraněn pouze jednou, tedy amortizovaná složitost přidání intervalu do kontejneru je $\mathcal{O}(\log n)$.

IntervalList

Tato implementace množiny intervalů využívá toho, že při průchodu AST se do výsledného `Container` pouze vkládá a při použití v dalších vrcholech se z něj již jenom získávají prvky a nikdy se nepřidávají. Vnitřně se prvky uchovávají pomocí kolekce `ArrayList`. Přidání prvku nekontroluje, zda přidávaný prvek má nějaký průnik s ostatními intervaly, pouze se uloží prvek na konec kolekce, což má konstantní časovou složitost [7]. Když nastane první přístup k uchovávaným intervalům, seřadí se všechny intervaly. Následně se projdou intervaly v pořadí a sjednotí se ty, které mají průnik. Teprve potom se umožní přístup k intervalům. Seřazení intervalů má asymptotickou složitost $\mathcal{O}(n \log n)$, sjednocení intervalů $\mathcal{O}(n)$, tedy dohromady $\mathcal{O}(n \log n)$. Další řazení a sjednocení intervalů není potřeba, pokud nejsou přidány další prvky, každá další iterace prvků je v takovém případě možná v čase $\mathcal{O}(n)$.

4.1.4 StringContainer

Nadstavba nad množinou řetězců implementovanou kolekcí `HashSet`. Navíc `StringContainer` obsahuje příznak, zda představuje množinu všech řetězců. Při nastavení tohoto příznaku není možné iterovat přes prvky množiny. Složitosti operací se odvíjí od složitostí kolekce `HashSet`, vložení prvku je $\mathcal{O}(1)$[7].

4.1.5 BooleanContainer

Kontejner pro datový typ boolean. Pro hodnoty `true` a `false` má kontejner příznak, zda jsou v něm obsaženy.

4.1.6 NullContainer

Reprezentuje množinu, která obsahuje pouze hodnotu `null`. Využívá se při vyhodnocení `LiteralNode`, který reprezentuje danou konstantu. Program udr-

žuje pouze jednu instanci třídy `NullContainer`, jelikož třída neudržuje žádný vnitřní stav.

4.2 Implementace průchodu

Průchod pro omezení hodnot výrazu má velmi podobné rozhraní jako stávající průchod pro vyhodnocení výrazu. Třída `Node` je rozšířena o nový průchod pomocí metody `rangeEvaluateNode`. Metoda má 2 parametry, slovník ze jména definovaných proměnných na množinu hodnot, kterých může proměnná nabývat a doplňující parametry průchodu specifikované pomocí třídy `RangeEvaluationInfo`. Možné hodnoty proměnných i výsledku metody jsou uchovávány v datové struktuře `Container`.

Pomocí třídy `RangeEvaluationInfo` lze průchodu specifikovat maximální počet vyhodnocení na jednom vrcholu, a která implementace rozhraní `Container` se využije pro uchovávání čísel.

Třídy dědící z `Node` vykonávají metodu `rangeEvaluateNode` následovně:

- `BlockNode`: Metoda se zavolá na všechny potomky, vrátí se výsledek volání metody na posledního potomka.
- `DeclarationNode`: Vyhodnotí se vrchol reprezentující pravou stranu přiřazení a výsledný kontejner se uloží do slovníku jako hodnota asociovaná s názvem proměnné, do které je přiřazováno.
- `FunctionNode`: Pro každou funkci, kterou reprezentuje třída `Function`, byla vytvořena potenční funkce, o kterou byla třída `Function` rozšířena. Při průchodu jsou nejdříve vyhodnoceni všichni potomci vrcholu. Jejich výsledek je společně s informacemi o funkci a funkcí samotnou předán argumentem potenční funkci. Vyhodnocování se provádí s odchytáváním výjimek `TooManyElementsException`, které nastanou, pokud by se provedlo více vyhodnocení na vrcholu než je specifikováno v `RangeEvaluationInfo`. V takovém případě se u výsledného kontejneru nastaví, že může obsahovat všechny možné hodnoty.
- `LiteralNode`: Pro konstantu se vytvoří nový kontejner dle datového typu konstanty, obsahující pouze její hodnotu.
- `OperatorNode`: Metoda se vykoná stejně jako pro třídu `FunctionNode`, ze které ji zdědí.
- `VariableNode`: Vrátí se kontejner pro reprezentovanou proměnnou ze slovníku.

