VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Nástroj pro modelování interpretačních struktur v PL1

Interpretation Structure Modeler for FOL

2013 Vladimír Matějičný

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Vladimír Matějičný**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Nástroj pro modelování interpretačních struktur v PL1
Interpretation Structure Modeler for FOL

Zásady pro vypracování:

Cílem práce je vytvořit návrh a implementaci systému v jazyce C# pro podporu výuky předmětu Matematická logika. Systém se bude týkat problematiky 3D vizualizace úlohy na hledání modelu množiny formulí v PL1 a modelu speciálních axiomů popisujících vlastnosti relací.

Systém bude obsahovat:
1. Grafické prostředí pro vytváření interpretační struktury množiny formulí v PL1.
2. Ověření, zda je daná interpretační struktura modelem množiny formulí.
3. Nápovědu pro vytvoření modelu.
4. Prostředí pro modelování důležitých vlastností relací a jejich vizualizaci.
5. Ověření, zda daná struktura splňuje axiomy popisující vlastnosti relací.

Práce bude obsahovat:
1. Sémantiku PL1.
2. Vymezení podmnožiny PL1, na kterou se bude vizualizace vztahovat.
3. Analýzu, návrh a implementaci systému pro podporu výuky.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Mgr. Martina Číhalová**

Datum zadání: 18.11.2011

Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2013 .....

Rád bych na tomto místě poděkoval všem, kteří mi s touto prací pomohli a byli mi po celou dobu oporou. Především pak Mgr. Martině Číhalové, Ph.D. a Mgr. Marku Menšíkovi, Ph.D. za jejich čas a cenné rady.

Abstrakt

Obsahem této diplomové práce je analýza a návrh systému pro podporu výuky předmětu Matematická logika. Vytvářený systém se bude týkat problematiky 3D vizualizace úlohy na hledání modelu množiny formulí v predikátové logice prvního řádu a modelování důležitých vlastností relací. Nástroj bude také obsahovat nápovědu pro vytvoření modelu množiny. V první části práce, je čtenář seznámen s teoretickými základy z dané oblasti, na které plynule navazuje specifikace požadavků na nový systém. Dále je text práce zaměřen především na analýzu a návrh jednotlivých částí systému, která je doplněna o třídní a sekvenční diagramy UML.

Klíčová slova: Predikátová logika, sémantika, formule, interpretační struktura, model, splintelnost, pravdovostní hodnota, predikát, kvantifikátor, funkce, konstanta, množina, orientovaný graf, uzel, hrana, relace, parser, lexikální analýza, syntaktická analýza

Abstract

The goal of this thesis is to Provide design and analysis of a new system, which could then be used as a teaching tool in subject called Mathematical Logic. The new system will consist of two main parts. The first part will be dealing with the problem of 3D visualization of formula set models. The second part will include an environment for modeling important properties of relations. Our system will also include an aid for creating formula set models. In the first part of the thesis we focus on theoretical background and requirement specification for the new system. Following parts ot the thesis are mainly focused on analysis and design of individual system parts, accompanied with class and sequence UML diagrams.

Keywords: Predicate logic, semantics, formula, interpretation structure, model, satisfiability, truth value, predicate, quantificator, function, constant, set, oriented graph, node, edge, relation, parser, lexical analysis, syntactic analysis

Seznam použitých zkratek a symbolů

PL1	–	Predikátová logika 1. řádu
UML	–	Unified Modeling Language
IS	–	Interpreatční struktura
XML	–	Extensible Markup Language
AST	–	Abstraktní syntaktický strom
WPF	–	Windows Presentation Foundation
XSD	–	XML Schema

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Cílem této diplomové práce je vytvořit návrh a implementaci systému pro podporu výuky předmětu Matematická logika. Především se budeme věnovat problematice 3D vizualizace úlohy na hledání modelu množiny formulí predikátové logiky 1. řádu (PL1). Systém však, mimo jiné, bude obsahovat i prostředí pro modelování důležitých vlastností relací a jejich vizualizaci. Díky tomuto, může nalézt využití i v ostatních předmětech.

Pokud je nám známo, tak v současné době existují pouze dva podobné systémy. Prvním z nich je „Tarski's World" [1], který však nabízí pouze omezené možnosti tvorby interpretačních struktur a hlavně neumožňuje práci s funkčními symboly s aritou $n \geq 1$. Faktem je, že i přes to může dobře posloužit při výuce úplných základů predikátové logiky 1. řádu. Mírným problémem pro studenty může být, že aplikace „Tarski's World" používá pro zápis formulí PL1 poněkud odlišnou notaci než tu, která je vyučována v předmětu Matematická logika. Tyto a další problémy, se snažila řešit aplikace „FOLModeler", která je součástí mé bakalářské práce „3D vizualizace modelů v PL1" [4]. Aktuální práce, respektive výsledný systém, bude tedy vylepšením a rozšířením aplikace „FOLModeler" o nové úlohy a prostředí, případně úpravou chybně navržených částí.

Text této práce je koncipován tak, aby mu porozuměl i čtenář neznalý predikátové logiky. Proto si hned ve 2. kapitole objasníme vybrané pojmy a definice, bez kterých si také lze jen těžko představit korektní provedení analýzy celého systému. V následující kapitole „Specifikace požadavků" si shrneme všechny důležité požadavky na vyvíjenou aplikaci. Dozvíme se zde, co vše, respektive jakou funkcionalitu, lze od nového systému očekávat. Na závěr kapitoly uvedeme také několik základních případů užití, tedy příkladů interakce mezi aplikací a uživatelem, což nám pomůže vytvořit si alespoň zevrubný mentální model o podobě celého systému. Nejdůležitější kapitolou je „Analýza a návrh", která nám poskytne ucelený přehled o podobě systému na té nejnižší úrovni. Mimo jiné si zde popíšeme i všechny důležité algoritmy použité při návrhu, jmenovitě například algoritmus pro vyhodnocení pravdivosti formule v interpretaci, nebo interakci mezi jednotlivými komponentami systému.

2 Teorie

V této kapitole si objasníme vybrané pojmy a definice z předmětné oblasti práce, které budeme využívat v dalších kapitolách. Nejprve začneme s tzv. „naivní teorií množin," kde si vysvětlíme pojmy jako *množina*, *uspořádaná dvojice* či *kartézský součin*. Poté si ujasníme pojmy *relace* a *funkce*, bez kterých nelze plně pochopit závěr kapitoly pojednávající o sémantice predikátové logiky prvního řádu.

Všechny definice a pojmy v této kapitole jsou čerpány z [2], především pak z [3].

2.1 Množiny

Definice 2.1 ***Množina*** *je kolekce vzájemně odlišitelných objektů, které nazýváme jejími prvky. Jestliže je objekt x prvkem množiny S , píšeme $x \in S$. Jestliže x není prvkem S, píšeme $x \notin S$. [2]*

Množina jako taková nemůže obsahovat určitý prvek více než jednou a prvky množiny nejsou nijak seřazeny. Dvě množiny A a B jsou si *rovny* právě tehdy, když obsahují stejné prvky. Pro označení rovnosti dvou množin používáme zápis $A = B$. Vezměme tedy například množiny $A = \{1, 2, 3\}$, $B = \{3, 1, 2\}$ a $C = \{1, 3\}$, pro které platí $A = B$, ale $A \neq C$ a $B \neq C$.

Množinu neobsahující žádné prvky nazýváme *prázdná množina* a označujeme jí symbolem $\emptyset$.

Jestliže všechny prvky množiny A patří zároveň i do množiny B, pak říkáme, že *A je podmnožinou B*, což zapisujeme výrazem $A \subseteq B$. Množina *A je vlastní podmnožinou množiny* B právě tehdy, když $A \subseteq B$, ale $A \neq B$. Jinými slovy existuje prvek x, takový že $x \in B$, ale $x \notin A$. To, že je A vlastní podmnožinou B, zapisujeme výrazem $A \subset B$. Pro množiny A, B a C z předchozího odstavce například platí následující: $A \subseteq B$, $B \subseteq A$, $C \subseteq B$ a $B \nsubseteq C$.

Uspořádaná dvojice prvků a a b, označovaná (a, b), je formálně definována jako množina $(a, b) = \{\{a\}, \{a, b\}\}$. Na rozdíl od množin u uspořádaných dvojic již záleží na pořadí jednotlivých prvků. Dvojice (a, b) je tedy něco jiného než (b, a).

Definice 2.2 ***Kartézský součin*** *množin A a B, označovaný jako A x B, je množina všech uspořádaných dvojic, kde první prvek z dvojice patří do množiny A a druhý do množiny B:*

$$A \times B = \{(a, b) | a \in A, b \in B\}$$

[2]

Je-li jedna z množin prázdná, je i kartézský součin prázdná množina. Kartézský součin n množin $A_1 \times A_2 \times \cdots \times A_n$ je množina $n - tic$

$$A_1 \times A_2 \times \cdots \times A_n = \{(a_1, a_2, \ldots, a_n) | a_i \in A_i, i = 1, 2, \ldots, n\}$$

Příklad 2.1

$$A = \{a, b, c\}, B = \{d, e\}$$

$$A \times B = \{(a,d),(a,e),(b,d),(b,e),(c,d),(c,e)\}$$

■

2.2 Relace

Definice 2.3 ***Relace*** *R na množinách $A_1, A_2, \ldots A_n$ je libovolná podmnožina kartézského součinu $A_1 \times A_2 \times \cdots A_n$. Relace na n množinách se nazývá n-ární relace. Jestliže $n = 2$, jedná se o binární relaci. Jestliže $n = 3$, jedná se o ternární relaci. [2]*

Příklad 2.2
Vezměme kartézský součin z příkladu 2.1. Jedna z možných binárních relací $R \subseteq \{a,b,c\} \times \{d,e\}$ může být

$$R = \{(a,d),(c,d),(c,e)\}$$

■

2.3 Funkce

Definice 2.4 ***Funkce*** *f z množiny A do množiny B je binární relace $f \subseteq A \times B$ taková, že pro každé $a \in A$ existuje právě jedno $b \in B$ takové, že $(a,b) \in f$. Množina A se nazývá* ***definiční obor*** *funkce f, množina B se nazývá* ***obor hodnot*** *funkce f. [2]*

Takto uvedená definice odpovídá tzv. *totální funkci*. „Funkce, která vznikne z dané funkce zúžením (restrikce) množiny, na niž je definována nazýváme *parciální*. Je-li f funkce definovaná na nějaké množině M a je-li $P \subset M$ podmnožinou množiny M, pak funkce f_P definovaná na P předpisem $f_P(x) = f(x)$ pro $x \in P$ je parciální funkce k funkci f na množině P.“ [5]. V predikátové logice 1. řádu používáme pro interpretaci funkčních symbolů pouze totální funkce.

Výrazem $f : A \to B$ obvykle značíme funkci f z množiny A do množiny B. Místo zápisu $(a,b) \in f$ můžeme také použít $b = f(a)$, neboť volbou prvku a je prvek b jednoznačně určen. Funkce $f : A \to B$ tedy každému prvku z A přiřazuje právě jeden prvek z B.

Příklad 2.3
Vezměme relaci $R = \{(a,d),(c,d),(c,e)\}$ z příkladu 2.2. Tato relace není funkcí, neboť pro prvek $c \in A$ existují dva prvky $d \in B$ a $e \in B$. Relace $R' = \{(a,d),(c,e)\}$ již funkcí je, protože pro $a \in A$, $c \in A$ existuje právě jeden prvek $d \in B$ respektive $e \in B$. ■

2.4 Formální teorie jazyků

Definice 2.5

- **Abeceda** je libovolná konečná množina, často jí značíme $\sum$. Prvky abecedy nazýváme **symboly**

- **Slovem** nebo **řetězcem**, nad abecedou $\sum$ rozumíme libovolnou konečnou posloupnost prvků množiny $\sum$
- Výrazem $\sum^*$ značíme množinu všech slov nad abecedou $\sum$
- **Formální jazyk** nad abecedou $\sum$, je libovolná množina slov v abecedě $\sum$, tedy libovolná podmnožina $\sum^*$

[2]

Definice 2.6 ***Bezkontextová gramatika*** *je uspořádaná čtveřice* $G = (\pi, \sum, S, P)$*, kde:*

- Π *je konečná množina neterminálních symbolů (neterminálů)*
- $\sum$ *je konečná množina terminálních symbolů (terminálů), přičemž* $\pi \cap \sum = \emptyset$
- $S \in \pi$ *je počáteční (startovací) neterminál*
- P *je konečná množina pravidel typu* $A \rightarrow \beta$*, kde*
  - A *je neterminál, tedy* $A \in \pi$
  - β *je řetězec složený z terminálů a neterminálů, tedy* $\beta \in (\pi \cup \sum)^*$

[2]

Definici bezkontextové gramatiky 2.6 zde uvádíme spíše pro základní přehled a nebudeme jí nijak dále popisovat. Podrobný popis teorie formálních jazyků a gramatik můžeme nalézt například v [2]. Bezkontextové gramatiky využijeme v pozdějších kapitolách například pro vymezení podmnožiny formulí PL1.

