Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Ladybug: Teoretický model objektového programovacího jazyka

2023

Adéla Hlaváčová

Vedoucí práce:
doc. RNDr. Michal Krupka, Ph.D.

Studijní program: Informatika,
Specializace: Obecná informatika

Bibliografické údaje

Autor:	Adéla Hlaváčová
Název práce:	Ladybug: Teoretický model objektového programovacího jazyka
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2023
Studijní program:	Informatika, Specializace: Obecná informatika
Vedoucí práce:	doc. RNDr. Michal Krupka, Ph.D.
Počet stran:	29
Přílohy:	elektronická data v úložišti katedry informatiky
Jazyk práce:	český

Bibliographic info

Author:	Adéla Hlaváčová
Title:	Ladybug: Theoretical model of object-oriented programming language
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2023
Study program:	Computer Science, Specialization: General Computer Science
Supervisor:	doc. RNDr. Michal Krupka, Ph.D.
Page count:	29
Supplements:	electronic data in the storage of department of computer science
Thesis language:	Czech

Anotace

Cílem práce je definovat teoretické základy objektově orientovaného programovacího jazyka založeného na formální konceptuální analýze.

Synopsis

The aim of this thesis is to define theoretical basics of object-oriented programming language based on formal concept analysis together with the theory on which it is based.

Klíčová slova: formální konceptuální analýza; programovací jazyk

Keywords: formal concept analysis; programming language

Tímto bych chtěla poděkovat doc. RNDr. Michalu Krupkovi, Ph.D., který svým nadšením a cennými radami přispěl ke zdárnému konci této bakalářská práce.

Odevzdáním tohoto textu jeho autor/ka místopřísežně prohlašuje, že celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

Obsah

Seznam tabulek

Seznam zdrojových kódů

1 Úvod

V čestině existuje rčení „Kolik jazyků znáš, tolikrát si člověkem.“ Takový jazyk se pokusíme definovat v této práci. Konktrétně popíšeme teoretický model objektově orientovaného programovacího jazyka založeného na formální konceptuální analýze.

Nejprve se seznámíme s matematickou metodou zvanou Formální konceptuální analýza. Definujeme si rozšíření klasického pojmu formálního kontextu, který nazveme objektový kontext, a na němž teoretický model postavíme. V další fázi vytvoříme teoretický základ jazyka včetně definice jeho výpočetního modelu. Jedná se tedy o teoretickou práci s tím, že na závěr uvedeme příklad průběhu základních výpočtů, konkrétně počítání s přirozenými čísly.

2 Formální konceptuální analýza

Nejprve stručně vysvětlíme základní poznatky Formální konceptuální analýzy. Zdrojem informací pro tuto část byl učební text [1].

Formální konceptuální analýza určitým způsobem zachycuje vztah mezi *objekty* a jejich *atributy (vlastnostmi)*. V základní podobě tento vztah popisuje, zda objekt daný atribut má, či nemá. Proto si ho lze velice snadno představit jako tabulku, v jejíchž řádcích jsou uloženy jednotlivé objekty a sloupcích atributy, a vztah „objekt x má atribut y“ je zachycen křížkem na průsečíku řádku x a sloupce y.

Mějme neprázdnou množinu objektů X, neprázdnou množinu atributů Y a binární relaci I mezi množinami X a Y. Pak trojici (X, Y, I) nazveme *(formálním) kontextem*. Pokud dvojice (x, y) je v relaci I, říkáme, že objekt x má atribut y.

Každý formální kontext (X, Y, I) vytváří zobrazení $^{\uparrow_I} : 2^X \to 2^Y$ a $^{\downarrow_I} : 2^Y \to 2^X$, kde pro každou množinu $A \subseteq X$ je $A^{\uparrow_I}$ množina všech atributů společných všem objektům z A a pro každou množinu $B \subseteq Y$ je $B^{\downarrow_I}$ množina všech objektů majících všechny atributy z B. Pokud bude z kontextu jasné, ke kterému formálnímu kontextu zobrazení přísluší, budeme místo $^{\uparrow_I}$ a $^{\downarrow_I}$ psát jen $^{\uparrow}$ a $^{\downarrow}$.

Formální koncept kontextu (X, Y, I) je dvojice (A, B), kde $A \subseteq X$ a $B \subseteq Y$, taková, že $A^{\uparrow} = B$ a $B^{\downarrow} = A$. Množina A se pak nazývá *extent* a množina B *intent* formálního konceptu (A, B). Dále platí, že dvojice (A, B) je konceptem, pokud je *pevným bodem* zobrazení $^{\uparrow}$ a $^{\downarrow}$.

Zobrazení $f : 2^X \to 2^Y$ a $g : 2^Y \to 2^X$ tvoří *Galoisovu konexi* mezi množinami X a Y, pokud pro $A, A_1, A_2 \subseteq X$ a $B, B_1, B_2 \subseteq Y$ platí:

- $A_1 \subseteq A_2 \Rightarrow f(A_2) \subseteq f(A_1)$,
- $B_1 \subseteq B_2 \Rightarrow g(B_2) \subseteq g(B_1)$,
- $A \subseteq g(f(A))$,
- $B \subseteq f(g(B))$.

Mějme Galoisovu konexi (f, g) mezi množinami X a Y a libovolné množiny $A \subseteq X$ a $B \subseteq Y$. Pak platí, že $f(A) = f(g(f(A)))$ a $g(B) = g(f(g(B)))$. Pokud navíc platí, že $f(A) = B$ a $g(B) = A$, pak dvojici (A, B) nazveme *pevným bodem* Galoisovy konexe (f, g).

Pro formální kontext (X, Y, I) tvoří zobrazení $^{\uparrow}$ a $^{\downarrow}$ Galoisovu konexi mezi množinami X a Y.

Uzávěrovým operátorem na množině X chápeme zobrazení $C : 2^X \to 2^X$ splňující pro libovolné množiny $A, A_1, A_2 \subseteq X$, že

1. $A \subseteq C(A)$,
2. $A_1 \subseteq A_2 \Rightarrow C(A_1) \subseteq C(A_2)$,
3. $C(A) = C(C(A))$.

Pro uzávěrový operátor C na množině X nazveme množinu $A \subseteq X$ splňující, že $C(A) = A$, *pevným bodem* uzávěrového operátoru C.

Pro Galoisovu konexi (f, g) mezi množinami X a Y tvoří složené zobrazení $C_X = f \circ g$, kde pro libovolnou množinu $A \subseteq X$ je $f \circ g(A) = g(f(A))$, uzávěrový operátor na množině X a složené zobrazení $C_Y = g \circ f$, kde pro libovolnou množinu $B \subseteq Y$ je $g \circ f(B) = f(g(B))$, uzávěrový operátor na množině Y.