4.2.1 Vyhodnocení funkcí

Definice funkce jazyka Expev byla rozšířena z definice hlavičky a funkce, která určuje její vyhodnocení, o potenční funkci a o dodatečné informace o funkci pomocí třídy `FunctionInfo`. Potenční funkce přijímá jediný argument `FunctionRangeEvaluationArguments`, ve kterém obdrží funkci pro normální vyhodnocení, informace o funkci skrz `FunctionInfo` a možné hodnoty, které mají být vyhodnoceny, pomocí třídy `ContainerArgumentsIterator`.

Vícero funkcí jazyka Expev můžou mít přiřazenou stejnou potenční funkci, v takovém případě potenční funkce pouze vybírá vhodné argumenty, které stačí vyhodnotit pro získání všech možných ohodnocení. Rozdělení funkcí dle shodné potenční funkce odpovídá rozdělení v sekci 3.2 pokud není řečeno jinak.

FunctionInfo

Třída `FunctionInfo` uchovává informace o chování funkce. Má vícero atributů, ne všechny musí být pro danou funkci vyplněny. Některé atributy jsou určeny pro všechny funkce a jiné pro vybrané skupiny funkcí.

Mezi atributy určené pro všechny funkce patří:

- `range`: Obor hodnot pro funkce, jejichž návratový typ je číslo. Atribut je použit k nastavení výsledku, pokud se předčasně ukončí vyhodnocování potenční funkce.
- `resultContainer`: Kontejner, ve kterém bude uchován výsledek potenční funkce.
- `maxElements`: Určuje nejvyšší možný počet vyhodnocení funkce.
- `nullReplacements`: List hodnot říkající, čím se nahradí hodnota `null` v odpovídajícím argumentu, pokud jí bude obsahovat.
- `nullResult`: Atribut značící, kdy může funkce vyprodukovat hodnotu `null` – nikdy, pouze pokud libovolný argument je `null` nebo i v jiných případech. Pokud by bylo rozsahové vyhodnocení předčasně ukončeno, tak dle tohoto atributu se přidá hodnota `null` do výsledku.

Atributy `resultContainer` a `maxElements` jsou nastavovány až při vyhodnocování vrcholu `FunctionNode` dle nastavení průchodu, ostatní atributy mohou být nastaveny při definování funkce.

Funkce monotonní na konečném počtu intervalů

Potenční funkcí pro tuto skupinu funkcí je `monotonousFunction`. Ta pro vyhodnocení vyžaduje, aby funkce měla vyplněný atribut `domainWithMonotony` ve `FunctionInfo` popisující její monotonii, dle které jsou intervaly v argumentu rozděleny.

Periodické funkce

Pro periodické funkce je potřeba ve `FunctionInfo` definovat délku periody `period` a průběh jedné libovolné periody pomocí `domainWithMonotony`. Interval se nejdříve posune o celočíselný násobek period, aby byl hned za začátkem prvního intervalu z `domainWithMonotony` a následně se rozdělí dle monotonie funkce na jedné periodě a vyhodnotí. Z důvodu nepřesnosti datového typu `Double` se intervaly, které jsou velmi vzdáleny od 0 a jsou reprezentovány velmi omezeným počtem čísel, vyhodnotí v každé možné hodnotě. Intervaly s délkou větší než perioda se vyhodnotí na obor hodnot funkce. Pokud je krajní bod intervalu plus nebo mínus nekonečno, přidá se do možných výsledků hodnota `null`.

Oproti rozdělení funkcí v sekci 3.2 do této skupiny patří pouze `sin`, `cos` a jen ty používají potenční funkci `continuousPeriodicFunction`. Funkce `tan` se vyhodnocuje zvlášť kvůli nespojitosti v bodech $\frac{\pi}{2} + k \cdot \pi; k \in \mathbb{Z}$, nicméně využívá stejnou logiku pro výpočet rozdělení intervalů.