2.5 Sémantika PL1

Predikátová logika 1. řádu (dále jen PL1) formalizuje úsudky a výroky o vlastnostech předmětů a vztazích mezi těmito předměty na pevně dané předmětné oblasti - *univerza*. „Výroková logika na rozdíl od PL1 nedokáže zachytit vazby mezi vnitřními komponentami elementárních výroků. PL1 je tedy zobecněním výrokové logiky, kterou můžeme považovat za logiku nultého řádu.“ [3] Abychom mohli rozhodovat o pravdivosti formulí PL1 je třeba definovat jejich jazyk (abeceda a gramatika). Dále musíme definovat sémantiku těchto formulí - tj. význam logických spojek a interpretace speciálních symbolů jako funkce nebo predikáty.

Část textu této podkapitoly je založena na předchozí práci [4]. Všechny další definice v této kapitole jsou již čerpány z [3].

2.5.1 Jazyk PL1

Definice 2.7 *Jazyk predikátové logiky prvního řádu:*

1. ***Abeceda predikátové logiky*** *je tvořena následujícími skupinami symbolů:*

   (a) ***Logické symboly***

   i. *předmětové (individuové) proměnné:* $x, y, z, \ldots$ *(případně s indexy)*

   ii. *symboly pro spojky:* $\neg, \wedge, \vee, \supset, \equiv$

   iii. *symboly pro kvantifikátory* $\exists, \forall$

   iv. *případně binární predikátový symbol* $=$ *(predikátová logika s rovností)*

   (b) ***Speciální symboly*** *(určují specifiku jazyka)*

   i. *predikátové symboly:* $P, Q, R, \ldots$ *(případně s indexy)*

   ii. *funkční symboly:* $f, g, h, \ldots$ *(případně s indexy)*

   Ke každému funkčnímu a predikátovému symbolu je přiřazeno nezáporné číslo n $(n \geq 0)$*, tzv.****arita****, udávající počet individuových proměnných, které jsou argumenty funkce nebo predikátu.*

   (c) ***Pomocné symboly***

   i. *závorky (), {}, []*

2. ***Gramatika,*** *která udává, jak tvořit:*

   (a) ***termy***

   i. *každý symbol proměnné je term*

   ii. *jsou-li* $t_1, \ldots, t_n (n \geq 0)$ *termy a je-li* f n*-ární funkční symbol, pak výraz* $f(t_1, \ldots, t_n)$ *je term; pro* $n = 0$ *se jedná o nulární funkční symbol, neboli individuovou konstantu (značíme* $a, b, c, \ldots$*)*

   iii. *jen výrazy dle i. a ii. jsou termy*

   (b) ***atomické formule***

   i. *je-li* P n*-ární predikátový symbol a jsou-li* $t_1, \ldots, t_n$ *termy, pak výraz* $P(t_1, \ldots, t_n)$ *je atomická formule*

   ii. *jsou-li* t_1 *a* t_2 *termy, pak výraz* $(t_1 = t_2)$ *je atomická formule*

   (c) ***formule***

   i. *každá atomická formule je formule*

   ii. *je-li výraz* A *formule, pak* $\neg A$ *je formule*

   iii. *jsou-li výrazy* A *a* B *formule, pak výrazy* $(A \wedge B)$*,* $(A \vee B)$*,* $(A \supset B)$*,* $(A \equiv B)$ *jsou formule*

   iv. *je-li* x *proměnná a* A *formule, pak výrazy* $\exists x A$ *a* $\forall x A$ *jsou formule*

   v. *jen výrazy dle i. – iv. jsou formule*

Pokud je arita funkčního symbolu $n = 0$, pak hovoříme o tzv. *individuové konstantě.* Nejedná se však o pravé logické konstanty, protože podle definice 2.7, stejně jako každý funkční symbol, podléhá interpretaci. Tyto konstanty značíme jako $a, b, c, \ldots$ (případně s indexy).

Pro výše definovaný jazyk je charakteristické to, že jediným přípustným typem proměnných jsou *individuové proměnné* (proměnná probíhající určitou předmětnou oblast). Pouze tyto proměnné lze vázat kvantifikátory.

Definice 2.8

Výskyt proměnné x ***ve formuli A je vázaný,*** *jestliže je součástí nějaké podformule* $\forall x B(x)$ *nebo* $\exists x B(x)$ *formule A.*

Proměnná x ***je vázaná ve formuli A,*** *má-li v A vázaný výskyt. Výskyt proměnné* x *ve formuli A, který není vázaný, nazýváme* ***volný.***

Proměnná x ***je volná ve formuli A,*** *má-li v A volný výskyt.*

Formule, v níž každá proměnná má buď všechny výskyty volné nebo všechny výskyty vázané, se nazývá ***formulí s čistými proměnnými.***

Formule se nazývá ***uzavřenou,*** *neobsahuje-li žádnou volnou proměnnou. Formule, která obsahuje aspoň jednu volnou proměnnou se nazývá* ***otevřenou.***

Symbolem $A(x/t)$ *označujeme formuli, která vznikne z formule A* ***korektní substitucí termu t za proměnnou x***

Vezměme například v úvahu formuli $\forall x[P(x) \supset Q(x, y)]$. Výskyt proměnné x je v této formuli vázaný, zatímco výskyt proměnné y je volný. Jedná se tedy o formuli otevřenou. Vezměme v úvahu jinou formuli $P(x) \supset \forall x Q(x)$. Z definice 2.8 vyplívá, že výskyt proměnné x v podformuli $P(x)$ je volný, zatímco její výskyt v podformuli $\forall x Q(x)$ je vázaný. Opět se tedy jedná o formuli otevřenou. Příkladem uzavřené formule může být $\forall x \exists y[P(x, y) \wedge Q(x, y)]$, výskyt obou proměnných x a y je v této formuli vázaný.

2.5.2 Interpretace formulí

V předchozí podkapitole jsme definovali abecedu a gramatiku jazyka PL1. Převeďme tedy nyní následující dva jednoduché výroky z přirozeného jazyka do jazyka predikátové logiky:

1. „Všichni hudebníci jsou umělci.“

2. „Všechna celá čísla jsou přirozenými čísly.“

První i druhý výrok můžeme formalizovat následující formulí: $\forall x[P(x) \supset Q(x)]$. Vidíme tedy, že oba výroky lze popsat ekvivalentní formulí. Je evidentní, že první výrok je pravdivý a druhý výrok pravdivý není. Tuto informaci nám však samotná formule představující tyto výroky nesděluje. Pokud se chceme bavit o pravdivosti či nepravdivosti výroků na základě formulí, které je představují, je třeba tyto formule nejprve nějakým způsobem interpretovat.

První věc, kterou si musíme při interpretaci formule ujasnit, je o čem daná formule hovoří. Musíme tedy vymezit tzv. předmětnou oblast, neboli obor proměnnosti (individuových) proměnných. Tato oblast je *neprázdnou* množinou a nazývá se **universum diskursu.** Prvky této množiny nazýváme **individua.** Dále každému n-árnímu **predikátovému symbolu** přiřadíme určitou n-ární **relaci** nad universem, která bude vyjadřovat vztahy mezi jednotlivými prvky tohoto universa. Pokud se jedná o unární predikátový symbol (jeho arita je n = 1), přiřadíme mu jistou podmnožinu universa. Stejně tak i n-arnímu **funkčnímu symbolu** přiřadíme určitou n-ární **funkci** nad universem. V neposlední řadě musíme interpretovat i všechny **individuové konstanty**, které jsou v podstatě nulárními funkcemi a musíme jim přiřadit **jedno** určité **individuum** z universa. Jakmile je formule takto interpretována, můžeme začít hovořit o **pravdivosti či nepravdivosti v dané interpretaci.**

Definice 2.9

Interpretace jazyka predikátové logiky 1. řádu *je tato trojice objektů (která je někdy nazývána* ***interpretační struktura****):*

- *Neprázdná množina M, která se nazývá* ***universum diskursu*** *a její prvky jsou* ***individua.***
- *Interpretace funkčních symbolů jazyka, která přiřazuje každému n-árnímu funkčnímu symbolu f určité* ***zobrazení*** $\boldsymbol{f_M : M^n \to M}$.
- *Interpretace predikátových symbolů jazyka, která přiřazuje každému n-árnímu predikátovému symbolu p jistou n-ární relaci* $\boldsymbol{p_M}$ *nad M, tj.* ***podmnožinu Kartézského součinu*** $\boldsymbol{M^n}$.

Sestavme nyní interpretační struktury k formulím výše uvedených výroků. Připomeňme, že oba výroky jsou formalizovány formulí $\forall x[P(x) \supset Q(x)]$.

- Interpretační struktura 1. výroku:
  Universum = lidé
  P = x je hudebníkem
  Q = x je umělec
- Interpretační struktura 2. výroku:
  Universum = reálná čísla $\mathbb{R}$
  P = celá čísla $\mathbb{Z}$
  Q = přirozená čísla $\aleph$

U takto interpretovaných formulí můžeme nyní rozhodovat o jejich pravdivosti.

2.5.3 Pravdivost formulí

Definice 2.10

Ohodnocení (valuace) individuových proměnných *je zobrazení e, které každé proměnné x přiřazuje hodnotu $e(x) \in M$ (prvek univerza).* ***Ohodnocení termů*** *e^* indukované ohodnocením proměnných e je induktivně definováno takto:*

- $e^*(x) = e(x)$
- $e^*(f(t_1, t_2, \ldots, t_n)) = f_M(e^*(t_1), e^*(t_2), \ldots, e^*(t_n))$*, kde f_M je funkce přiřazená v dané interpretaci funkčnímu symbolu f*

Definice 2.11

Pravdivost formule A v interpretaci I pro ohodnocení e *individuových proměnných (což značíme $\models_I A[e]$ – formule A je pravdivá v I pro e, nebo také A* ***je splněna v I ohodnocením e****, je definována v závislosti na tvaru formule:*

1. *Je-li A atomická formule tvaru:*
   (a) *$P(t_1, \ldots, t_n)$, kde P je predikátový symbol (různý od =) a $t_1, \ldots, t_n$ jsou termy, pak $\models_I A[e]$, jestliže platí $(e^*(t_1), e^*(t_2), \ldots, e^*(t_n)) \in p_M$, kde p_M je relace přiřazená interpretací I symbolu P – obor pravdivosti P. Tedy individua, která jsou hodnotou termů $t_1, \ldots, t_n$, jsou v relaci p_M.*
   (b) *$(t_1 = t_2)$, pak $\models_I A[e]$, jestliže platí $e^*(t_1) = e^*(t_2)$, tj. oba termy jsou realizovány týmž individuem.*
2. *Je-li A složená formule dle bodu 2. c) definice 1.1, tj. je-li tvaru:*
   (a) *$\neg B$, pak $\models_I A[e]$ jestliže neplatí $\models_I B[e]$*
   (b) *$B \wedge C$, pak $\models_I A[e]$, jestliže platí $\models_I B[e]$ a $\models_I C[e]$*
   (c) *$B \vee C$, pak $\models_I A[e]$, jestliže platí $\models_I B[e]$ nebo $\models_I C[e]$*
   (d) *$B \supset C$, pak $\models_I A[e]$, jestliže neplatí $\models_I B[e]$ nebo platí $\models_I C[e]$*
   (e) *$B \equiv C$, pak $\models_I A[e]$, jestliže platí $\models_I B[e]$ a $\models_I C[e]$, nebo neplatí $\models_I B[e]$ a neplatí $\models_I B[e]$*
3. *Je-li A formule tvaru:*
   (a) *$\forall x B$, pak $\models_I A[e]$, jestliže pro* libovolné *individuum $i \in M$ platí $\models_I B[e(x/i)]$, kde $e(x/i)$ je valuace stejná jako e až na to, že přiřazuje proměnné x individuum i.*
   (b) *$\exists x B$, pak $\models_I A[e]$, jestliže pro* alespoň jedno *individuum $i \in M$ platí $\models_I B[e(x/i)]$, kde $e(x/i)$ je valuace stejná jako e až na to, že přiřazuje proměnné x individuum i.*

Z definice kvantifikátorů vyplývá, že pokud je universum diskursu konečná množina $M = \{a_1, a_2, \ldots, a_n\}$, pak platí následující ekvivalence formulí:

- $\forall x A(x) \Leftrightarrow A_1(x) \wedge A_2(x) \wedge \ldots \wedge A_n(x)$

- $\exists x A(x) \Leftrightarrow A_1(x) \vee A_2(x) \vee \ldots \vee A_n(x)$

Všeobecný kvantifikátor $\forall$ je tedy zobecněním konjunkce a existenční kvantifikátor $\exists$ je zobecněním disjunkce. Tento fakt využijeme později při návrhu algoritmu pro vyhodnocení pravdivosti formule v interpretaci, neboť universum navrženého systému je konečné.