Uzávěrový systém na množině Y je libovolný systém $\mathcal{S} \subseteq 2^Y$, jenž obsahuje množinu Y a pro libovolný podsystém $\mathcal{R} \subseteq \mathcal{S}$ platí $\bigcap \mathcal{R} \in \mathcal{S}$. K danému uzávěrovému systému na množině Y lze sestrojit uzávěrový operátor $C_\mathcal{S}$ na množině Y předpisem $C_\mathcal{S}(B) = \bigcap\{S \in \mathcal{S} \mid B \subseteq S\}$, pro libovolnou množinu $B \subseteq Y$. Také pro zadaný uzávěrový operátor C lze sestrojit uzávěrový systém $\mathcal{S}_C = \{B \in 2^Y \mid B = C(B)\}$ na množině Y. Pak platí vztahy $C = C_{\mathcal{S}_C}$ a $\mathcal{S} = \mathcal{S}_{C_\mathcal{S}}$.

Pro podmnožiny B_1 a B_2 množiny Y je výraz $B_1 \Rightarrow B_2$ *atributovou implikací*. Když libovolná množina $B \subseteq Y$ splňuje, že pokud $B_1 \subseteq B$, pak i $B_2 \subseteq B$, říkáme, že $B_1 \Rightarrow B_2$ *platí* v B nebo B je *modelem* $B_1 \Rightarrow B_2$. Množinu $M \subseteq Y$ nazveme *modelem* systému T atributových implikací, jestliže pro každou atributovou implikaci $B_1 \Rightarrow B_2$ z T je M jejím modelem. Mějme systém $\mathcal{M} \subseteq 2^Y$, pak atributová implikace $B_1 \Rightarrow B_2$ je splněna v systému $\mathcal{M}$, když je splněna v každé množině M systému $\mathcal{M}$. Atributová implikace $B_1 \Rightarrow B_2$ platí v kontextu (X, Y, I), jestliže platí v systému $M = \{x^\uparrow \mid x \in X\}$ (systém množin všech atributů všech objektů).

Pro systém T atributových implikací tvoří systém $\mathrm{Mod}(T)$ všech jejích modelů uzávěrový systém na množině Y. A $C_{\mathrm{Mod}(T)}(B) = \bigcap\{M \in \mathrm{Mod}(T) \mid B \subseteq M\}$ je příslušným uzávěrovým operátorem na množině Y. Proto pro libovolnou množinu $B \subseteq Y$ existuje *minimální model* $M \in \mathrm{Mod}(T)$ *obsahující množinu* B, konkrétně $M = C_{\mathrm{Mod}(T)}(B)$.

3 Objektový kontext a jeho rozvití

V této části si zavedeme dvě různá rozšíření klasického pojmu kontextu. V prvním *objektovém*, též *základním*, mají objekty v atributech uloženy opět objekty. Druhou variantou bude *řetězcový kontext*, který spíše odpovídá základnímu pojetí formální konceptuální analýzy, ale má specifickou množinu atributů a to množinu rozvitých atributů Y^*. Na množině rozvitých atributů Y^* si poté definujeme operace *zřetězení* a *nalezení hodnoty atribut*. A nakonec definujeme *rozvitý kontext objektového kontextu*, kterým oba předchozí kontexty propojíme. Ve všech těchto kontextech budeme používat množiny *objektů* X a *(základních) atributů* Y.

Jak už jsme si řekli, v této části zavedeme množinu rozvitých atributů Y^*. Tu můžeme také chápat jako jazyk nad abecedou Y. Ve formálních jazycích se běžně vynechává tečka symbolizující operaci zřetězení a řetězce se zapisují jako posloupnosti prvků abecedy. Jelikož prvky množiny Y nebudou pouze znaky nýbrž celá slova, jak je pro pojmenování atributů běžné, mohlo by dojít k záměně prvků abecedy Y a řetězců jazyka Y^*. Proto budeme jejich zřetězení zapisovat explicitně tečkou mezi zřetězovanými prvky.

Objektový kontext

Nejprve zopakujme, že množinu *objektů* značíme X a množinu *(základních) atributů* Y. *Objektový kontext* je postavený na ternární relaci $I \subseteq X \times Y \times X$ splňující, že pro všechna x z X a pro všechna y z Y existuje nejvýše jedno x' z X takové, že uspořádaná trojice (x, y, x') je v relaci I. Tento kontext označíme trojicí (X, Y, I). Pokud (x, y, x') je prvkem I, pak objekt x' značíme $I(x, y)$. Dostáváme tak parciální funkci, která dvojici (x, y) přiřazuje prvek x'. Pokud objekt x' existuje, říkáme, že objekt x má základní atribut y a jeho hodnota je objekt x'. Dále předpokládejme, že v množině X neexistují dva různé objekty $x_1,\ x_2$, které by byly identické v tom smyslu, že množiny všech jejich atributů by byly stejné a pro jejich libovolný atribut y by byly objekty $I(x_1, y)$ a $I(x_2, y)$ identické.

Hodnotu atributu y objektu x také značíme $x.y$. Atribut y určuje parciální funkci $X \to X$, přiřazující objektům x z X objekty $x.y$. Tuto funkci také nazýváme operace *nalezení hodnoty atributu* a také ji značíme y . U vícenásobné aplikace operace nalezení hodnoty atributu můžeme vynechávat závorky. Tedy místo $(x.y_1).y_2$ můžeme psát $x.y_1.y_2$. Výraz $y_1.y_2$ tedy opět představuje parciální funkci $X \to X$ a říkáme ji operace *vícenásobného nalezení hodnoty atributu daného řetězcem* $y_1.y_2$. . Obecně: Operace α nalezení hodnoty atributu daného řetězcem $\alpha. = y_1.\cdots.y_n.$ je operace taková, že pro objekt x z X je $x.\alpha = x.y_1.\cdots.y_n$. Speciálně pro řetězec $\varepsilon.$ definujeme příslušnou operaci nalezení hodnoty atributu jako identitu, tedy $x.\varepsilon = x$.

Relaci I také můžeme reprezentovat pomocí tabulky, jejíž řádky jsou objekty a sloupce atributy. Pokud objekt x má atribut y, pak má v tomto atributu uložen objekt $x.y$. Tato skutečnost je v tabulce zachycena tak, že na průsečíku řádku x a sloupce y se nachází objekt $x.y$. Například vztah „objekt *kruh* má v atributu *střed* uložen objekt S“ můžeme zachytit Tabulkou 1.

	střed	...
kruh	*S*	
S		
⋮		

Tabulka 1: Příklad objektového kontextu

Řetězcový kontext

Označme Y^* množinu všech řetězců nad množinou Y. Její prvky nazveme *rozvité atributy* a zapisujeme je jako posloupnosti základních atributů z množiny Y oddělených tečkami a s tečkou na konci. Pro atribut y z Y značí $y.$ jednoprvkový řetězec z Y^*. Obecné symboly značící prvky Y^* také ukončujeme tečkou, například $\alpha.$ nebo $\varepsilon.$. Řekneme, že podmnožina množiny Y^* je *uzavřená na prefixy*, pokud s každým prvkem obsahuje i každý jeho prefix. *Zřetězením* prvku $\alpha.$ množiny Y^* s množinou $A \subseteq Y^*$ rozumíme množinu $\alpha.A$ všech řetězců $\alpha.\beta.$, kde $\beta.$ je prvek množiny A.