Funkce monotonní pro každý parametr

Z funkcí monotonních pro každý parametr jsou vyhodnocované dohromady pouze ty, které jsou v každém parametru rostoucí, funkce `minus` je vyhodnocována zvlášť. Potenční funkce `boundsDefinedFunction` vyhodnotí funkce pro všechny možné dolní a horní hranice intervalů. Pro některé funkce může být výsledek hodnota `null`, pokud se mezi argumenty nachází $+\infty$ i $-\infty$. Proto jsou provedeny vyhodnocení funkce, kde jsou nahrazeny hodnoty $\pm\infty$ největší (respektive nejmenší) možnou konečnou hodnotou. Některé funkce se mohou vyhodnotit na nekonečno, i když se v argumentu nekonečno nevyskytuje. Přidáním dalšího argumentu nekonečna s opačným znaménkem se funkce vyhodnotí na `null`. Pro získání této hodnoty se vhodně vyberou argumenty a na nich se funkce vyhodnotí. Celkový počet vyhodnocení argumentů omezených jedním intervalem je nejvýše 5, což neovlivní asymptotickou složitost vyhodnocení.

Funkce s diskrétními argumenty

Funkce, které mají parametry pouze výčtových typů, je speciální případ funkcí, které navíc povolují i číselné parametry. Tyto dvě skupiny, popsané v sekcích 3.2.2 a 3.2.3, sdílejí potenční funkci `discreteArgumentsFunction`. Od funkcí s číselným typem je vyžadováno, aby měly ve `FunctionInfo` vyplněný atribut `argumentLimitsFunctions` funkcemi, které shora omezí možné hodnoty číselných argumentů na základě předchozích argumentů. Číselné argumenty jsou dále omezeny zdola nulou a na celá čísla.

Před začátkem vyhodnocení všech možností se zkontroluje, jestli některá množina argumentů pro řetězce neobsahuje všechny možné řetězce. V takovém případě se rovnou vrátí kontejner obsahující obor hodnot funkce, pokud

je návratový typ číslo a je pro danou funkci definovaný, nebo se na něm nastaví všechny možné hodnoty. V obou případech se ještě nastaví, zda výsledek může obsahovat hodnotu `null` dle atributu `nullResult` ve `FunctionInfo`. Samotné vyhodnocení zkouší všechny možné argumenty rekurzivně zafixováním prvního argumentu a zavolání konstrukce argumentů na zbytek. Jakmile jsou zafixovány všechny argumenty zavolá se ně funkce pro standardní vyhodnocení.

Vyhodnocení nespojitých číselných funkcí

Počet vyhodnocení nespojitých funkcí často přesáhne maximální počet vyhodnocení. V tom případě je vyhodnocení zastaveno a jako výsledek je nastaven celý obor hodnot funkce, což je značné nepřesné. Výsledek se dá zpřesnit i se zachováním nízkého počtu vyhodnocení nalezením minima a maxima z možných vyhodnocení a přidáním nejmenšího intervalu je obsahující do výsledku. Takovéto vyhodnocení stále nezaručuje, že nebude překročen limit v počtu vyhodnocení, např. pro funkce s více parametry může být množin ohraničených argumenty více než limit.

Rozhodnutí, kdy vyhodnotit každou souvislou množinu zvlášť, a kdy pouze nalézt minimum a maximum, lze provést na základě porovnání hodnoty `maxElements` z `FunctionInfo` a počtu potřebných vyhodnocení. Toto řešení se využívá u funkce `pow`, pokud je báze záporná. Další možností je nejdříve zkusit vyhodnotit každý bod, a pokud se zjistí, že proběhne více vyhodnocení, než je limit, přepne se na hledání minima a maxima. Druhého typu vyhodnocení využívají funkce pro zaokrouhlování.