Vraťme se nyní k formulí $\forall x[P(x) \supset Q(x)]$ představující oba výroky z 2.5.2 a s pomocí definic 2.10 a 2.11 vyhodnoťme její pravdivost.

Nejprve se zaměřme na první z výroků - „*Všichni hudebníci jsou umělci*“. Jestliže má být tato formule v dané interpretaci pravdivá, musí pro libovolné individuum z universa diskursu (množina lidí) platit $\models_I A[e(x/i)]$. Tato formule je složenou formulí dle definice 2.11 bod 2d. Formule bude tedy v dané interpretaci I pravdivá, jestliže neplatí $\models_I P[e]$ nebo platí $\models_I Q[e]$. Jinými slovy nebude pravdivá pouze v případě, že v universu lidí nalezneme takového člověka, který by byl muzikantem (P), ale nebyl by umělcem (Q). A protože množina muzikantů je podmnožinou množiny umělců, žádného takového muzikanta nenalezneme. Formule je tedy v dané interpretaci pravdivá.

Zaměřme se nyní na druhý výrok - „*Všechna celá čísla jsou přirozenými čísly.*“. Dojdeme k závěru že druhá formule nebude pravdivá pokud nalezneme celé číslo, které není není přirozeným číslem. Takové číslo ovšem existuje, například valuace $e(x) = -3$ nebo $e(x) = -21$. Z toho plyne, že formule není v dané interpretaci pravdivá. Vidíme tedy, že pravdivost formule opravdu závisí na její interpretaci.

Pravdivostní hodnota formule nezávisí na hodnotě vázaných proměnných. Obsahuje-li však formule nějaké volné proměnné, můžeme vyhodnotit její pravdivost v interpretaci pouze v **závislosti na ohodnocení** (valuaci) **volných proměnných.** Pravdivostní hodnota formule může být pro různé valuace volných proměnných rozdílná. Pro příklad uveďme tuto formuli:

$$\forall x P(x, y)$$

Jako universum diskursu zvolme množinu přirozených čísel $\aleph$. Binárnímu predikátu P přiřaďme relaci $\{(x, y) \in \aleph \times \aleph | x \geq y\}$. Říkáme tedy, že všechna přirozená čísla x jsou větší nebo rovna určitému číslu y. Přiřadíme-li nyní proměnné y např. číslo 5, bude formule nepravdivá. Naopak pokud přiřadíme číslo 0 bude formule v této interpretaci pravdivá.

Definice 2.12

Formule A je splnitelná v interpretaci I, *jestliže* existuje *ohodnocení e proměnných takové, že platí* $\models_I A[e]$.

Formule A je pravdivá v interpretaci I, *značíme* $\models_I A$, *jestliže pro* všechna *možná ohodnocení e individuových proměnných platí, že* $\models_I A[e]$.

Model formule A *je interpretace I, ve které je A pravdivá.*

Formule A je splnitelná, *jestliže existuje interpretace I, ve které je splněna, tj. jestliže existuje interpretace I a valuace e takové, že* $\models_I A[e]$.

Formule A je tautologií *(logicky pravdivá), značíme* $\models A$, *jestliže je pravdivá v každé interpretaci.*

Formule A je kontradikcí, *jestliže nemá model, tedy neexistuje interpretace I, která by formuli A splňovala.*

Model množiny formulí *$\{A_1, \ldots, A_n\}$ je taková interpretace I, ve které jsou pravdivé všechny formule $A_1, \ldots, A_n$.*

Formule B logicky vyplývá z formulí *$A_1, \ldots, A_n$, značíme $A_1, \ldots, A_n \models B$, jestliže B je pravdivá v každém modelu množiny formulí $A_1, \ldots, A_n$. Tedy pro každou interpretaci I, ve které jsou pravdivé formule $A_1, \ldots, A_n$ ($\models_I A_1, \ldots, \models_I A_n$) platí, že je v ní pravdivá také formule B ($\models_I B$).*

Definice 2.13 ***Formule*** *A, B* ***jsou sémanticky ekvivalentní****, jestliže pro všechny interpretace I a všechny valuace e mají stejná pravdivostní ohodnocení.*

Příkladem dvou sémanticky ekvivalentních formulí jsou formule $\forall x(P(x) \supset Q(x))$ a $\forall x(\neg P(x) \vee Q(x))$. Uvažujme následující jednoduchou formuli:

$$\forall x P(f(x), x)$$

Jako universum diskursu zvolme množinu přirozených čísel $\aleph$. Interpretujme predikátový symbol P jako relaci „větší než“ - $\{(x, y) \in \aleph \times \aleph | x > y\}$ a funkčnímu symbolu f přiřaďme funkci $\aleph \to \aleph : f(x) = x^2$. Tato interpretační struktura (IS) není modelem formule, je v této IS nepravdivá a není splnitelná.

Pozměňme předchozí interpretační strukturu formule a pro funkční symbol f zvolme funkci $\aleph \to \aleph : f(x) = 2x + 1$. Pro všechna ohodnocení e nyní platí $\models_I A[e]$, protože každé přirozené číslo je menší než jeho dvojnásobek zvětšený o 1. Formule je **pravdivá interpretaci v této I** a tato interpretace je tedy zároveň i **modelem dané formule.**

Pokud bychom chtěli vědět zda má vůbec smysl hledat model formule, musíme zjistit zda se nejedná o **kontradikci**, tedy zda existuje nějaký model dané formule. Jako příklad si uveďme formuli $\forall x[P(x) \wedge \neg P(x)]$, pro niž neexistuje interpretace, ve které by byla pravdivá. Opakem kontradikce je **tautologie**. Pokud je formule tautologií, tak každá její interpretace je zároveň i jejím modelem. Příkladem takovéto formule je $\forall x[P(x) \vee \neg P(x)]$.

Definice 2.14 *Formule A predikátové logiky je v prenexním tvaru, má li podobu* $\boldsymbol{Q_1 x_1 Q_2 x_2 \ldots Q_n x_n B}$ *kde*

- $n \geq 0$ a pro každé $i = 1, 2, \ldots, n$ je Q_i buď všeobecný $\forall$ nebo existenční $\exists$ kvantifikátor
- $x_1, x_2, \ldots, x_n$ jsou navzájem různé individuové proměnné
- B je formule utvořená z elementárních formulí pouze užitím výrokových funktorů $\neg, \vee, \wedge$

Výraz $Q_1 x_1 Q_2 x_2 \ldots Q_n x_n$ se nazývá ***prefix*** *a B* ***otevřeným jádrem****.*

2.6 Vlastnosti relací

Predikátová logika 1. řádu je postačující pro formalizaci mnohých matematických a jiných teorií. Můžeme pomocí ní vyjadřovat i důležité vlastnosti relací. Například to, že je nějaká relace $R \subseteq A \times A$ *symetrická* můžeme slovně zapsat takto:

pro všechna $a, b \in A$ platí, že pokud $(a, b) \in R$, pak $(b, a) \in R$

čemuž odpovídá následující formule PL1:

$$\forall x \forall y [R(x, y) \supset R(y, x)]$$

Mezi další důležité vlastnosti relací patří například:

$reflexivita$	$\forall x R(x, x)$
$i - reflexivita$	$\forall x \neg R(x, x)$
$symetrie$	$\forall x \forall y [R(x, y) \supset R(y, x)]$
$asymetrie$	$\forall x \forall y [R(x, y) \supset \neg R(y, x)]$
$anti - symetrie$	$\forall x \forall y [(R(x, y) \wedge R(y, x)) \supset x = y]$
$transitivita$	$\forall x \forall y \forall z [(R(x, y) \wedge R(y, z)) \supset R(x, z)]$

2.7 Formální teorie

Definice 2.15 *Formální teorie je dána trojicí:*

- *formální jazyk teorie*
- *množina axiómů teorie*
- *množina dedukčních pravidel teorie*

Formální jazyk teorie 1. řádu je jazyk predikátové logiky 1. řádu (viz definice 2.7). Formální jazyk je tedy množina všech dobře utvořených formulí.

Množina axiómů teorie je podmnožina množiny všech dobře utvořených formulí. Sestává se ze dvou částí:

- množiny **logických** axiómů, tedy tautologií
- množiny **speciálních** axiómů. Množina speciálních axiómů charakterizuje pomocí formulí predikátové logiky vlastnosti a vztahy všech objektů primitivními pojmy teorie (tj. speciální predikátové a funkční symboly, speciální konstanty), které v jazyce teorie vystupují. Speciální axiómy jsou voleny tak, aby byly pravdivé v „zamýšlené“ interpretaci předmětné oblasti.

Množina dedukčních pravidel teorie splývá s množinou dedukčních pravidel použitého kalkulu predikátové logiky.

Formální teorie (v širším slova smyslu) je množina všech formulí, které lze odvodit z axiómů teorie pomocí dedukčních pravidel teorie. Vzhledem k tomu, že teorie je plně charakterizována množinou T speciálních axiómů, ztotožňujeme někdy formální teorii T s množinou speciálních axiómů teorie (pojem formální teorie v užším slova smyslu).

Definice formálních teorií zde uvádíme pouze pro úplnost. V prostředí pro modelování důležitých vlastností relací se budeme pouze zabývat tím, zda vytvářená struktura objektů

splňuje vlastnosti a vztahy dané speciálními axiómy určité teorie. Ve výsledném systému budou implementovany následující speciální axiómy z vybraných teorií:

Teorie ostrého uspořádání

- $\forall x \forall y((x < y) \supset \neg(y < x))$ $\quad asymetrie$
- $\forall x \forall y \forall z(((x < y) \wedge (y < z)) \supset (x < z))$ $\quad transitivita$

Teorie rovnosti (ekvivalence)

- $\forall x(x = x)$ $\quad reflexivita$
- $\forall x \forall y((x = y) \supset (y = x))$ $\quad symetrie$
- $\forall x \forall y \forall z(((x = y) \wedge (y = z)) \supset (x = z))$ $\quad transitivita$

Teorie částečného (neostrého) uspořádání

- $\forall x(x \leq x)$ $\quad reflexivita$
- $\forall x \forall y(((x \leq y) \wedge (y \leq x)) \supset x = y)$ $\quad anti - symetrie$
- $\forall x \forall y \forall z(((x \leq y) \wedge (y \leq z)) \supset (x \leq z))$ $\quad transitivita$

Teorie quasi uspořádání

- $\forall x(x \leq x)$ $\quad reflexivita$
- $\forall x \forall y \forall z(((x \leq y) \wedge (y \leq z)) \supset (x \leq z))$ $\quad transitivita$

3 Specifikace požadavků

V této kapitole si nejprve stručně a v několika bodech shrneme základní požadavky na vyvíjený systém. Některé z komplexnějších bodů poté popíšeme více podrobněji. Následně, již poněkud formálněji, rozvineme celý systém na jednotlivé funkční požadavky a případy použití, které budou sloužit jako základ pro pozdější objektovou analýzu a návrh. Vyvíjený systém bude obsahovat následující:

- Grafické prostředí pro vytváření interpretační struktury množiny formulí PL1
- Ověření, zda je daná interpretační struktura modelem množiny formulí
- Nápovědu pro vytvoření modelu množiny formulí
- Prostředí pro modelování důležitých vlastností relací a jejich vizualizaci
- Ověření, zda daná struktura splňuje axiomy popisující vlastnosti relací
- Uložení a otevření vytvořené interpretační struktury, včetně množiny formulí, z a do formátu XML

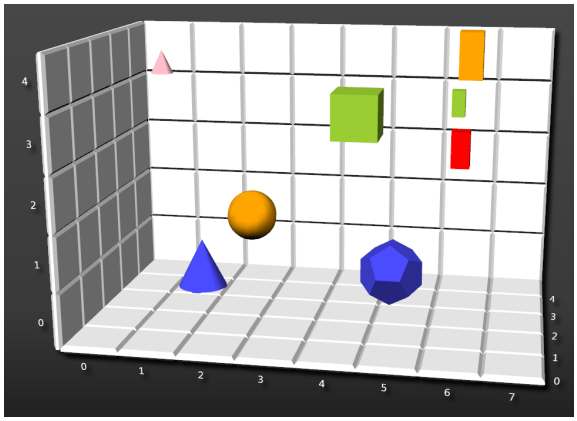
Jak v prostředí pro vytváření IS množiny formulí PL1, tak i v prostředí pro modelování vlastností relací, bude uživateli umožněno vkládat jednotlivé formule pomocí klávesnice, či podobného vstupního zařízení. Dále bude umožněno načtení (tzv. „import“) jedné či více formulí ze souboru nebo schránky (tzv. „clipboard“). Přitom musí být zajištěno, aby do systému mohly být vloženy pouze správně utvořené formule PL1 (viz definice 2.7), vyhovující daným omezením (viz kapitola 4). V případě nekorektního vstupu, systém uživatele vhodným způsobem upozorní, na které pozici v řetězci formule se chyba nachází.