U operace nalezení hodnoty atributu rozšíříme platnost i na množinu rozvitých atributů. Aplikací $A.\alpha$ operace α *nalezení hodnoty atributu daného řetězcem* $\alpha.$ *z* Y^* *na množinu rozvitých atributů* A rozumíme největší možnou množinu $B \subseteq Y^*$ takovou, že vysledek zřetězení $\alpha.B$ je podmnožinou množiny A. Operaci α nalezení hodnoty atributu množiny rozvitých atributů A také říkáme *odříznutí rozvitého atributu* $\alpha.$ *od množiny* A. Pokud je α dáno jednoprvkovým řetězcem $y.$ (tedy $\alpha = y$), pak aplikaci $A.\alpha$ můžeme značit jako $A.y$.

Trojicí (X, Y^*, J) označme *řetezcový kontext*, ve kterém J je relace mezi množinou objektů X a množinou rozvitých atributů Y^*. Pokud dvojice $(x, \alpha.)$ je v relaci J, říkáme, že objekt x má rozvitý atribut $\alpha.$.

Řekneme, že množina X je *úplná množina objektů* v řetězcovém kontextu (X, Y^*, J), když pro každý objekt x z X a základní atribut y z Y takový, že $y. \in x^{\uparrow}$, existuje objekt x' z X tak, že $x'^{\uparrow} = x^{\uparrow}.y$.

Stejně jako v předchozím kontextu i v tomto můžeme vztah objektů a rozvitých atributů reprezentovat pomocí tabulky. Její řádky odpovídají objektům, sloupce rozvitým atributům a vztah „objekt x má rozvitý atribut $\alpha.$“ je chápán jako křížek v řádku x a sloupci $\alpha.$. Potom příklad „objekt *kruh* má v atributu *střed* objekt S“ z předchozí části můžeme reprezentovat Tabulkou 2.

	střed.	*střed.x.*	*střed.y.*	*x.*	*y.*	...
kruh	×	×	×			
S				×	×	
⋮						

Tabulka 2: Příklad řetězcového kontextu

Rozvitý kontext objektového kontextu

Mějme objektový kontext (X, Y, I). Pak řetězcový kontext (X, Y^*, J) nazveme *rozvitým kontextem* (X, Y^*, I^*) *objektového kontextu* (X, Y, I), pokud pro každý objekt x z X platí, že objekt x má v kontextu (X, Y^*, J) rozvitý atribut $y_1.\cdots.y_n.$ (pro $n \geq 0$), když v kontext (X, Y, I) objekt $x.y_1.\cdots.y_n$ existuje.

V rozvitém kontextu (X, Y^*, J) objektového kontextu (X, Y, I) budeme relaci J značit I^* a příslušné operátory Galoisovy konexe $^{\blacktriangle}$ a $^{\blacktriangledown}$. Pro objekt x z X nazveme množinu $x^{\blacktriangle}$ *rozvitím objektu* x.

Věta 1

Řetězcový kontext (X, Y^*, J) je rozvitým kontextem nějakého objektového kontextu, když splňuje podmínky:

1. *Pro každé x z X je množina $x^{\uparrow_J}$ uzavřená na prefixy.*
2. *Množina objektů X je úplná v kontextu (X, Y^*, J).*

Důkaz

Větu dokážeme pomocí jednotlivých implikací.

1. $\Longrightarrow$

   Mějme rozvitý kontext (X, Y^*, I^*) objektového kontextu (X, Y, I). Čili pro každý objekt x z X platí, že objekt x má v kontextu (X, Y^*, I^*) rozvitý atribut $y_1.\cdots.y_n.$ (pro $n \geq 0$), právě když v kontext (X, Y, I) objekt $x.y_1.\cdots.y_n$ existuje.

   Pro libovolný objekt x dokážeme první podmínku indukcí vzhledem k délce jeho libovolného rozvitého atributu $y_1.\cdots.y_n.$. Pro $n = 0$ platí triviálně, jelikož atribut $\varepsilon.$ nemá žádný prefix. Předpokládejme, že podmínka platí pro atributy délky $n-1$. Objekt x má rozvitý atribut $y_1.\cdots.y_n.$, když objekt $x.y_1.\cdots.y_n$ existuje. To znamená, že musí existovat objekt $x.y_1.\cdots.y_{n-1}$ mající základní atribut y_n . Dále pak objekt $x.y_1.\cdots.y_{n-1}$ existuje, když objekt x má rozvitý atribut $y_1.\cdots.y_{n-1}.$. Vzhledem k tomu, že tento atribut je délky $n-1$, má, dle předpokladu, objekt x i všechny jeho prefixy. Tedy množina všech rozvitých atributů objektu x je uzavřená na prefixy.

   Pokud pro libovolný objekt x a jeho libovolný základní atribut y položíme objekt x' z definice úplnosti množiny objektů roven objektu $x.y$, je druhá podmínka splněna triviálně.

2. $\Longleftarrow$

   K řetězovému kontextu (X, Y^*, J) splňujícímu obě podmínky, definujeme relaci I takovou, že (x, y, x') je v relaci I, když $(x, y.)$ je v relaci J a x' je libovolný objekt z druhé podmínky.

V dalším kroku dokážeme, že řetězcový kontext (X, Y^*, J) je skutečně rozvitým kontextem objektového kontextu (X, Y, I). Dokážeme to indukcí vzhledem k délce rozvitého atributu $y_1.\cdots.y_n.$. Pro $n = 0$ chceme dokázat, že $(x, \varepsilon.) \in J$ když x existuje. To vyplývá z kontrukce objektového kontextu (X, Y, I). Předpokládejme, že konstrukce je správná pro rozvité atributy délky $n - 1$. Tedy chceme dokázat, že $(x, y_1.\cdots.y_n.) \in J$ když objekt $x.y_1.\cdots.y_n$ existuje. To dokážeme pomocí jednotlivých implikací.

(a) $\Rightarrow$

Předpokládejme, že $(x, y_1.\cdots.y_n.) \in J$. Z uzavřenosti na prefixy dostaneme $(x, y_1.\cdots.y_{n-1}.) \in J$. Z předpokladu poté vyplývá, že objekt $z = x.y_1.\cdots.y_{n-1}$ existuje, a z úplnosti množiny X, že má rozvitý atribut $y_n.$. Dle předpokladu také existuje objekt $z.y_n$. To ale znamená, že objekt $(x.y_1.\cdots.y_{n-1}).y_n$ existuje.

(b) $\Leftarrow$

Mějme objekt $x.y_1.\cdots.y_n$, označme si ho z'. To znamená, že existuje objekt $z = x.y_1.\cdots.y_{n-1}$ se základním atributem y_n, a že $(z, y_n, z') \in I$. Z předpokladu vyplývá, že objekt x má rozvitý atribut $y_1.\cdots.y_{n-1}.$, a z konstrukce, že $z = x.y_1.\cdots.y_{n-1}$ má rozvitý atribut $y_n.$. Pak z úplnosti množiny X má objekt x také rozvitý atribut $y_1.\cdots.y_{n-1}.y_n.$.