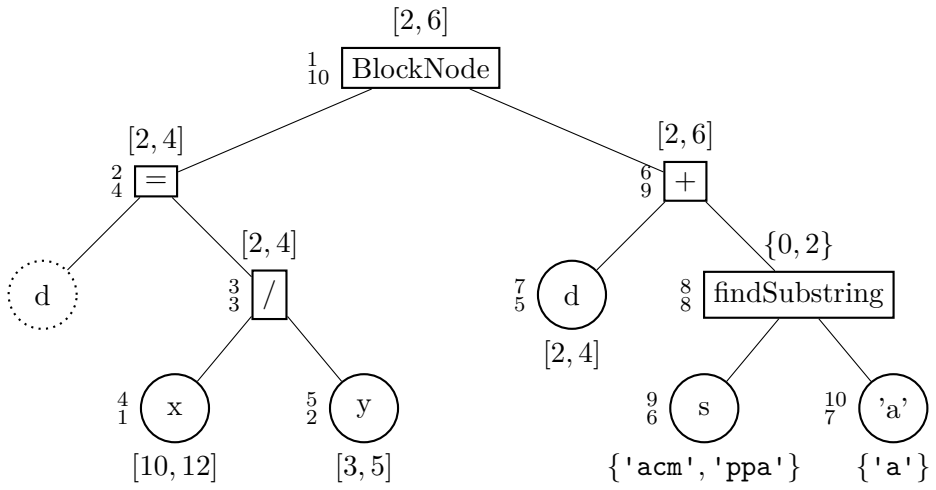
Vyhodnocení dle datového typu

Jelikož `Number` je reprezentován datovým typem `Double`, je třeba věnovat se nedefinovaným hodnotám, zejména při použití hodnot $\pm\infty$ a ± 0.0. Následující výrazy se interně vyhodnotí na `NaN`:

- $\pm\infty \mp \infty$
- $\frac{\pm\infty}{\pm\infty}$
- $\frac{\pm 0.0}{\pm 0.0}$
- $\pm 0.0 \cdot \pm\infty$
- $\texttt{pow}(1, \pm\infty)$

Při vyhodnocení se navíc vyhodnotí funkce v následujícím (respektive předcházejícím) prvku, pokud ho argument může nabývat.

Datový typ má omezenou přesnost, což může vyústit v nečekané chování, např. platí rovnost `0.0 == -0.0`, ale `pow(0.0, -1)` a `pow(-0.0, -1)` se vyhodnotí na různé hodnoty. Dále pro funkci `pow` s záporným základem a sudým



Obrázek 4.1: Průchod AST pro rozsahové vyhodnocení výrazu

Kontejnery s výsledky rozsahového vyhodnocení vrcholu jsou uvedeny nad vrcholem (resp. pod vrcholem pro konstanty a proměnné). Levý horní index značí pořadí začátků vyhodnocování vrcholu, levý dolní index pořadí ukončení.

exponent z intervalu $[10^{15}, 10^{16}]$ může být na některých platformách [2] výsledek $-\infty$.

4.3 Příklad navrženého průchodu

Navržený a implementovaný průchod je demonstrován na výrazu jazyka Expev:

```
Number d = x/y; d + findSubstring(s,'a')
```

Kde proměnné mohou nabývat následujících hodnot:

- $x \in [10, 12]$
- $y \in [3, 5]$
- $s \in \{\texttt{'acm'}, \texttt{'ppa'}\}$

Sestavený AST pro výraz je znázorněn na obrázku 4.1. Obrázek dále zobrazuje pořadí vyhodnocení vrcholů AST – levý horní index znázorňuje pořadí

[2]Tato nepřesnost byla zpozorována na stroji s operačním systémem Microsoft Windows 10 a spouštěné pomocí Oracle JDK verze 12.0.2.

začátků vyhodnocení a levý dolní index pořadí konců. Nad každým operátorem a pod každým listem jsou vypsány kontejnery obsahující výsledek rozsahového vyhodnocení vrcholu. Levý potomek vrcholu = je uveden pouze pro vizualizaci, do které proměnné se přiřazuje.

Průchod začíná ve vrcholu `BlockNode` s vstupními proměnnými x, y, s ve slovníku jako parametr průchodu. Parametr určující maximální počet vyhodnocení je dostatečně velký (alespoň 2). Každý vrchol nejdříve vyhodnotí všechny své potomky, popis průběhu průchodu se touto části nezabývá. Pořadí kroků průchodu odpovídá pořadí ukončení vyhodnocení vrcholů.