Prostředí pro vytváření IS množiny formulí

Prostředí nabídne možnost sestavení kompletní interpretační struktury množiny formulí PL1. Bude zahrnovat vizualizaci universa v trojrozměrném prostoru s možností jeho úprav - tzn. vložení a smazání individua, nebo úpravu některé z jeho vlastností. Prvky universa budou prostorové tvary s vlastnostmi – *barva*, *velikost*, *pozice* v prostoru a vyobrazený *tvar*. Jednotlivé prvky universa budou umístěny do trojrozměrného systému kartézských souřadnic (příklad na obrázku 1). Prostředí musí také umožnit interpretaci funkčních a predikátových symbolů. Aplikace též musí umožnit uživateli ověřit, zda je vytvořená IS modelem množiny formulí. V případě, že některé predikátové či funkční symboly před tímto ověřením nejsou interpretovány, systém na tento fakt uživatele vhodným způsobem upozorní. Systém jako takový, bude pracovat vždy jen s jednou množinou formulí.

Nápověda pro vytvoření modelu množiny formulí

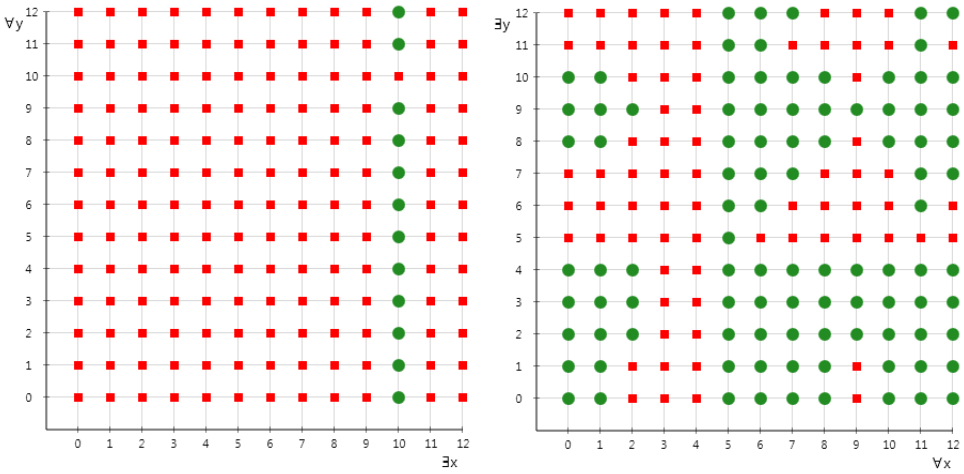
V případě, že vytvořená interpretační struktura není modelem množiny formulí, systém umožní uživateli zobrazit, které z formulí nejsou v této struktuře pravdivé. U všech



Obrázek 1: Zobrazení universa ve 3D prostoru

formulí budou dále, pomocí interaktivního grafu, vizualizovány všechny možné kombinace vztahů nebo vlastností (příklad na obrázku 2). Osy grafu, respektive jejich hodnoty $0, 1, 2, \ldots$, jsou jednotlivá individua $i_1, \ldots, i_n$. Rozměr grafu, nebo jinak řečeno počet hodnot na jeho osách, přímo odpovídá počtu individuí v universu. Jednotlivé body pak reprezentují, zda daná dvě individua jsou či nejsou v relaci. Pokud v relaci nejsou, bod je vykreslen červenou barvou, jinak zelenou.

Systém uživateli umožní z grafu vybrat konkrétní valuaci proměnných a zobrazí detaily vyhodnocení pravdivosti formule při dané valuaci. Máme především na mysli zobrazení pravdivostních hodnot predikátů nebo výsledků funkčních zobrazení. Tímto umožníme uživateli zjistit, kde se nachází chyba a tuto chybu následně opravit. Systém by měl dále zajistit, aby nápověda byla automaticky aktualizována po každé změně interpretační struktury.

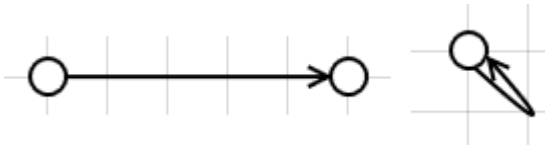


Obrázek 2: Příklad vizualizace pravdivostních hodnot formule $\exists x \forall y Q(x, y)$ a $\forall x \exists y Q(x, y)$ vzhledem k interpretaci

Prostředí pro modelování důležitých vlastností relací

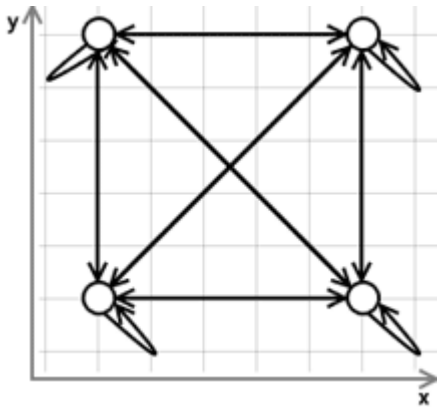
Uživatel bude v tomto prostředí vytvářet „relační strukturu“ za pomocí orientovaného grafu. Systém mu následně umožní ověřit, zda tato struktura má vlastnosti popsané jednotlivými axiómy. Můžeme tedy například ověřit, zda struktura splňuje axiómy reflexivity a transitivity. Prostředí musí uživateli umožnit modelovat veškeré důležité vlastnosti relací z kapitoly 2.6. Dále požadujeme, aby prostředí umožnilo modelovat alespoň struktury splňující speciální axiomy teorií z kapitoly 2.7. Všechny vlastnosti relací z 2.6 a speciální axiomy jednotlivých teorií z 2.7 budou již v systému připraveny, tak aby je uživatel mohl jednoduše vkládat, namísto opakovaného vepisování přes klávesnici.

Zobrazení „relační struktury“ tedy bude provedeno pomocí orientovaného grafu $G = (V, E)$, kde V je množina vrcholů a $E \subseteq V \times V$ je množina hran. Dva vrcholy v_1 a v_2 grafu G, budou spojeny hranou právě tehdy, když $(v_1, v_2) \in R$, kde R je relace $R \subseteq V \times V$. Příklad zobrazení relace mezi dvěma vrcholy můžeme vidět na obrázku 3. Zobrazení „reflexivity“, tj. případu, kdy $(v_i, v_i) \in R$, provedeme orientovanou hranou, jejíž počátek a konec bude ve stejném vrcholu grafu G (viz obrázek 3).



Obrázek 3: Relace mezi dvěma prvky struktury a zobrazení reflexivity

Abychom byli schopni modelovat i speciální predikátové symboly uspořádání jako $<$ nebo $\leq$, umístíme jednotlivé vrcholy orientovaného grafu do kartézského systému souřadnic. Uspořádání vrcholů grafu budeme, podobně jako v Hasseových diagramech, brát dle jejich pozice na ose y. Například pro predikátový symbol $<$ pak platí, že $v_1 < v_2$ právě tehdy, když pozice vrcholu v_1 na ose y je menší než pozice vrcholu v_2. Kompletní „relační strukturu“, splňující například axiomy reflexivity a transitivity můžeme vidět na obrázku 4.



Obrázek 4: Příklad kompletní relační struktury

3.1 Funkční požadavky

Funkční požadavky určují, co by měl systém z pohledu uživatele dělat a mohou také definovat chování za určitých situací. Spíše než o popis celého systému jako celku, se jedná o popis jeho konkrétních částí na vyšší úrovni, který by měl čtenáři dopomoci k hlubšímu pochopení vyvíjené aplikace. Všechny funkční požadavky jsou strukturovány do číselného seznamu, abychom se na ně mohli později v textu odkazovat.

1. **Obecné požadavky**

   (a) **Převedení formule z textového řetězce do stromové struktury**
   Vstup: Textový řetězec
   Výstup: Stromová struktura formule

   Systém musí akceptovat pouze správně utvořené formule PL1 (viz definice 2.7), vyhovující daným omezením (viz kapitola 4). V případě nekorektního vstupu, systém uživatele upozorní zprávou „Neočekávaný znak na pozici x“.

   Předpokládá se, že uživatel bude formule do systému vkládat pomocí klávesnice, či podobného vstupního zařízení. Některé speciální symboly jazyka PL1 jako všeobecný kvantifikátor $\forall$ nebo implikace $\supset$ se běžně na těchto vstupních zařízeních nevyskytují nebo bývá přinejmenším obtížné je pomocí těchto zařízení zapsat. Proto v zájmu uživatelského komfortu provedeme nahrazení určitých speciálních symbolů jazyka PL1 symboly, které lze pomocí vstupních zařízení lépe zapisovat. Seznam všech nahrazených speciálních symbolů jazyka PL1 můžeme vidět v tabulce 1.

   Systém by měl dále akceptovat formule jazyka PL1 zapsané ve formátu TEX. Budeme vyžadovat aby formule v tomto formátu byla ohraničena znakem $. Příkladem budiž formule $\forall x(P(x) \supset \neg Q(x)$, kterou lze ve formátu TEXzapsat jako „*$\forall x(P(x) \supset \neg Q(x))$*“. Seznam speciálních symbolů ve formátu TEXnalezneme opět v tabulce 1.

	Symbol PL1	Vnitřní symbol	Symbol TEX
konjunkce	$\wedge$	*	\wedge
disjunkce	$\vee$	+	\vee
implikace	$\supset$	$>$	\supset
ekvivalence	$\equiv$	$\sim$	\equiv
negace	$\neg$	!	\neg
všeob. kvantifkátor	$\forall$	V	\forall
exist. kvantifikátor	$\exists$	1	\exists
rovnost	$=$	$=$	$=$

Tabulka 1: Nahrazení symbolů PL1 vnitřními symboly

(b) **Vložení formule do množiny**
Vstup: Formule PL1
Výstup: Úspěch či neúspěch

Pro úspěšné vložení formule do množiny musí být splněny tyto předpoklady:

- „Semi-ekvivalentní“ (viz definice 5.1) formule nesmí být prvkem cílové množiny formulí. Tato podmínka nijak danou množinu neomezuje ve smyslu nesplnitelnosti, nicméně může uživateli zabránit v překlepech, kdy cílová množina například obsahuje formuli $\forall x P(x)$, uživatel opět vloží formuli $\forall x P(x)$, přitom měl však na mysli vložit formuli $\forall x R(x)$
- Musí být zachována konzistence arity predikátů. Například, pokud některá z formulí cílové množiny obsahuje predikátový symbol P s aritou $n = 2$ a formule na vstupu obsahuje predikátový symbol se stejným názvem, ale s aritou $n = 1$, bude uživatel systémem upozorněn, aby změnil aritu nebo název predikátu ve vkládané formuli.