□

Dvojice (A, B), kde $A \subseteq X$ a $B \subseteq Y^*$, je *konceptem objektového kontextu* (X, Y, I), když B je množinou všech rozvitých atributů, které mají všechny objekty z A, a A je množina objektů, které mají všechny rozvité atrinuty z B. Z Věty 1 plyne, že dvojice (A, B) je konceptem objektového kontextu (X, Y, I), právě když je konceptem v rozvitém kontextu (X, Y^*, I^*) ve smyslu klasické formální konceptuální analýzy.

4 Ladybug: teoretický základ

Teorii, která se ukrývá za teoretickým modelem programovacího jazyka Ladybug, si popíšeme v této části. Konkrétně si řekneme, co je programem a pravidly, jakým způsobem zadáváme množiny rozvitých atributů, definujeme minimální model obsahující zárodek a ukážeme si tvar uživatelských dotazů a jejich vyhodnocování.

4.1 Program

Program v tomto modelu chápeme jako sadu, či množinu, uživatelsky zadaných *pravidel*.

Pravidla

Pravidlo je výraz tvaru *levá strana* $\rightarrow$ *pravá strana*, kde *levá* a *pravá strana* jsou množiny rozvitých atributů.

Zadání množiny rozvitých atributů

Množiny rozvitých atributů můžeme zadat:

1. Jedním prvkem: $y_1.\cdots.y_n.$.
2. Výčtem množin rozvitých atributů: $A_1, \ldots, A_n$, značící sjednocení $A_1 \cup \cdots \cup A_n$.
3. Zřetězením $\alpha.A$ rozvitého atributu $\alpha.$ z Y^* s množinou rozvitých atributů $A \subseteq Y^*$.
4. Jako aplikaci operace α nalezení hodnoty atributu na množinu rozvitých atributů $A \subseteq Y^*$, tedy $A.\alpha$.
5. Operací α nalezení hodnoty atributu, je-li dán tzv. *aktuální objekt*. Je-li x aktuální objekt, pak α značí množinu $x.\alpha^{\blacktriangle}$.

V případech, kdy potřebujeme množiny strukturovat, používáme složené závorky. Pro množinu rozvitých atributů $A \subseteq Y^*$ značíme uzávěr $A^{\blacktriangledown\blacktriangle}$ této množiny $[A]$. V pravidlech se tento zápis nevyskytuje.

Řekneme, že množina rozvitých atributů A *splňuje* pravidlo $B \rightarrow C$, když platí, že pokud B je podmnožinou A, pak i C je podmnožinou A s tím, že jako aktuální objekt bereme objekt s rozvitím rovným množině A.

4.2 Model

Množinu M rozvitých atributů z Y^* nazveme *modelem* systému pravidel P, pokud platí:

1. Množina M je uzavřená na prefixy.
2. Pro libovolnou operaci α nalezení hodnoty atributu danou řetězcem $\alpha. \in M$ množina $M.\alpha$ splňuje všechna pravidla systému P.

Podmínky kladené na model systému pravidel P můžeme splnit, přidáním pravidel:

1. Pro každý rozvitý atribut $\alpha.y. \in Y^*$ přidáme pravidlo $\alpha.y. \rightarrow \alpha.$.
2. Pro každý atribut $\alpha. \in Y^*$ a pro všechna pravidla $B \rightarrow C$ ze systému pravidel P přidáme pravidlo $\alpha.B \rightarrow \alpha.C$.

Každé pravidlo $B \rightarrow C$ obsahující operaci nalezení hodnoty atributu můžeme nahradit množinou pravidel takových, že pro každou množinu $A \subseteq Y^*$ přidáme pravidlo $B' \cup A \rightarrow C'$, kde množiny B' a C' vznikly nahrazením každého výskytu operace α výrazem $A.\alpha$. Vzniklý systém pravidel je pak systém atributových implikací v klasické smyslu známém z formální konceptuální analýzy. Nazýváme ho *systém atributových implikací příslušný systému pravidel* P. Přitom platí, že množina rozvitých atributů je modelem systému pravidel P, právě když je modelem příslušnému systému atributových implikací. Z toho vyplývá, že každý systém pravidel P má minimální model vzhledem k množinové inkluzi. Minimální model obsahující množinu rozvitých atributů $S \subseteq Y^*$ nazveme *minimálním modelem systému pravidel* P *daným zárodkem* S.

Pro model M systému pravidel P definujme množinu $X = \{M.\alpha \mid \alpha. \in M\}$. A definujme relaci $J \subseteq X \times Y^*$, jejíž prvky jsou všechny dvojice $(A, \alpha.)$, kde $\alpha. \in A$. Pak trojice (X, Y^*, J) je řetězcový kontext. Dále definujeme relaci $I \subseteq X \times Y \times X$ takovou, že trojice $(A, y, A') \in I$ právě když $y. \in A$ a $A' = A.y$. Tím získáme objektový kontext (X, Y, I). Z konstrukce kontextů a z Věty 1 okamžitě plyne, že řetězcový kontext (X, Y^*, J) je rozvitým kontextem (X, Y^*, I^*) objektového kontextu (X, Y, I). Objektový kontext (X, Y, I) nazveme *objektovým kontextem daným modelem* M. Pro minimální model M systému pravidel P daným zárodkem S nazveme příslušný objektový kontext (X, Y, I) *minimálním kontextem daný zárodkem* S.

Objekt $x \in X$ *splňuje* pravidlo $B \rightarrow C \in P$, když splňuje každou atributovou implikaci vzniklou dle pravidel uvedených výše příslušících pravidlu $B \rightarrow C$. Objekt $x \in X$ *splňuje* atributovou implikaci $B \rightarrow C \in P$, když platí, že pokud pro všechny rozvité atributy $\beta. \in B$ existuje objekt $x.\beta$, pak také pro každý atributu $\gamma. \in C$ objekt $x.\gamma$ existuje. Objektový kontext (X, Y, I) splňuje pravidlo $B \rightarrow C$, když každý objekt $x \in X$ splňuje toto pravidlo.

4.3 Uživatelské dotazy a jejich vyhodnocování

Elementární dotaz je tvaru atributové implikace *předpoklad* $\rightarrow$ *závěr*, kde předpokladem je množina rozvitých atributů a závěrem rozvitý atribut. Významem je zjištění, zda jestliže má objekt všechny rozvité atributy předpokladu, pak má rozvitý atribut závěru.

Odpověď na elementární dotaz se hledá v následujících krocích.

1. Uživatel zadá dotaz.
2. Vytvoří se minimální kontext daný zárodkem rovným předpokladu dotazu.
3. Ověří se, jestli v tomto minimálním kontextu je splněno pravidlo dané dotazem.