1. Vrátí se kontejner pro proměnnou x, obsahující interval $[10, 12]$.
2. Vrátí se kontejner pro proměnnou y, obsahující interval $[3, 5]$.
3. Všechny možné hodnoty dělitele i dělence jsou větší než 0. Výsledek je získán vyhodnocením ve dvou bodech: $\frac{[10,12]}{[3,5]} = [\frac{10}{5}, \frac{12}{3}] = [2, 4]$.
4. Do slovníku definovaných proměnných se přidá proměnná d a s ní kontejner, který byl získaný jako výsledek předchozího kroku a obsahuje interval $[2, 4]$.
5. Vrátí se kontejner pro proměnnou d, obsahující interval $[2, 4]$.
6. Vrátí se kontejner pro proměnnou s, obsahující řetězce `'acm'` a `'ppa'`.
7. Vytvoří se nový `StringContainer` obsahující pouze řetězec `'a'`.
8. Vyhodnotí se všechny možné hodnoty argumentů:
   - `findSubstring('acm','a') = 0`
   - `findSubstring('ppa','a') = 2`

   Výsledkem je `IntervalContainer` představující množinu $\{0, 2\}$.
9. Čísla jsou reprezentována intervalem s nulovou délkou: $0 = [0, 0], 2 = [2, 2]$. Vyhodnotí se všechny případy:
   - $[2, 4] + [0, 0] = [2 + 0, 4 + 0] = [2, 4]$
   - $[2, 4] + [2, 2] = [2 + 2, 4 + 2] = [4, 6]$

   Intervaly mají neprázdný průnik, a proto jsou po přidání do výsledného kontejneru sjednoceny: $[2, 4] \cup [4, 6] = [2, 6]$.
10. Výsledek levého potomka se ignoruje a pouze se vrátí výsledek vyhodnocení pravého potomka, tedy výsledkem je `IntervalContainer` obsahující interval $[2, 6]$.

Výsledkem průchodu pro rozsahové vyhodnocení výrazu je interval $[2, 6]$. Ve výrazu se nevyskytují jednotlivé proměnné vícekrát, ani nebyla vyhodnocením žádného vrcholu zavedena nepřesnost, výsledek je tedy přesný.

4.4 Testování

Testování implementace je prováděno na několika úrovních. Pro datové struktury byly vytvořeny jednotkové testy ověřující korektnost jednotlivých metod datových struktur. Rozsahové vyhodnocení bylo ověřeno manuálně vytvořenými testy i automatizovanými testy pro náhodné výrazy s náhodně omezenými proměnnými.

4.4.1 Automatizované testování

Od rozsahového vyhodnocení výrazu se očekává, že jeho výsledek obsahuje všechna možná vyhodnocení, která by šla získat pomocí standardního vyhodnocení výrazu na omezených argumentech. Tato skutečnost umožňuje využít standardní vyhodnocení k automatizovanému testování implementace rozsahového vyhodnocení.

Třída `SampleTest` obsahuje několik testů, které vygenerují náhodný výraz s náhodně omezenými proměnnými a ověří korektnost implementace rozsahového vyhodnocení za předpokladu správnosti standardního vyhodnocení výrazu. Testy jsou vyhodnocovány tak dlouho, dokud není nalezen spor mezi oběma vyhodnoceními.

Generování testů

Zkontrolovat správné vyhodnocení každé potenční funkce pro všechna možná omezení argumentů není uskutečnitelné kvůli velikosti potenční množiny čísel i řetězců. Testy jsou generovány náhodně s účelem pokrýt co největší množinu možných argumentů a výrazů s důrazem na mezní hodnoty.

Existují dva druhy testů – kontrolující vyhodnocení jedné funkce a testy kontrolující náhodně vygenerovaný výraz složený z vícero funkcí. Testy jediné funkce se provedou rychleji a umožňují snadnější nalezení případné chyby. Testování výrazu složeného z více funkcí je podobnější reálnému využití, ale je také náročnější. Pro některé funkce jsou v jednotlivých testech při určitých argumentech uděleny výjimky. Ověření, zda se nemá výraz vyhodnotit kvůli takové výjimce, by znamenalo několikanásobné vyhodnocení výrazu, a proto jsou z těchto testů dané funkce vyřazeny.