Jestliže vkládaná formule obsahuje nové predikátové či funkční symboly, provedeme zobrazení nabídky jejich interpretace (viz bod 2.h).

(c) **Odebrání formule z množiny**
Vstup: Formule PL1

Po odebrání formule z množiny systém provede kontrolu, zda se predikátové a funkční symboly této formule nacházejí i ve zbývajících formulích množiny. Pokud ne, systém tyto symboly odstraní i z nabídky interpretací.

(d) **Import více formulí**
Vstup: Textový řetězec
Výstup: Množina formulí ve stromové struktuře

Na vstupu očekáváme neprázdný textový řetězec. Jednotlivé formule budou odděleny novým řádkem. Převedení jednotlivých formulí z textového řetězce do stromové struktury bude probíhat dle 1.a. Vložení formule do množiny bude probíhat dle bodu 1.b.

(e) **Zobrazení množiny formulí**
Vstup: Stromová struktura formule
Výstup: Zobrazená množina formulí

Systém, pro účely zobrazení formule, musí zajistit zpětné převedení vnitřních symbolů (viz tabulka 1) na standardní symboly jazyka PL1. Uživatel tak nebude muset dále přemýšlet nad významem vnitřních symbolů a bude se tak moci lépe soustředit na řešený problém.

(f) **Vyhodnocení pravdivosti formule v IS pro valuaci e**
Vstup: Interpretovaná formule PL1, valuace e
Výstup: Ano, v případě, že je formule v IS pro valuaci e pravdivá. Jinak ne.

Vyhodnocení bude probíhat přesně dle definice 2.11.

(g) **Vyhodnocení pravdivosti formule v interpretaci**
Vstup: Interpretovaná formule PL1
Výstup: Ano, právě tehdy, když je formule v IS pravdivá. Jinak ne.

Systém, před vyhodnocením pravdivosti formule v interpretaci, musí ověřit, zda je interpretační struktura dané formule kompletní.

2. **Vytváření IS množiny formulí**

(a) **Zobrazení universa ve 3D**
Universum bude složeno z individuí, které budou mít tyto vlastnosti:

- pozice v 3D prostoru - (x, y, z)
- velikost - *malá*, *střední*, *velká*
- barva - *modrá*, *zelená*, *červená*, *žlutá*, *růžová*, *oranžová*
- tvar - *pravid. čtyřboký jehlan*, *kužel*, *koule*, *rotační válec*, *krychle*, *pravid. čtyřboký hranol*, *čtyřstěn*, *osmistěn*, *dvanáctistěn*

Systém musí být schopen přehledně zobrazit všechny vlastnosti individuí v trojrozměrném prostoru s možností přiblížení, nebo oddálení vyobrazené scény. Systém by měl dále umožnit změnu úhlu pohledu „kamery“, pro případ překrytí individuí.

(b) **Přidání individua do universa**
Systém musí při přidávání individua do universa zajistit i jeho přidání do definičního oboru funkčních symbolů s aritou $n = 1$ (viz bod 2.f).

(c) **Úprava vlastnosti individua**
Systém musí uživateli umožnit úpravu všech vlastností individua. Tato změna by se měla ihned projevit na vyobrazeném universu.

(d) **Odebrání individua z universa**
Vstup: Individuum

Systém při odebírání individua z universa musí zajistit i jeho odstranění ze všech interpretací funkčních symbolů. Před samotným odebráním však systém uživatele upozorní, že případně dojde i k odebrání z interpretací funkčních symbolů.

(e) **Interpretace predikátových symbolů**
Vstup: Predikátový symbol
Výstup: Interpretace predikátového symbolu (unární nebo binární relace)

Predikátové symboly bude uživatel interpretovat výběrem jedné z předem připravených unárních, potažmo binárních relací.

Unární relace:
<barva> = jedna z 6 barev, <tvar> = jeden z 8 tvarů

- x má barvu <barva>

- x je <tvar>
- x je mnohostěn
- x má malou velikost
- x má střední velikost
- x má velkou velikost
- x má kruhovou podstavu
- x má čtvercovou podstavu

Binární relace:

- x je výše nebo stejně vysoko jako y
- x je níže nebo stejně nízko jako y
- x je nalevo od y
- x je napravo od y
- x je nad nebo pod y
- x je vedle y
- x je blíže nebo stejně blízko jako y
- x je dále nebo stejně daleko jako y
- x je větší nebo stejně velké jako y
- x je menší nebo stejně velké jako y
- x je stejně velké jako y
- x je stejně vysoko jako y
- x je větší než y
- x je menší než y
- x má stejnou barvu jako y
- x má stejný tvar jako y

(f) **Interpretace funkcí s aritou** $n = 1$
Vstup: Funkční symbol f
Výstup: Definice zobrazení f pomocí tabulky

Systém umožní složení funkčního zobrazení výčtem hodnot, neboli tabulkou. Příklad definice jednoduché unární funkce představující zobrazení $f : U \to U$ pro universum $U = \{i_1, i_2, i_3, i_4\}$ můžeme vidět v tabulce 2. Pro zvýšení uživatelského komfortu systém automaticky „předvyplní“ vzory (levý sloupec tabulky 2) všemi individui universa. Uživatel k těmto vzorům pouze doplní jejich obrazy. Systém dále musí zajistit, aby funkční hodnota $f(x)$ byla definována pro celé universum (tzv. *totalita funkce*). Jinými slovy, musí být zajištěno, aby pro každý vzor byl definován i obraz.

(g) **Interpretace funkcí s aritou** $n = 0$ **(konstanty)**
Vstup: Konstanta a
Výstup: Interpretace konstanty - individuum

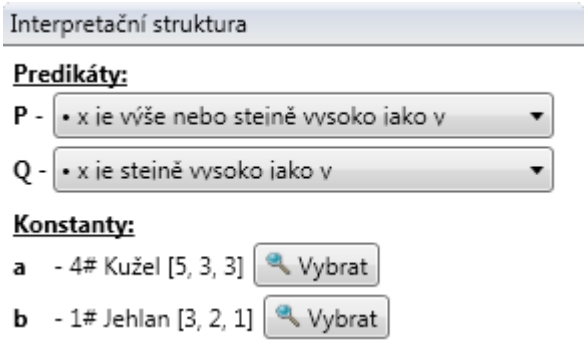
U	$\rightarrow$	U
i_1	$\rightarrow$	i_2
i_2	$\rightarrow$	i_3
i_3	$\rightarrow$	i_2
i_4	$\rightarrow$	i_1

Tabulka 2: Příklad definice interpretace funkčního symbolu

Interpretace konstant bude umožněna výběrem jednoho konkrétního individua z universa. U takto vybraného individua poté systém, ve trojrozměrném zobrazení universa, vykreslí příslušný název konstanty.

(h) **Zobrazení nabídky interpretací symbolů**
Vstup: Množina formulí PL1

Systém musí uživateli zobrazit seznam všech symbolů, z množiny formulí, které podléhají interpretaci. Přitom musí zajistit, aby „symboly stejného typu“ byly zobrazeny jen jednou. Řekněme, že množina formulí obsahuje formule $\forall x \forall y(Q(x, y) \supset P(x, y))$ a $P(a, b)$. Uživateli bude v tomto případě nabídnuta možnost pro výběr interpretace predikátových symbolů P a Q, i konstant a, b. Příklad zobrazení interpretace symbolů můžeme vidět na obrázku 5.



Obrázek 5: Zobrazení interpretační struktury

(i) **Ověření zda je interpretační struktura modelem množiny formulí**
Vstup: Množina formulí PL1, interpretační struktura
Výstup: Ano, právě tehdy, když je IS modelem množiny formulí. Jinak ne.

Systém před samotným ověřením provede kontrolu, zda je daná IS kompletní. Tj. jsou interpretovány všechny predikátové a funkční symboly. U funkčních symbolů s aritou $n = 1$ musí být také ověřena totalita funkce. Pokud je IS korektně sestavena, systém provede vyhodnocení pravdivostí formulí v interpretaci a jeho výsledek oznámí uživateli.

(j) **Uložení IS a množiny formulí do formátu XML**

Vstup: Universum, množina formulí PL1, interpretační struktura
Výstup: Soubor ve formátu XML

(k) **Načtení IS a množiny formulí z formátu XML**
Vstup: Soubor ve formátu XML
Výstup: Universum, množina formulí PL1, interpretační struktura

3. **Nápověda pro vytvoření modelu množiny IS**

(a) **Vizualizace pravdivostních hodnot formulí pro jednotlivé valuace**
Vstup: Množina formulí, seznam pravd. hodnot formulí pro jednotlivé valuace

Systém umožní uživateli pro vybranou formuli zobrazit detaily o jejím vyhodnocení pravdivosti v interpretaci. Vizualizace bude provedena vykreslením interaktivního grafu v jedno a dvou rozměrném kartézském systému souřadnic, kde hodnoty souřadnic os představují jednotlivé valuace proměnných x_i (viz obrázek 2). Z praktických důvodů se omezíme pouze na dvě proměnné, nicméně uvedené postupy se dají aplikovat i na tři proměnné vykreslením trojrozměrného grafu. Systém umožní uživateli z grafu výběr konkrétní valuace, pro kterou zobrazí dodatečné informace o vyhodnocení pravdivosti formule při této valuaci:

- všechny vlastnosti individuí x_i, případně y_j
- výsledky funkčních zobrazení. Například pro formuli $\forall x P(f(x))$ toto bude výsledek zobrazení f pro konkrétní valuaci $e(x_i)$
- pravdivostní hodnoty jednotlivých predikátových symbolů formule
- nápovědu pro binární logické spojky

Konkrétní podobu jednotlivých detailů nápovědy si blíže popíšeme v kapitole 5.9. U formulí s kvantifikátory, systém zobrazí i následující pomocné věty:

- $\forall x A$ - formule musí být splněna pro všechna ohodnocení $e(x_i)$
- $\exists x A$ - formule musí být splněna alespoň pro jedno ohodnocení $e(x_i)$
- $\forall x \forall y A$ - formule musí být splněna pro všechna ohodnocení $e(x_i)$, $e(y_j)$
- $\forall x \exists y A$ - pro každé ohodnocení $e(x_i)$ musí existovat alespoň jedno ohodnocení $e(y_j)$, ve kterém je formule splněna
- $\exists x \forall y A$ - musí existovat alespoň jedno ohodnocení $e(x_i)$, ve kterém je formule splněna pro všechna ohodnocení $e(y_j)$
- $\exists x \exists y A$ - formule musí být splněna alespoň pro jedno ohodnocení $e(x_i)$, $e(y_j)$

(b) **Zobrazení pravdivostí jednotlivých formulí v IS**
Vstup: Množina formulí, interpretační struktura
Výstup: Seznam pravdivosti jednotlivých formulí v IS

Systém pro účely nápovědy zajistí zobrazení i pravdivostních hodnot jednotlivých formulí, namísto celé množiny.

4. **Modelování vlastností relací**

   (a) **Zobrazení relační struktury pomocí orientovaného grafu**
   Vstup: Orientovaný graf G

   Zobrazení orientovaného grafu by mělo vypadat podobně jako na obrázcích 3 a 4. Jednotlivé vrcholy grafu musí být přehledně umístěny do kartézského systému souřadnic s osami x a y, probíhající množinou přirozených čísel $\mathbb{N}$.

   (b) **Přidání uzlu do grafu**
   Vstup: Bod kartézského systému souřadnic $B = (x, y)$
   Výstup: Úspěch / neúspěch

   Přidání nového vrcholu do grafu uživatel provede kliknutím myší na určitou pozici v kartézském souřadnicovém systému. Systém před přidáním vrcholu do grafu a jeho vykreslením, zaokrouhlí jednotlivé složky bodu B na nejbližší celé číslo. Tímto zajistíme přehlednější vykreslení celého grafu. Systém dále musí zajistit, aby se na pozici B nacházel maximálně jeden vrchol grafu. Pro zamezení konfliktů s vytváření hran grafu, systém přidá nový uzel do grafu pouze tehdy, pokud je stisklá klávesa „Ctrl“.

   (c) **Označení uzlu z grafu**

   Označení uzlu provede uživatel pouhým kliknutím na daný uzel.

   (d) **Odebrání uzlu z grafu**
   Vstup: Vrchol v_i grafu G

   Systém umožní uživateli odebrat pouze označený vrchol grafu, stisknutím klávesy „Delete“. Systém dále musí zajistit odebrání veškerých hran vedoucích z a do vrcholu v_i.