(Obecné) dotazy zapisujeme stejně jako pravidla s tím rozdílem, že předpoklad dotazu smí obsahovat pouze zápisy operací nalezení hodnoty atributu daných globálními atributy, tedy atributy jež mají všechny objekty kontextu. V dotazech můžeme také použít uzávěr množiny rozvitých atributů.

Na obecný dotaz se hledá odpověď následujícím způsobem.

1. Uživatel zadá dotaz.
2. Vytvoří se objektový kontext, neobsahující žádný objekt.
3. Každý zápis operace α nalezení hodnoty atributu symbolizující globální atribut se nahradí výrazem $\{\}.\alpha$.
4. Postupně zevnitř ven se pro každý výraz $A.\alpha$ v předpokladu dotazu vytvoří minimální kontext daný zárodkem rovným množině A. V tomto kontextu se výraz $A.\alpha$ vyhodnotí a v předpokladu dotazu nahradí rozvitím výsledku. Vzniklý kontext se poté sjednotí s již známým kontextem. U sjednocení se za identické považují ty objekty, které mají stejné atributy a mají v nich uloženy identické objekty. Předpoklad dotazu se také upravuje zřetězováním atributů.
5. Vytvoří se minimální kontext daný zárodkem rovným upravenému předpokladu a sjednotí se s už známým kontextem.
6. Ověří se, zda ve vzniklém kontextu je splněno pravidlo dané dotazem.

5 Příklad

Na řadu přichází ukázka definice přirozených čísel v teoretickém modelu Ladybug. Také si ukážeme způsob, jakým definujeme operace a relace na číslech. A na závěr uvedeme vyhodnocení jednoduchého uživatelského výrazu.

Systém pravidel spolu se zadaným zárodkem vytvářejí minimální kontext. Obecně vyloučíme případ, kdy by tento kontext neměl žádný objekt. Zárodek může způsobit vznik objektů, majících všechny atributy nějakého objektu, který se v kontext již nachází.

Také se může stát, že dva objekty budou ekvivalentní v tom smyslu, že mají shodné některé atributy významné pro daný problém. Například budou reprezentovat stejná čísla. To nás vede k pojetí považovat za důležité místo jednotlivých objektů celé koncepty. A také výsledky výpočtu zadaného dotazem i samotný zárodek vztahovat ke konceptům.

Přirozená čísla

Teď si zavedeme pravidla definující přirozená čísla a atributy, kterými budou čísla reprezentována.

Jak bylo napsáno na začátku i u přirozených čísel může dojít k tomu, že dva různé objekty mají atributy reprezentující stejné číslo. Budeme je tedy chápat, jako reprezentanty téhož čísla.

Nejprve definujeme globální atribut `zero`. V něm bude uložen prototyp čísla nula v tom smyslu, že bude obsahovat všechny atributy, které nula má. Ostatní objekty, mající stejné atributy jako prototyp nuly, chápeme také jako reprezentanty nuly. Následující pravidla a pak i některá další se týkají prototypu nuly a tedy i všech objektů, které nulu reprezentují.

```
1    →  zero.is-zero.
2 is-zero.  →  succ.
3 succ.  →  succ.succ.
```

Zdrojový kód 1: Globální atribut nuly

Prázdné místo na levé straně prvního pravidla symbolizuje prázdnou množinu atributů. Rozvitý atribut `zero.is-zero.` na jeho pravé straně značí jednoprvkovou množinu s prvkem `zero.is-zero.`. Podobně atributy `is-zero.`, `succ.` a `succ.succ.` znamenají jednoprvkové množiny postupně s prvkem `is-zero.`, `succ.` a `succ.succ.`. Z prvního pravidla a uzavřenosti na prefixy plyne, že každý objekt má také atribut `zero.`, jinak by toto pravidlo nesplnil. Proto atribut `zero.` označujeme jako globální atribut. Druhým a třetím pravidlem definujeme atribut `succ`, v němž je uložen následník čísla. Třetí pravidlo objekt $x \in X$ splňuje, když pokud má atribut `succ`, pak objekt v tomto atributu uložený má také atribut `succ`.

Dále definujeme předchůdce v závislosti na následníkovi.

```
succ.  →  succ.pred.{ε}
```

Zdrojový kód 2: Předchůdce

V pravidle se nachází triviální operace ε nalezení hodnoty atributu, která každému objektu $x \in X$ přiřazuje hodnototu x. Pro aktuální objekt $x \in X$ zápis `succ.pred.{ε}` znamená množinu vzniklou zřetězením rozvitého atributu `succ.pred.` s množinou všech rozvitých atributů objektu x (jeho rozvitím). Objekt $x \in X$ splňuje pravidlo, když pokud má atribut `succ`, pak hodnotou atributu `succ` je objekt, který má v atributu `pred` objekt x.

Přirozená čísla chápeme jako koncept daný atributem `natural.`, proto zavádíme následující dvě pravidla:

```
is-zero.  →  natural.
natural.  →  succ.natural.
```

Zdrojový kód 3: Přirozená čísla

Prvním pravidlem nule přidáme atribut `natural.`, tím ji také definujeme jako přirozené číslo. Druhé pravidlo objekt $x \in X$ splní, když pokud má atribut `natural`, pak má v atributu `succ` objekt s atributem `natural`. Jinými slovy pokud je objekt přirozené číslo, pak i jeho následník je přirození číslo.

Lemma 2

Mějme neprázdný minimální kontext. Položme $B_0 =$ `[is-zero.]`, $B_1 =$ `[pred.`B_0`, succ.]`, ... , $B_n =$ `[pred.`B_n`, succ.]`, ... *Pak:*

1. $B_{n+1} = B_n$`.succ`
2. *Jestliže* m *je různé od* n, *tak* B_m *je různé od* B_n.

Důkaz

První tvrzení dokážeme sestrojením jednotlivých množin. Nejprve zkonstruujeme množinu B_{n+1}. Zde pokud v atributu použijeme zápis `pred.`$\cdots$`.pred.`, myslíme tím zápis atributu `pred.` $(n+1)$-krát za sebou.

$$
\begin{aligned}
B_{n+1} &= \texttt{[pred.}B_n\texttt{, succ.]} = \cdots = \texttt{[pred.[}\cdots\texttt{[is-zero.]}\cdots\texttt{], succ.]} \\
&= \{\varepsilon\texttt{., pred., pred.pred.,} \ldots\texttt{, pred.}\cdots\texttt{.pred.is-zero.}, && (1)\\
&\quad \texttt{succ.}, && (2)\\
&\quad \texttt{succ.pred., succ.pred.pred.,} \ldots, \\
&\quad\ \texttt{succ.pred.pred.}\cdots\texttt{.pred.is-zero.,} \ldots, && (3)\\
&\quad \texttt{pred.succ., pred.succ.pred.,} \ldots, && (4)\\
&\quad \texttt{pred.pred.succ., pred.pred.succ.pred.,} \ldots, && (5)\\
&\qquad\quad \vdots \\
&\quad \texttt{succ.succ., succ.succ.pred.,} \ldots, && (6)\\
&\quad \texttt{succ.succ.succ., succ.succ.succ.pred.,} \ldots, && (7)\\
&\qquad\quad \vdots \\
&\quad \}
\end{aligned}
$$

Řádek (1) vyplyne z postupného zřetězování a uzavřenosti na prefixy. Využitím pravidla definujícího předchůdce přirozeného čísla získáme atributy na řádku (3). Zřetězením a s pomocí pravidla předchůdce dostaneme atributy na řádcích (4) a (5). První atributy na řádcích (6) a (7) získáme využitím třetího pravidla z definice nuly, zbylé atributy těchto řádků odvodíme využitím druhé podmínky kladené na model a pravidla definujícího předchůdce. Další atributy bychom odvozovali analogicky.