Pro zajištění rychlosti vyhodnocení testu je počet hodnot v argumentu potenční funkce omezen, stejně tak i maximální počet argumentů variadických funkcí, zanoření funkcí v náhodném výrazu a parametr na maximální počet vyhodnocení na jednom vrcholu. To by významně nemělo ovlivnit schopnost testů odhalovat chyby, jelikož jediná komponenta, která přímo pracuje s velikostí kontejnerů, je parametr určující maximální počet vyhodnocení. Ten je pro každé ohodnocení generován náhodně.

Náhodně zvolený argument

Test `singleFunctionTest` kontroluje, zda vyhodnocení funkce s náhodně vybranými argumenty dle omezení se vyskytuje v rozsahovém vyhodnocení všech možných argumentů. Test se neprovádí pro funkci `pow(řetězec, číslo)`, pokud je druhý argument až příliš velký z důvodu, že výsledný řetězec může požadovat na uchování více paměti, než je přidělená paměť programu. Obdobně funguje test `treeEvaluationTest`, který testuje náhodný výraz místo náhodné funkce.

Stejné vyhodnocení

Při omezení každého argumentu funkce jedinou hodnotou existuje jediné standardní ohodnocení těchto argumentů. Test `exactSingleResultTest` kontroluje, zda výsledek rozsahového vyhodnocení obsahuje pouze hodnotu standardního vyhodnocení. Výjimkou je funkce `rand`, která se nechová deterministicky. A také funkce `pow(číslo, číslo)`, jelikož pro poskytnutou implementaci platí `pow(0.0, -1) != pow(-0.0, -1)` a pro záporný základ a sudý exponent z intervalu $[10^{15}, 10^{16}]$ může být na některých platformách výsledek $-\infty$. Ze stejného důvodu jako v předchozím testu není vyhodnocena ani funkce `pow(řetězec, číslo)`. Vyhodnocení náhodných výrazů kontroluje test `treeExactEvaluationTest`.

Ověření prostých funkcí

Pro prosté funkce platí, že libovolné rozdílné argumenty z definičního oboru funkce se nevyhodnotí na stejnou hodnotu. Zda tuto vlastnost splňuje i vyhodnocení prostých funkcí, je testováno metodou `simpleFunctionTest`. Test kontroluje, zda rozsahové vyhodnocení intervalu a standardní vyhodnocení bodu je disjunktní. Nepřesnost datového typu `double` způsobuje, že více hodnot se může vyhodnotit stejně. Například funkce `tanh` má definiční obor $\mathbb{R}$, ale obor hodnot pouze interval $(-1, 1)$. Jelikož `double` může nabývat konečně mnoha hodnot, zobrazí se některé čísla na stejnou hodnotu. Test ztrátu přesnosti detekuje a výraz nevyhodnocuje.

Vykonání automatizovaných testů

Testy byly prováděny na stroji s procesorem Intel Core i7-8550U, 16 GB DDR4 RAM, operačním systémem Microsoft Windows 10 a spouštěné pomocí Oracle JDK verze 12.0.2. Pro finální verzi programu bylo uskutečněno 100 milionů náhodných testů za 1 hodinu a 4 minuty bez nalezení jakékoliv chyby. Z toho však nelze usoudit, že by program nemohl obsahovat žádné chyby, protože provedené testy nepokrývají všechna možná vyhodnocení. Nicméně počet provedených testů bez nalezení chyby alespoň částečně validuje řešení.

4.4.2 Ověření stávající implementace vyhodnocovače

Vytvořené testy odhalily nestandardní chování vyhodnocovače v následujících případech a pomohly k jeho nápravě.

- Funkce `round(číslo)` i `round(číslo, číslo)` vracely špatné výsledky kvůli konverzi na datový typ `long`.
- Nesprávné porovnání dvou nekonečen ve funkcích `eq` a `neq`.
- Funkce `div` se vyhodnotila pro dělitel 0.0 na hodnotu `null`, ale pro dělitel -0.0 na $\pm\infty$.
- Vyhodnocovač mohl vrátit hodnotu `NaN`.
- Nekonečný cyklus při vyhodnocení funkce `pow(řetězec, číslo)`.
- Nesprávný výsledek funkce `pow(řetězec, číslo)`, pokud byl druhý argument záporný.
- Nekonzistentní zaokrouhlování číselných argumentů pro funkce, jejichž argumenty jsou řetězce i čísla.