   (e) **Přidání hrany do grafu**
   Vstup: Vrcholy v_i a v_j grafu G

   Systém musí zajistit, aby uživatel nemohl vložit více než jednu hranu mezi dvojicí vrcholů (v_i, v_j). Jestliže $v_i = v_j$, pak systém vytvoří speciální hranu pro reflexivitu (viz obrázek 3).
   Na uživatelské úrovni by přidání hrany do grafu mělo fungovat, kliknutím na výchozí uzel, při stisknuté klávese „Alt“ a následným kliknutím na uzel cílový.

   (f) **Označení hrany grafu**
   Označení hrany provede uživatel stejným způsobem, jako označení uzlu - tedy kliknutím na požadovanou hranu.

   (g) **Odebrání hrany z grafu**
   Vstup: Hrana e_i grafu G

   Systém umožní uživateli odebrat pouze označenou hranu grafu, stisknutím klávesy „Delete“.

(h) **Přesun uzlu grafu na jinou pozici**
Vstup: Vrchol v_i grafu G, nová pozice $B = (x, y)$
Výstup: Úspěch / neúspěch

Proces přesunu končí neúspěchem, pouze v případě, když na pozici B již existuje jiný vrchol v_j. Jinak musí systém zajistit překreslení vrcholu v_i, všech hran vedoucích z a do v_i na novou pozici.

(i) **Interpretace predikátových symbolů**
Vstup: Predikátový symbol
Výstup: Interpretace predikátového symbolu (relace)

V jazyce axiómů speciálních teorií jsou definovány tzv. „speciální predikátové symboly“ (viz kapitola 2.7). Například speciální symbol $<$ ve formuli $\forall x \forall y(x < y)$, stejně jako ostatní predikátové symboly podléhá interpretaci. Standardně je však interpretován jako relace „ostře menší“. Systém proto využije poněkud obecnější řešení a formule se speciálními predikátovými symboly bude uživateli prezentovat ve standardním jazyce formulí PL1, tak jak je definován v 2.7. Formuli $\forall x \forall y(x < y)$ tedy systém zobrazí jako $\forall x \forall y R(x, y)$, kde predikátový symbol R bude interpretován jednou z těchto binárních relací:

- *x je v relaci s y* (tzn. existuje orientovaná hrana grafu G z x do y)
- $<$
- $\leq$
- $>$
- $\geq$

Výjimkou bude pouze speciální symbol pro „rovnost“ =. Tento symbol bude z pohledu uživatele vždy interpretován jako rovnost. Jinak systém nabídne uživateli interpretaci predikátových symbolů stejným způsobem jako na obrázku 5.

(j) **Ověření zda relační struktura splňuje požadované vlastnosti**
Vstup: Orientovaný graf G, Množina formulí PL1, interpretační struktura
Výstup: Ano, právě tehdy, když je IS modelem množiny formulí. Jinak ne.

Před ověřením požadovaných vlastností systém provede kontrolu, zda je množina vrcholů V orientovaného grafu G neprázdnou množnou - $V \neq \emptyset$.

(k) **Uložení IS, grafu a množiny formulí do formátu XML**
Vstup: Orientovaný graf G, množina formulí PL1, interpretační struktura
Výstup: Soubor ve formátu XML

(l) **Načtení IS, grafu a množiny formulí z formátu XML**
Vstup: Soubor ve formátu XML
Výstup: Orientovaný graf G, množina formulí PL1, interpretační struktura

(m) **Vložení známé vlastnosti relací do množiny formulí**
Vstup: Seznam známých vlastností relací
Výstup: Úspěch / neúspěch

Systém musí uživateli umožnit vložení vybrané známé vlastnosti relací do množiny formulí. Vkládání „vlastnosti“ do množiny formulí se bude řídit stejnými pravidly jako standardní vložení formule (viz bod 1.a). Seznam známých vlastností by měl být uložen ve formátu XML, pro možnost doplnění dalších vlastností.

(n) **Vložení speciálních axiomů teorie do množiny formulí**
Vstup: Seznam známých teorií a jejich speciálních axiómů

Systém musí uživateli umožnit vložení speciálních axiómů vybrané teorie do množiny formulí. Vkládání jednotlivých axiómů se bude řídit stejnými pravidly jako standardní vložení formule (viz bod 1.a). Před vložením axiómů systém provede odstranění všech formulí z množiny formulí. Seznam známých teorií by měl být uložen ve formátu XML, pro možnost doplnění dalších teorií.

3.2 Případy použití

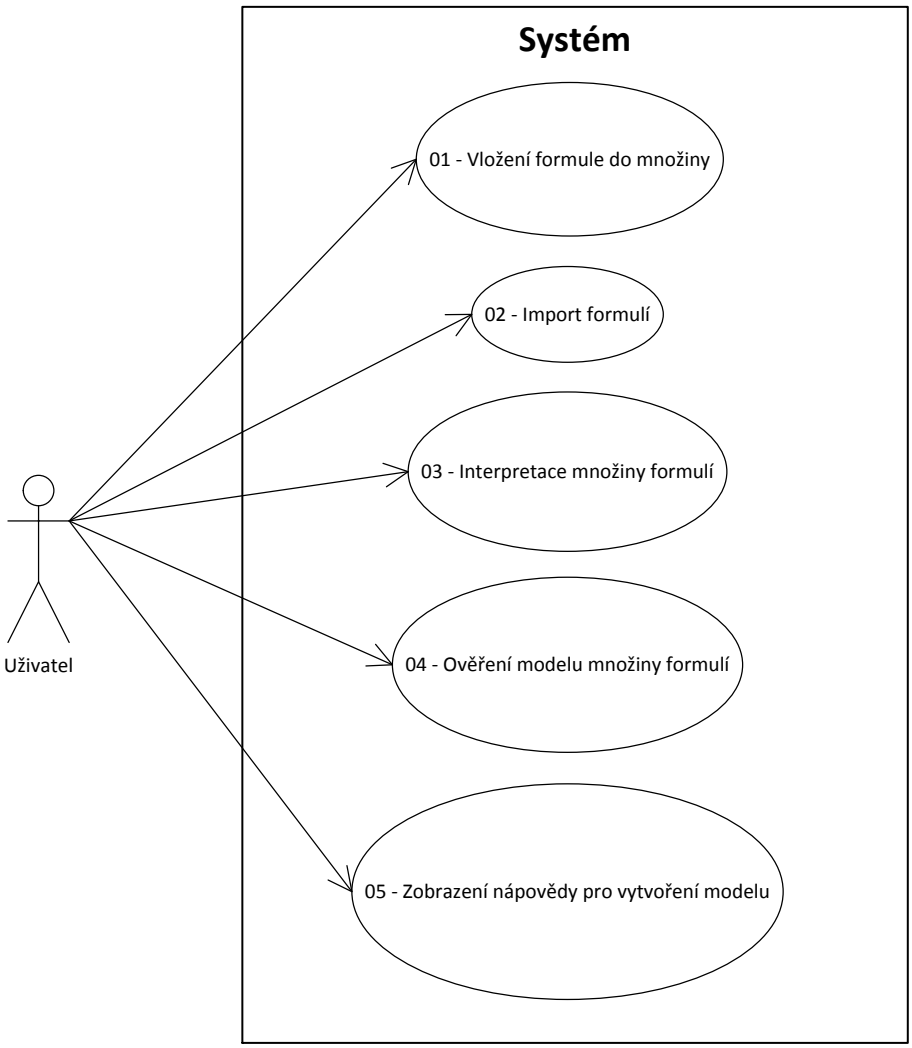
Případy užití specifikují vzory chování realizovaných softwarovým systémem. Každý případ užití lze chápat jako posloupnost vzájemně navazujících kroků popisující interakci mezi aktérem a systémem [6]. *Aktérem* popisujeme uživatele vně systému, který je s ním v interakci. Může se jednat o fyzické osoby, jiné systémy nebo i hardwarové zařízení [7]. Nás vyvíjený systém bude v základu obsahovat pouze jediného aktéra - *uživatele*. Vzhledem k povaze celého vyvíjeného systému se v této podkapitole zaměříme pouze na pět základních případů užití. Souhrn těchto případů užití můžeme vidět na UML diagramu 6. Zbytek požadovaných funkcí aplikace by již měl být dostatečně specifikován dílčími funkčními požadavky.

Jelikož způsob specifikace případů užití není žádným způsobem standardizován, využijeme pro naše potřeby modifikovanou definici z [7] a [8]. Specifikace jednotlivých případů použití se budou skládat z těchto bodů:

- **Identifikace případu** - jednoznačné hierarchické označení případu užití. Například 1.0, 1.1, . . .
- **Název případu užití** - srozumitelný název případu užití
- **Cíl případu užití** - stručný popis cíle, který má být případem užití splněn
- **Vstupní podmínky** - podmínky, které musí být plněny, pokud má případ užití začít
- **Výstupní podmínky** - podmínky, které definují stav systému po dokončení případu užití

- **Scénář případu užití** - jednotlivé kroky k dosažení cíle případu užití

Poznámka 3.1 Všechny specifikace jednotlivých případů užití nalezneme v příloze A.



Obrázek 6: Diagram základních případů užití

4 Vymezení podmnožiny PL1

V definici 2.7 jsme vymezili jazyk formulí PL1. Pouze formule vyhovující této definici jsou dobře utvořenými formulemi jazyka PL1. Takto „dobře utvořených“ formulí je však nekonečně mnoho a proto je z praktického hlediska vhodné vymezit určitou podmnožinu těchto formulí, se kterou bude výsledný systém pracovat.

Ze specifikace požadavků vyplývá, že vyvíjený systém by měl obsahovat dvě prostředí. Zaprvé, prostředí pro vytváření interpretačních struktur množiny formulí PL1 a zadruhé prostředí pro modelování důležitých vlastností relací. Abychom mohli v systému modelovat jednu ze základních vlastností relací tzv. anti-symetrii, musíme do jazyka PL1 zavést binární predikátový symbol pro rovnost $=$. Dále v prostředí pro vytváření interpretačních struktur je, z didaktického hlediska, vhodné ponechat funkční symboly. To však způsobí poněkud zbytečné komplikace při návrhu prostředí pro modelování vlastností relací. Proto bylo rozhodnuto, že každé z prostředí bude mít svou vlastní vymezenou podmnožinu formulí PL1.

4.1 Vymezení podmnožiny pro interpretační struktury

Neformálně můžeme kladená omezení na formule v prostředí pro vytváření interpretačních struktur shrnout do těchto bodů:

- všechny kvantifikátory mohou být pouze v hlavě (prefixu) formule
- maximální počet binárních logických spojek je 1
- maximální počet proměnných ve formulích je 2 (povoleny jsou i volné proměnné)
- povoleny jsou pouze predikáty o aritě $n = 1$ nebo $n = 2$
- argumentem predikátu může být pouze proměnná, konstanta nebo unární funkce
- povoleny jsou pouze unární funkce, jejímiž argumenty mohou být pouze proměnné nebo konstanty. Počet funkcí ve formuli není nijak omezen
- povolené názvy predikátových symbolů jsou P, Q, R nebo S
- povolené názvy funkcí jsou f nebo g
- povolené názvy konstant jsou a, b, c, d, e, $a1$, $b1$ nebo $c1$
- povolené názvy proměnných jsou x nebo y

Takto zadaným omezením poté odpovídá následující bezkontextová gramatika (viz definice 2.6)[1]:

$$
\begin{aligned}
St &\rightarrow QntFo \mid Fo \\
Qnt &\rightarrow Qntvar \mid QntvarQntvar \\
Qntvar &\rightarrow QuantifVariable \\
Fo &\rightarrow\ !Fconn \mid Fconn \mid\ !A \mid A \\
Fconn &\rightarrow (ABinaryconnA) \\
A &\rightarrow Pred(Arg) \mid Pred(Arg, Arg) \\
Arg &\rightarrow Varconst \mid Func(Varconst) \\
Varconst &\rightarrow Variable \mid Constant \\
Quantif &\rightarrow V \mid 1 \\
Binaryconn &\rightarrow * \mid\ + \mid\ > \mid\ \sim \\
Pred &\rightarrow P \mid Q \mid R \mid S \\
Func &\rightarrow f \mid g \\
Variable &\rightarrow x \mid y \\
Constant &\rightarrow a \mid b \mid c \mid d \mid e \mid a1 \mid b1 \mid c1
\end{aligned}
$$