Stejným způsobem zkonstruujeme množinu B_n. Jen zápis `pred.`$\cdots$`.pred.` bude obsahovat n-krát atribut `pred.` .

$$
\begin{aligned}
B_n = {} & \texttt{[pred.}B_{n-1}\texttt{, succ.]} = \cdots = \texttt{[pred.[}\cdots\texttt{[is-zero.]}\cdots\texttt{], succ.]} \\
= {} & \{\varepsilon\texttt{.,\ pred.,\ pred.pred.,\ ...,\ pred.}\cdots\texttt{.pred.is-zero.,} \\
& \quad \texttt{succ.,} \\
& \quad \texttt{succ.pred.,\ succ.pred.pred.,\ ...,} \\
& \quad\ \ \texttt{succ.pred.pred.}\cdots\texttt{.pred.is-zero.,} \\
& \quad\ \ \texttt{succ.pred.succ.,\ succ.pred.succ.pred.,\ ...,} \\
& \quad\ \ \texttt{succ.pred.pred.succ.,\ succ.pred.pred.succ.pred.,\ ...,} \\
& \quad \texttt{pred.succ.,\ pred.succ.pred.,\ ...,} \\
& \quad \texttt{pred.pred.succ.,\ pred.pred.succ.pred.,\ ...,} \\
& \qquad\qquad \vdots \\
& \quad \texttt{succ.succ.,\ succ.succ.pred.,\ succ.succ.pred.pred.,\ ...,} \\
& \quad \texttt{succ.succ.succ.,\ succ.succ.succ.pred.,\ ...,} \\
& \quad \texttt{succ.succ.succ.succ.,\ succ.succ.succ.succ.pred.,\ ...,} \\
& \qquad\qquad \vdots \\
& \quad \}
\end{aligned}
$$

Dále zkonstruujeme výsledek aplikace $B_n\texttt{.succ}$ operace `succ` nalezení hodnoty atributu množiny rozvitých atributů B_n. Tedy vybereme všechny atributy s prefixem `succ.` a tento prefix odstraníme.

$$
\begin{aligned}
B_n\texttt{.succ} = {} & \{\varepsilon\texttt{.}, && (1) \\
& \quad \texttt{pred.,\ pred.pred.,\ ...,} && \\
& \quad\ \ \texttt{pred.pred.}\cdots\texttt{.pred.is-zero.,} && (1) \\
& \quad \texttt{pred.succ.,\ pred.succ.pred.,\ ...,} && (4) \\
& \quad \texttt{pred.pred.succ.,\ pred.pred.succ.pred.,\ ...,} && (5) \\
& \qquad\qquad \vdots && \\
& \quad \texttt{succ.,\ succ.pred.,\ succ.pred.pred.,\ ...,} && (2)+(3) \\
& \quad \texttt{succ.succ.,\ succ.succ.pred.,\ ...,} && (6) \\
& \quad \texttt{succ.succ.succ.,\ succ.succ.succ.pred.,\ ...,} && (7) \\
& \qquad\qquad \vdots && \\
& \quad \}
\end{aligned}
$$

Řádky označené stejným číslem v množinách $B_n\texttt{.succ}$ a B_{n+1} si vzájemně odpovídají. Proto jsou množiny $B_n\texttt{.succ}$ a B_{n+1} stejné.

V důkazu druhého tvrzení předpokládejme, že m je různé od n. Pak v důsledku zřetězení a uzavřenosti na prefixy budou obě množiny obsahovat atri-

but $\underbrace{\texttt{pred.}\cdots\texttt{.pred.}}_{min(m,n)\text{-krát}}$. Jelikož neexistuje pravidlo, které by definovalo atribut `is-zero.` jinak, než jako atribut objektu v globálním atributu `zero`, bude pouze jedna z množin B_m a B_n obsahovat atribut $\underbrace{\texttt{pred.}\cdots\texttt{.pred.}}_{min(m,n)\text{-krát}}\texttt{is-zero.}$. A tedy budou množiny B_m a B_n různé. □

Z Lemma 2 vyplývá, že množina $\{B_0, B_1, \dots\}$ je izomorfní s $\mathbb{N}_0$, kde operace následníka přirozeného čísla odpovídá operaci `succ` nalezení hodnoty atributu. Díky izomorfismu a tomu, že objekt v globálním atributu `zero` má právě ty atributy, které dostaneme uzávěrem `[is-zero.]`, můžeme zavést značení $0 \equiv$ `[is-zero.]` a $n \equiv (n-1)$`.succ`.

V případě definice operace, či relace, na číslech vytváříme pomocné objekty, které mají krom atributů pro čísla také atributy pro argumenty a výsledek. Názvy atributů argumentů píšeme s prefixem `arg2`, `arg3`,..., v případě binární operace jen `arg`, následované názvem definované operace. Například druhý argument binární operace sčítání uložíme do atributu `arg+`. Výsledek poté ukládáme do atributu s názvem příslušné operace, například `+`.

Sčítání

Obecně postupujeme dle vzorce $a + b = (a + 1) + (b - 1)$. V případě, kdy k libovolnému číslu přičítáme nulu, dostáváme totéž číslo.

```
1 succ., arg+.is-zero.  →  +.{ε}
2 succ., arg+.pred.  →  succ.arg+.{arg+.pred}, +.{succ.+}
```

Zdrojový kód 4: Sčítání

Objekt $x \in X$ splní první pravidlo, když pokud má atribut `succ` a současně má v atributu `arg+` objekt s atributem `is-zero`, pak má v atributu `+` uložen objekt x. Tím jsme vyřešili přičítání nuly. Výraz `arg+.pred`, na pravé straně druhého pravidla, je zápisem operace vícenásobného nalezení hodnoty atributu přiřazující libovonému objektu $x \in X$ objekt uložený v atributu `pred` objektu v atributu `arg+` objektu x. Stejně tak výraz `succ.+` je zápisem operace vícenásobného nalezení hodnoty atributu přiřazující libovolnému objektu $x \in X$ objekt uložený v atributu `+` následníka, tedy objektu v atributu `succ`, objektu x. Obecný postup zachytíme druhým pravidlem, které objekt $x \in X$ splní, když jestliže má atribut `succ` a v atributu `arg+` má objekt s atributem `pred`, tak

1. se v atributu `arg+` objektu, který je uložen v atributu `succ` objektu x, nachází stejný objekt, jaký je v atributu `pred` objektu v atributu `arg+` objektu x, a

2. má v atributu `+` objekt, který se také nachází v atributu `+` objektu uloženého v atributu `succ` objektu x.