Závěr

Cílem práce bylo navrhnout průchod nad abstraktním syntaktickým stromem, který co nejpřesněji omezí možné hodnoty reprezentovaného výrazu na základě omezení vstupních proměnných a implementovat jej do vyhodnocovače výrazů v aplikaci Grades. Tento průchod byl navrhnut a implementován včetně konkrétních datových struktur pro datové typy řetězec, číslo a boolean s podmínkou na omezení počtu kroků průchodu v každém vrcholu. Vybrané typy vnitřních vrcholů AST byly rozděleny do skupin dle podobnosti vyhodnocení pro zjednodušení implementace a přidávání dalších typů vrcholů do vyhodnocovače. Korektnost implementovaného průchodu byla otestována pomocí náhodných automatizovaných testů s použitím standardního vyhodnocení jako reference.

Rozšíření vyhodnocovače aplikace Grades umožní přidání funkcionality pro zjištění potencionální klasifikace v jednotlivých předmětech studenta a pro automatizovanou kontrolu definice hodnocení předmětů.

Na práci je možné navázat navrhnutím datových struktur pro omezení hodnot nečíselných datových typů jinak než výčtem, např. pomocí regulárních výrazů pro řetězce nebo pro složené datové typy a taktéž pro ně stanovit rozsahové vyhodnocení funkcí. Dalším možným rozšířením je pro vybranou omezenou množinu výsledků vyhodnocení výrazů omezit vstupní proměnné tak, aby jakékoliv jejich ohodnocení se nacházelo ve vybrané množině.

Bibliografie

1. BOURBAKI, Nicolas. *Elements of mathematics*. Springer, 2004.
2. AHO, Alfred V.; LAM, Monica S.; SETHI, Ravi; ULLMAN, Jeffrey D. *Compilers: principles, techniques, & tools*. 2. vyd. Pearson/Addison Wesley, 2007.
3. JONES, Joel. Abstract syntax tree implementation idioms. In: *Proceedings of the 10th conference on pattern languages of programs (plop2003)*. 2003, s. 26.
4. VOJTĚCH, Jarník. *Diferenciální počet II*. Nakladatelství Československé akademie věd, 1956.
5. HAMHALTER, Jan; JAROSLAV, Tišer. *Diferenciální počet funkcí více proměnných*. Česká technika - nakladatelství ČVUT, 2005.
6. EVOLUTION-TEAM, ICT oddělení FIT ČVUT v Praze. *Expev* [online] [cit. 2020-04-11]. Dostupné z: `https://gitlab.fit.cvut.cz/evolution-team/expev`.
7. ORACLE CORPORATION. *Java™ Platform, Standard Edition 8 API Specification* [online] [cit. 2020-04-11]. Dostupné z: `https://docs.oracle.com/javase/8/docs/api/`.
8. EVOLUTION-TEAM, ICT oddělení FIT ČVUT v Praze. *classification-documentation* [online] [cit. 2020-04-11]. Dostupné z: `https://gitlab.fit.cvut.cz/evolution-team/classification-documentation`.
9. BEYER, Dirk; GULWANI, Sumit; SCHMIDT, David A. Combining model checking and data-flow analysis. In: *Handbook of Model Checking*. Springer, 2018, s. 493–540.
10. NOCEDAL, Jorge; WRIGHT, Stephen J. *Numerical optimization*. 2. vyd. Springer, 2006.