4.2 Vymezení podmnožiny pro modelování vlastností relací

Podobně jako v předchozí podkapitole si nyní neformálně shrneme omezení kladená na formule v prostředí pro modelování vlastností relací:

- všechny kvantifikátory mohou být pouze v hlavě (prefixu) formule
- maximální počet binárních logických spojek je 3
- maximální počet proměnných ve formulích jsou 3 (povoleny jsou i volné proměnné)
- povoleny jsou pouze predikáty o aritě $n = 2$
- argumentem predikátu může být pouze proměnná
- funkční symboly nejsou povoleny
- povolené názvy predikátových symbolů jsou P, Q, R nebo S. Dále ještě speciální predikátový symbol pro rovnost $=$
- povolené názvy proměnných jsou x, y nebo z

[1] V gramatice jsou již některé speciální znaky jazyka PL1 nahrazeny vnitřními (viz 3.1)

Zadaným omezením opět odpovídá následující bezkontextová gramatika:

$$
\begin{aligned}
St &\rightarrow QntFo \mid Fo \\
Qnt &\rightarrow Qntvar \mid QntvarQntvar \mid QntvarQntvarQntvar \\
Qntvar &\rightarrow QuantifVariable \\
Fo &\rightarrow\ !Fconn \mid Fconn \\
Fconn &\rightarrow (AnBinaryconnAn) \mid An \\
An &\rightarrow\ !A \mid A \\
A &\rightarrow (A2BinaryconnA2) \mid A2 \\
A2 &\rightarrow\ \ Pred(Variable, Variable) \mid Pred(Variable, Variable) \\
Quantif &\rightarrow V \mid 1 \\
Binaryconn &\rightarrow * \mid + \mid > \mid \sim \\
Pred &\rightarrow P \mid Q \mid R \mid S \mid = \\
Variable &\rightarrow x \mid y \mid z
\end{aligned}
$$

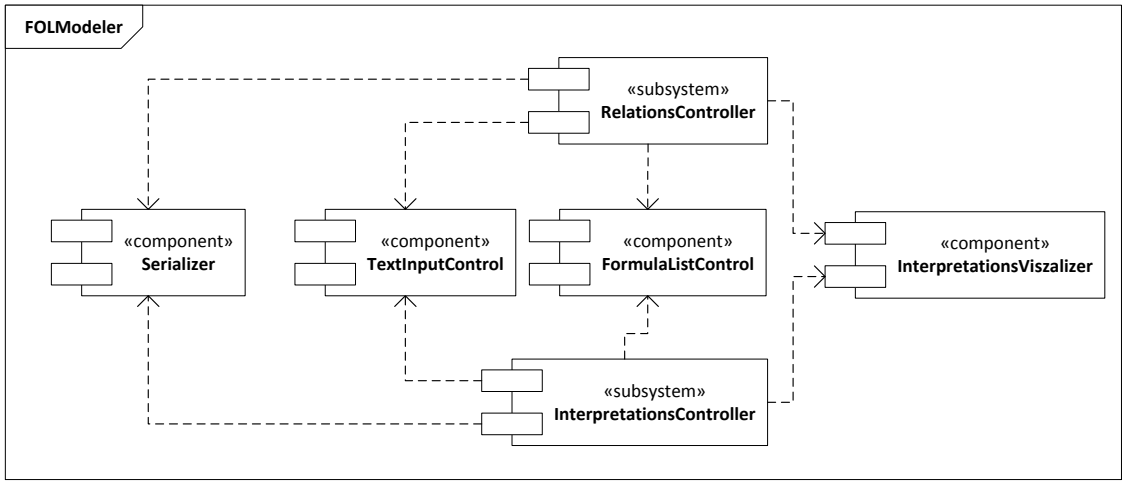
Bezkontextové gramatiky upravené pro implementaci zpracování uživatelského vstupu nalezneme na přiloženém CD v souborech *FOL.g*, respektive *FOLRelational.g*.

5 Analýza a návrh

V této kapitole se budeme zabývat analýzou požadavků a objektovým návrhem jednotlivých částí systému. Podrobně si rozebereme dílčí části aplikace, popíšeme si fungování důležitých algoritmů a také komunikaci mezi jednotlivými komponentami systému. Stejně jako v předchozí kapitole Specifikace požadavků nejprve začneme s obecnějšími funkcemi a poté plynule navážeme na konkrétnější požadavky z obou prostředí. Analýzu a návrh budeme provádět s pomocí sekvenčních, třídních a komponentních UML diagramů, dle anotace převzané z [9], [10] a [11]. Výsledkem této kapitoly pak bude kompletní konceptuální model systému.

5.1 Architektura systému

Ještě než se pustíme do analýzy jednotlivých požadavků, si krátce popíšeme u architekturu celého systému. Díky tomuto získáme daleko lepší nadhled nad jednotlivými funkcemi a jejich rolemi v celém systému. Z požadavků vyplývá, že systému musí obsahovat dvě rozdílná prostředí - pro vytváření interpretační struktury formulí PL1 a prostředí pro modelování důležitých vlastností relací. Dále systém musí umožnit uživateli vkládání formulí pomocí klávesnice, zobrazení množiny formulí a uložení celé práce do formátu XML. Například požadavek na vkládání formulí pomocí klávesnice je společný pro obě zmíněná prostředí a je tedy zbytečné mít tuto funkcionalitu v systému dvakrát. Všechna tato fakta byla tedy při analýze architektury systému brána v potaz a výsledkem je schéma architektury z obrázku 7.



Obrázek 7: Architektura systému

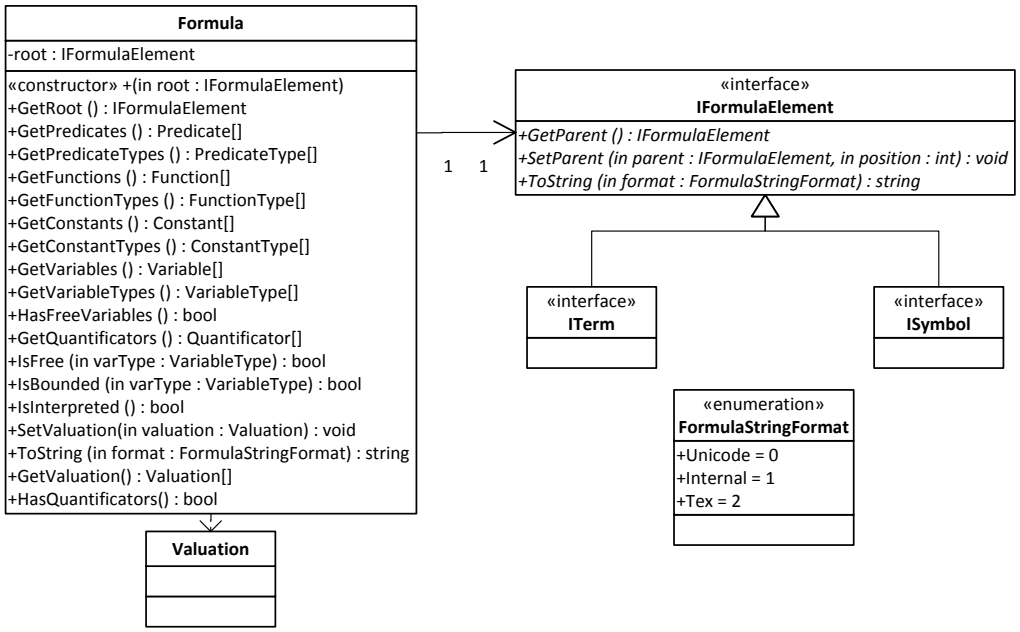
- **RelationsController** - subsystém prostředí pro modelování vlastností relací
- **InterpretationsController** - subsystém prostředí pro vytváření interpretační struktury množiny formulí

- **Serializer** - komponenta pro ukládání a načítání práce ze souboru XML
- **TextInputControl** - komponenta pro textový vstup
- **FormulaListControl** - komponenta pro zobrazení množiny formulí
- **InterpretationsVisualizer** - komponenta pro zobrazení interpretací jednotlivých symbolů

5.2 Statická struktura formule

Základním artefaktem, se kterým systém pracuje jsou formule PL1. Jako první si tedy rozebereme právě jejich statickou strukturu. Formule bude reprezentována stromovou strukturou, jejímž „kontejnerem" je třída *Formula* (obrázek 8). Tento kontejner bude sloužit zbytku systému pro práci s celou formulí. Například pro zjištění, zda je formule interpretována nebo zda obsahuje volné proměnné. Kompletní třídní diagram formule nalezneme v příloze (38 a 39).

Jednotlivé elementy formule jsou rozděleny do dvou skupin - *termy* (rozhraní *ITerm*) a *speciální symboly* (*ISymbol*)[2]. Každý symbol či term jazyka PL1 je reprezentován vlastní třídou - negace (*Negation*), predikátové symboly (*Predicate*), funkční symboly (*Function*) atd.

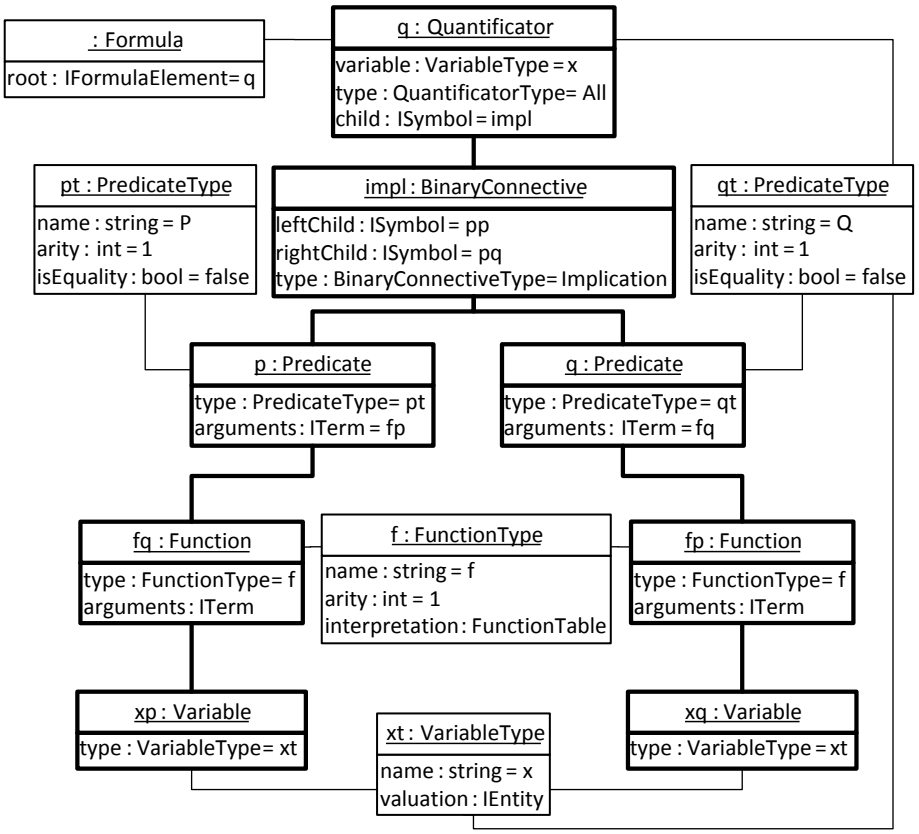


Obrázek 8: Základní třídní diagram formule

[2]Kvantifikátory nepatří do žádné z těchto skupin. Pouze implementují obecné rozhraní IFormulaElement

Mírnou komplikací je efektivní reprezentace symbolů, které se vyskytují na více místech formule. Vezměme například formuli $\forall x(P(f(x)) \supset Q(f(x)))$. Zde je funkce f argumentem predikátů P a Q. V objektové struktuře bude tato funkce reprezentována dvěmi různými instancemi třídy *Function*. Abychom zdůraznili fakt, že se ve skutečnosti jedná o stejnou funkci, obě instance třídy *Function* budou sdílet instanci třídy *FunctionType*, nesoucí informaci o názvu, aritě a interpretaci symbolu. Podobným způsobem jsou pak řešeny i predikátové symboly, konstanty a proměnné. Takto zvolená reprezentace nám pak výrazně zjednoduší nejrůznější operace prováděné nad formulí.