Pravidla pro sčítání se dají použít obecně na jakékoliv objekty, které mají definovány atributy `succ` a `pred`. Stačí pouze doplnit pravidlo zachycující krajní případ, stejně jako jsme pomocí prvního pravidla učinili pro přirozená čísla.

Jelikož by byli popisy následujících pravidel dosti podobné jako u operace sčítání, nebudeme dále definovaná pravidla takto dopodrobna popisovat.

Odčítání

Když odčítáme od libovolného čísla nulu, dostaváme opětovně toto číslo. V případě odčítání dvou nenulový čísel použijeme vzorec $a-b = (a-1)-(b-1)$. Vzhledem k tomu, že definuje odčítání na přirozených číslech, nebudeme situaci odčítání nenulového čísla od nuly vůbec řešit. Jelikož tento případ nebude definovaný, nemůže se nám stát, že by nula měla v atributu `arg-` nenulové číslo a současně měla atribut `-` .

```
1 succ., arg-.is-zero.  →  -.{ε}
2 pred., arg-.pred.  →  pred.arg-.{arg-.pred}, -.{pred.-}
```

Zdrojový kód 5: Odčítání

Násobení

Operaci násobení si označíme symbolem $*$, jak je v programování běžné. Pokud bude první argument operace násobení nula, výsledkem bude tento argument. Jinak budeme násobit podle vzorce $a * b = ((a-1) * b) + b$.

```
1 is-zero., arg*.succ.  →  *.{ε}
2 succ., arg*.succ.  →
     pred.arg*.{arg*}, pred.*.arg+.{arg*}, *.{pred.*.+}
```

Zdrojový kód 6: Násobení

Vysvětleme si pravou stranu druhého pravidla, pokud máme k dispozici objekt $x \in X$ splňující jeho levou stranu. Výrazem `pred.arg*.{arg*}` řekneme, že objekt v atribut `pred` objektu x má v atributu `arg*` stejný objekt, jako má objekt x v atributu `arg*` . To nám umožňuje provedení části vzorce $(a-1) * b$. Přičtení čísla b k číslu $(a-1) * b$ umožníme výrazem `pred.*.arg+.{arg*}`. Kdy atributu `arg+` objektu, který se nachází v atributu `*` objektu uloženého v atributu `pred` objektu x, přiřadíme stejný objekt jako je uložen v atributu `arg*` objektu x. A posledním výrazem `*.{pred.*.+}` pouze sdělujeme, že v atributu `*` objektu x se nachází výsledek součtu čísel b a $(a-1) * b$.

Dělení

Dvě přirozená čísla můžeme dělit v případě, že se nepokoušíme dělit nulou nebo je výsledkem dělení přirozené číslo. Pak probíhá výpočet podle vzorce $a/b = (a-b)/b+1$ s tím, že pokud dělíme nulu nenulovým číslem, je výsledkem ona nula.

```
1 is-zero., arg/.pred.  →  /.{ε}
2 succ., arg/.pred.  →  arg-.{arg/}, -.arg/.{arg/}, /.{-./.succ}
```

Zdrojový kód 7: Dělení

Pro objekt $x \in X$ splňující levou stranu druhého pravidla si rozebereme pravou stranu tohoto pravidla. Nejprve výrazem `arg-.{arg/}` řekneme, že v atributu `arg-` objektu x je uložen stejný objekt, jaký se nachází v jeho atributu `arg/`. Tím se, pokud existuje, uloží výsledek rozdílu $a-b$ do atributu - objektu x. Dále se má provést podíl $(a-b)/b$. To zajistíme výrazem `-.arg/.{arg/}`, ve kterém atributu `arg/` objektu uloženého v atributu - objektu x přiřadíme objekt nacházející se v atributu `arg/` objektu x. A posledním výrazem do atributu `arg/` uložíme objekt, který je uložen v atributu `succ` objektu uloženého v atributu `/` objektu v atributu - objektu x.

Rovnost

Nejprve definujeme rovnost dvou nul. Poté řekneme, že pokud se dvě čísla rovnají, rovnají se i jejich následníci.

```
1 is-zero., arg=.is-zero.  →  =.
2 =.  →  succ.arg=.{arg=.succ}, succ.=.
```

Zdrojový kód 8: Rovnost

Menší než

Relaci menší než definujeme podobně jako relaci rovnosti. Nejprve definujeme, že nula je menší než libovolné nenulové číslo. Pak pokud jsou dvě libovolná čísla v relaci menší než, jsou i jejich následníci v této relaci.

```
1 is-zero., arg<.pred.  →  <.
2 <.  →  succ.arg<.{arg<.succ}, succ.<.
```

Zdrojový kód 9: Menší než

Menší nebo rovno

Relaci menší nebo rovno definujeme využitím již definovaných relací.

```
1 succ., arg<=.succ.  →  arg<.{arg<=}, arg=.{arg<=}, <=.{<, =}
```

Zdrojový kód 10: Menší nebo rovno

Pokud uživatel nezadá špatně zárodek, nemůže dojít k situaci, že by objekt $x \in X$ měl současně atribut `<` a `=` . Proto výrazem `<=.{<, =}` uložíme do atributu `<=` objektu x jen jeden objekt. V případě, že objekt x nemá ani jeden z atributů `<` a `=`, nedefinuje se ani atribut `<=` .

Větší než, větší nebo rovno

Zjistit, zda je číslo a větší než (nebo rovno) číslo b, je totéž jako zjistit, zda je b menší než (nebo rovno) a.

```
1 succ., arg>.succ.  →  arg>.arg<.{ε}, >.{arg>.<}
2 succ., arg>=.succ.  →  arg>=.arg<=.{ε}, >=.{arg>=.<=}
```

Zdrojový kód 11: Větší než, větší nebo rovno

Teď si ukážeme, jakým způsobem probíhá vyhodnocení uživatelského dotazu.

Vyhodnocení uživatelského dotazu `[3, arg+.{3}].+ → 6`

Po zadání uživatelského dotazu `[3, arg+.{3}].+ → 6` dojde k jeho přepsání na výraz `[zero.succ.succ.succ, arg+.{zero.succ.succ.succ}].+ → zero.succ.succ.succ.succ.succ.succ` neobsahující zkratky `3` a `6` . Vytvoří se prázdný objektový kontext. Globální atributy `zero.succ.succ.succ` a `zero.succ.succ.succ.succ.succ.succ` se nahradí výrazy `{}.zero.succ.succ .succ` a `{}.zero.succ.succ.succ.succ.succ.succ` .

Jelikož předpoklad dotazu není zadaný výčtem prvků, dojde k jeho upravení. Při procesu úpravy se vytvoří tři pomocné kontexty, jež se poté sjednotí s prázným kontextem a kontextem daným zárodkem rovným upravenému předpokladu.