Příloha A

Seznam použitých zkratek

AST Abstraktní syntaktický strom

MDS Množinová datová struktura

NaN Not a Number, hodnota nereprezentující číslo

MOP Meet Over all Paths

MFP Maximal Fixed Point

Příloha B

Funkce a operátory jazyka Expev

Tabulka B.1: Funkce jazyka expev

identifikátor	návratový typ	parametry	popis
ABS	číslo	číslo	absolutní hodnota
SQRT	číslo	číslo	druhá odmocnina
LOG	číslo	číslo	logaritmus o základu e
LOG10	číslo	číslo	logaritmus o základu 10
SIN	číslo	číslo	sinus
COS	číslo	číslo	cosinus
TAN	číslo	číslo	tangens
ASIN	číslo	číslo	arkus sinus
ACOS	číslo	číslo	arkus cosinus
ATAN	číslo	číslo	arkus tangens
SINH	číslo	číslo	hyperbolický sinus
COSH	číslo	číslo	hyperbolický cosinus
TANH	číslo	číslo	hyperbolický tangens
MIN	číslo	číslo, ...	minimum
MAX	číslo	číslo, ...	maximum
SUM	číslo	číslo, ...	součet
AVG	číslo	číslo, ...	aritmetický průměr
PRODUCT	číslo	číslo, ...	produkt
CEIL	číslo	číslo	horní celá část
FLOOR	číslo	číslo	dolní celá část
ROUND	číslo	číslo	zaokrouhlení na celá čísla
ROUND	číslo	číslo, číslo	zaokrouhlení na daný počet desetinných míst
RAND	číslo	-	náhodné číslo z intervalu $[0, 1)$
OR	boolean	boolean, ...	logické or

AND	boolean	boolean, ...	logické and
TOSTRING	řetězec	číslo	převedení čísla na řetězec, ”null”pokud je argument null
TOSTRING	řetězec	boolean	řetězec ”true”, ”false”nebo ”null”
TONUMBER	číslo	řetězec	převedení řetězce na číslo, null pokud argument není číslo
TONUMBER	číslo	boolean	1 pro true, 0 pro false, null pro null
TOBOOLEAN	boolean	číslo	null pro null, false pro 0, jinak true
TOBOOLEAN	boolean	řetězec	true pro ”true”, false pro ”false”, jinak null
LENGTH	číslo	řetězec	počet znaků řetězce v argumentu
CONCAT	řetězec	řetězec, ...	zřetězení
POW	řetězec	řetězec, ...	zřetězení
SUBSTRING	řetězec	řetězec, číslo, číslo	podřetězec (1) v zadaných mezích (2) a (3)
PREFIX	řetězec	řetězec, číslo	podřetězec (1) končící na pozici (2)
SUFFIX	řetězec	řetězec, číslo	podřetězec (1) začínající na pozici (2)
CONTAINS	boolean	řetězec, řetězec	zda (2) je podřetězcem (1)
STARTSWITH	boolean	řetězec, řetězec	zda (1) začíná na (2)
ENDSWITH	boolean	řetězec, řetězec	zda (1) končí na (2)
MATCHES	boolean	řetězec, řetězec	zda (1) odpovídá regulárnímu výrazu (2)
TOLOWERCASE	řetězec	řetězec	převedení do malých písmen
TOUPPERCASE	řetězec	řetězec	převedení do velkých písmen
FINDSUBSTRING	číslo	řetězec, řetězec	pozice prvního výskytu (2) uvnitř (1), null pokud se nevyskytuje

Tabulka B.2: Operátory jazyka Expev, seřazené sestupně dle priority

operátor	návratový typ	syntaxe	funkce	popis
!	boolean	!boolean	NEG	logická negace
-	číslo	-číslo	UMINUS	unární mínus
+	číslo	+číslo	UPLUS	unární plus
^	číslo	číslo ^číslo	POW	mocnění
^	řetězec	řetězec ^číslo	POW	(2) krát zopakování (1)
*, / %	číslo	číslo * číslo	PRODUCT, DIV, MOD	násobení, dělení, modulo
+, -	číslo	číslo + číslo	SUM, MINUS	sčítání, odčítání
+	řetězec	řetězec + řetězec	SUM	řetězení
>, <, >=, <=	boolean	číslo > číslo	GT, GTE, LT, LTE	porovnání
==, !=	boolean	výraz == výraz	EQ, NEQ	rovnost, nerovnost
&&	boolean	boolean && boolean	AND	logické and
||	boolean	boolean || boolean	OR	logické or
?:	výraz	boolean ? výraz : výraz	IF	(2) pokud platí (1), jinak (3)

Příloha C

Obsah přiloženého DVD

- `readme.md` ..... popis obsahu DVD a návod k spuštění
- `expev` ..... zdrojové kódy implementace
- `text`
  - `src` ..... zdrojová forma práce ve formátu XeLaTeX
  - `thesis.pdf` ..... text práce ve formátu PDF