Nejprve se upraví výraz `arg+.{{}.zero.succ.succ.succ}` . Vytvoří se tedy minimální kontext daný zárodkem rovným prázdné množině. V tomto kontextu se

vyhodnotí výraz `{}.zero.succ.succ.succ` a v předpokladu dotazu se nahradí rozvitím výsledného objektu. Poté se kontext sjednotí s prázdným kontextem. Na konec se provede operace zřetězení atributu `arg+` se vzniklou množinou rozvitých atributů.

Dále dojde k upravení výrazu `{}.zero.succ.succ.succ` . Vytvoří se tedy minimální kontext daný zárodkem `{}`, v něm se výraz `{}.zero.succ.succ.succ` vyhodnotí a v předpokladu nahradí rozvitím výsledného objektu. Minimální kontext se pak sjednotí s kontextem vzniklým předchozím sjednocením.

Po těchto úpravách je předpoklad dotazu tvaru `[pred.pred.pred .is-zero.,... ,arg+.pred.pred.pred.is-zero.,...].+`, ve kterém kvůli potencionální nekonečnosti dosazovaných množin nevypisujeme všechny jeho prvky. Dojde tedy ještě k vytvoření minimálního kontextu daného zárodkem rovným množině `{pred.pred.pred.is-zero.,... ,arg+.pred.pred.pred.is-zero.,...}`. Výraz `[pred.pred.pred.is-zero.,... ,arg+.pred.pred.pred.is-zero.,...].+` se v něm vyhodnotí a v předpokladu dotazu nahradí rozvitím výsledku. A opět dojde k sjednocení tohoto minimálního kontextu s předchozím.

Předpoklad dotazu již neobsahuje žádný zápis aplikace nalezení hodnoty atributu, proto se vytvoří minimální kontext daný zárodkem rovným předpokladu a sjednotí se s předchozím kontextem. Na závěr se v tomto výsledném kontextu ověří platnost pravidla daného dotazem.

Závěr

Původním cílem této práce mělo být navrhnout a naprogramovat objektově orientovaný programovací jazyk založený na formální konceptuální analýze.

Proto jsme v kapitole 2 začali tím, že jsme se seznámili s formální konceptuální analýzou. Konkrétně jsme zjistili, že zachycuje vztah objektů a jejich atributů. Objasnili jsme si pojmy jako formální kontext a koncept. Zjistili jsme, jaká zobrazení kontext vytváří, co tvoří Galoisovu konexi, co je uzávěrový operátor, co je uzávěrový systém a všechny tyto pojmy dali do souvislosti. Také jsme si uvedli, co je atributová implikace, model systému atributových implikací a co je minimální model obsahující nějakou množinu atributů.

Pokračovali jsme zavedením dvou rozšíření klasického pojmu kontextu, objektového a řetězcového. V obou jsme definovali operaci nalezení hodnoty atributu a ukázali, jak je můžeme reprezentovat tabulkou. U řetězcového kontextu jsme si navíc zavedli množinu rozvitých atributů a operaci zřetězení. Rozvitým kontextem jsme poté oba dříve uvedené kontexty propojili. To vše jsme zapsali v kapitole 3.

Dále jsme v kapitole 4 definovali teoretický základ jazyka jako program, tvar pravidel, zápis množiny rozvitých atributů a kdy množina rozvitých atributů splňuje pravidlo. Zavedli jsme pojem model systému pravidel spolu s podmínkami na něj kladenými. Ukázali jsme si, jak můžeme převést systém pravidel na příslušný systém atributových implikací, a že množina rozvitých atributů, jež je modelem systému pravidel, je i modelem příslušného systému atributových implikací. Dále jsme si řekli, co je minimální kontext daný zárodkem a kdy objekt splňuje pravidlo. Na konci kapitoly jsme si definovali elementární a obecné dotazy a způsob jejich vyhodnocování.

Na závěr jsme v kapitole 5 v zavedeném modelu z kapitoly 4 definovali přirozená čísla včetně některých relací a operacích na nich a ukázali si způsob vyhodnocení uživatelských dotazů.

V průběhu práce se teoretické základy ukázali být složitějšími než jsme předpokládali. Bylo nutné je několikrát přepracovat a přitom není jisté, že předkládáná verze je konečná. Z toho důvodu jsme se rozhodli zůstat pouze u teoretického modelu s tím, že v budoucnu plánujeme vytvořit i jeho implementaci.

Conclusions

The original aim of this thesis was to design and program an object-oriented programming language based on formal concept analysis.

Therefore, in Chapter 2, we began by introducing formal concept analysis. Specifically, we found out that it describe the relationship between objects and their attributes. We clarified terms like formal context and concept. We found out which functions context creates, what make a Galois connections, what is a closure operators, what a closure system is, and interrelated all these terms. We also specify what is attribute implication, the model of the system of attribute implications, and what is the least model which contains any set of attributes.

We continued by definition of two extensions of the classic context, Objects' and Strings'. In both, we defined the operation Finding attribute value and shown how to represent them by a table. At the Strings' context we introduced the set of expanded attributes and the concatenation operation. Then with the Expanded context we linked both previously mentioned contexts. All of this we wrote in Chapter 3.

Then, in Chapter 4, we defined the theoretical base of our language, such as the program, form of the rules, notation of the set of expanded attributes, and when the rule is valid in the set of expanded attributes. We introduced the model of the system of rules along with the conditions we put on it. We shown how to convert a system of rules into a corresponding system of attribute implications, and that a set of expanded attributes which is the model of the system of rules is also the model of the corresponding system of attribute implications. Then we said what is the least context given by a seed and when a rule is valid for object. At the end of the chapter, we defined basic and general queries and how to evaluate them.

Finally, in Chapter 5, we defined natural numbers at the model from Chapter 4, along with some relations and operations on them, and we shown how to evaluate user's queries.

During the work the theoretical basics shown to be more complicated than we expected. It was necessary to rework them several times, and even so it is not sure that this version is the final one. Because of that we decided to stay with the theoretical model only, with the intention of creating its implementation in the future.

A Obsah elektronických dat

`bakalarka.pdf`
: Textem práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech příloh.

`text/`
: Adresář se všechmi soubory potřebnými pro bezproblémové vygenerování PDF dokumentu textu, tj. zdrojový text textu a příloh, vložené obrázky, apod.

Literatura

[1] Bělohlávek, Radim. Introduction to formal concept analysis. *Palacky University, Department of Computer Science, Olomouc*. 2008, roč. 47.

[2] Seibel, Peter. *Practical Common Lisp*. Berkeley, California: Apress, c2005. 499 s. ISBN 9781590592397.

[3] Wille, Rudolf; Ganter, Bernhard. *Formal concept analysis: Mathematical foundations*. Berlin: Springer-Verlag, c1999. 284 s. ISBN 3-540-62771-5.

[4] Krupka, Michal. *Paradigmata programování 3*. 2021. Elektronický učební text.