Jelikož objektová struktura pro konkrétní formuli nemusí být z třídních diagramů ihned patrná, je dobré třídní diagram doplnit ještě o tzv. *objektový diagram*. Příklad objektové struktury pro formuli $\forall x(P(f(x)) \supset Q(f(x)))$ pak můžeme vidět na obrázku 9.

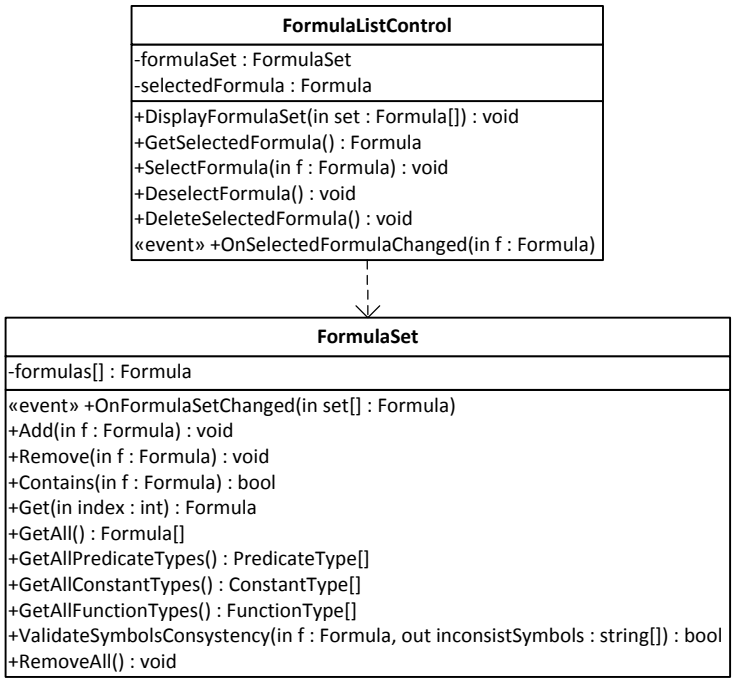


Obrázek 9: Objektový diagram formule $\forall x(P(f(x)) \supset Q(f(x)))$

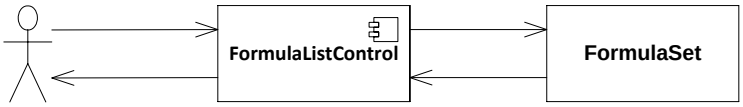
5.3 Množina formulí

Nyní tedy víme, jak vypadá statická struktura formule. Dalším naším krokem bude analýza množiny formulí. Z požadavků je jasné, že uživateli musí být umožněno provést

dvě základní operace - přidání a odebrání formule. Vzhledem k tomu, že s touto množinou bude pracovat interně i sám systém, je více než vhodné oddělit uživatelské rozhraní od samotné množiny. Podívejme se na třídní diagram na obrázku 10. Třída *FormulaListControl* zde zastupuje prezentační vrstvu. Třída *FormulaSet* pak představuje samotnou množinu formulí. Blokové schéma interakce uživatele s množinou formulí můžeme vidět na obrázku 11.



Obrázek 10: Třídní diagram množiny formulí



Obrázek 11: Blokové schéma interakce uživatele s množinou formulí

Přidání formule do množiny

Poněkud složitější operací je přidání nové formule do množiny. Již víme, že musí být splněny dva předpoklady - „semi-ekvivalentní“ formule nesmí být prvkem cílové množiny a musí být zachována konzistence arity predikátů [3]. Co však ještě nevíme, je význam pojmu „semi-ekvivalentní“ a způsob jak tyto dva předpoklady zajistit.

[3]Pokud bychom do vymezené podmnožiny zařadili i funkce s aritou $n > 1$, platila by pro ně stejná podmínka konzistence arity

Definice 5.1 *Dvě formule f_i a f_j jsou **semi-ekvivalentní** právě tehdy, když f_i.ToString(FormulaStringFormat.Internal) == f_j.ToString(FormulaStringFormat.Internal)*

Jinými slovy, dvě formule považujeme za semi-ekvivalentní, právě tehdy, když jsou shodné jejich textové reprezentace ve formátu se zástupnými symboly (viz 5.6). Smyslem tohoto prvního předpokladu, je snaha zamezit duplicitám v množině formulí. Tedy, aby uživatel nemohl například vložit dvakrát stejnou formuli $\forall x P(x)$. V ideálním případě by systém měl porovnat zda dvě formule f_i a f_j jsou „skutečně" ekvivalentní (viz definice 2.13). Nicméně z praktických důvodů si vystačíme i s touto výrazně slabší ekvivalencí.

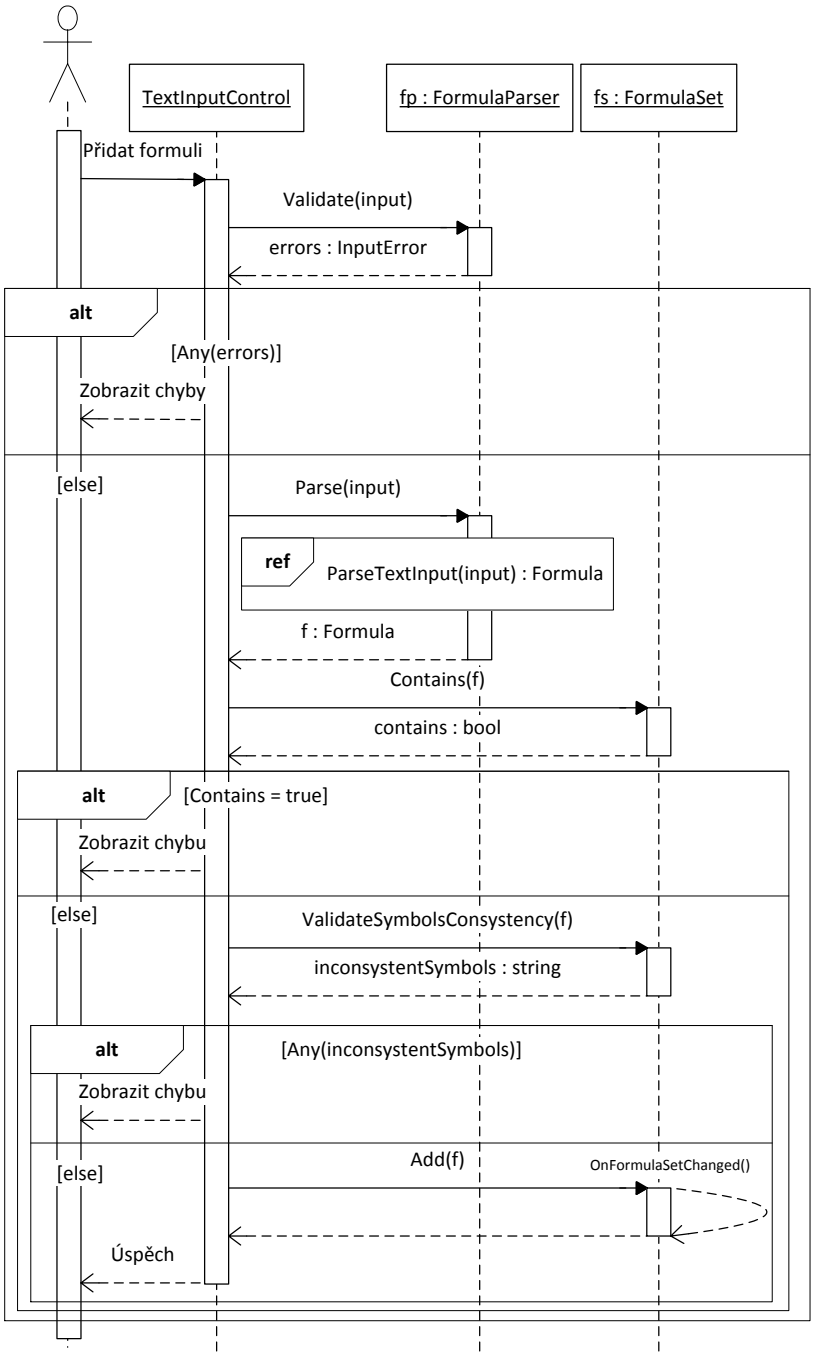
Druhý předpoklad nám zajišťuje to, aby se v množině nemohly vyskytovat predikátové symboly s rozdílnou aritou (například $M = \{P(a), P(a, b)\}$), což by způsobovalo zbytečné komplikace při vyhodnocování modelu množiny formulí(nutnost přejmenovat jeden z predikátů), ale hlavně nejednoznačnost zobrazené interpretace z pohledu uživatele. Příklad implementace obou předpokladů můžeme vidět na výpisech 1, respektive 2. Sekvenční diagram zachycující celý proces přidání formule do množiny, včetně interakce uživatele pak můžeme vidět na obrázku 12.

```
bool Contains(Formula newFormula)
{
    string newFormulaStr = newFormula.ToString(FormulaStringFormat.Inernal)

    foreach (Formula f in formulas)
    {
        if (f.ToString(FormulaStringFormat.Inernal) == newFormulaStr)
            return true;
    }

    return false;
}
```

Výpis 1: Zjištění zda množina obsahuje formuli

```
bool ValidateSymbolsConsystency(Formula f, out List<string> inconsystentSymbols)
{
    inconsystentSymbols = new List<string>();
    Dictionary<string, int> existingTypes = (from f in formulas
                                             from pt in f.GetAllPredicateTypes()
                                             select new{Name=pt.Name, Arity=pt.Arity}
                                             ).ToDictionary(k => k.Name, v => v.Arity);
    bool isValid = true;
    foreach (PredicateType newType in f.GetAllPredicateTypes())
    {
        string ntn = newType.Name;
        if (existingTypes.ContainsKey(ntn) && existingTypes[ntn].Arity != newType.Arity) {
         isValid = false;
         inconsystentSymbols.Add(ntn);
        }
    }
     return isValid;
}
```

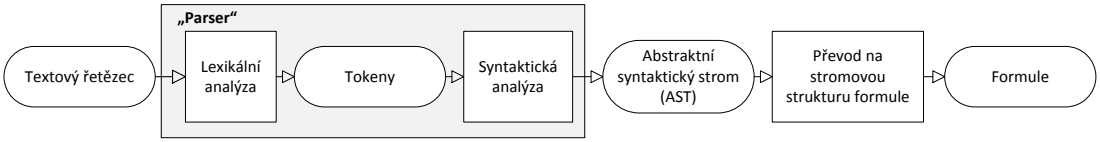
Výpis 2: Zajištění konzistence arity



Obrázek 12: Přidání formule do množiny formulí

5.4 Zpracování uživatelského vstupu z textového řetězce

V této fázi již máme představu, jakým způsobem bude systém reprezentovat jednotlivé formule a množinu formulí. Dalším logickým krokem je tedy analýza zpracování uživatelského vstupu z textového řetězce. V předchozích kapitolách jsme si řekli, že uživatel bude formule do systému zadávat ve formátu se zástupnými symboly. Například formule $\forall x(P(x) \supset Q(x))$ je zapsána jako $Vx(P(x) > Q(x))$. Jak můžeme vidět na blokovém schématu (obr. 13), zpracování takovéhoto vstupu probíhá v několika krocích, které si nyní popíšeme.



Obrázek 13: Blokové schéma převodu textového řetězce na stromovou strukturu

Prvním krokem je aplikování „parseru“ na textový řetězec. **Parser** si můžeme představit jako komponentu, jejímž vstupem je textový řetězec a výstupem je datová struktura - typicky tzv. **abstraktní syntaktický strom** (dále jen AST). Celý proces „parsování“ bývá rozdělen na dva kroky. Tím prvním je *lexikální analýza* a druhým *analýza syntaktická*. Vstupem do lexikální analýzy je textový řetězec, který je během této fáze rozdělen na tzv. *tokeny*. Token, představuje „kategorizovanou“ dále nedělitelnou část řetězce. Pro jazyk formulí PL1 jsou to tedy jednotlivé logické, speciální a funkční symboly (viz definice 2.7). Příklad několika tokenů pro jazyk PL1 můžeme vidět v tabulce 10. Vstupem do syntaktické analýzy je seznam tokenů, výstupem je AST. Účelem syntaktické analýzy je kontrola, zda daná posloupnost tokenů je korektní. Pro definici „korektní posloupnosti tokenů“ se typicky využívají bezkontextové gramatiky, my tedy využijeme gramatiky z kapitoly 4.

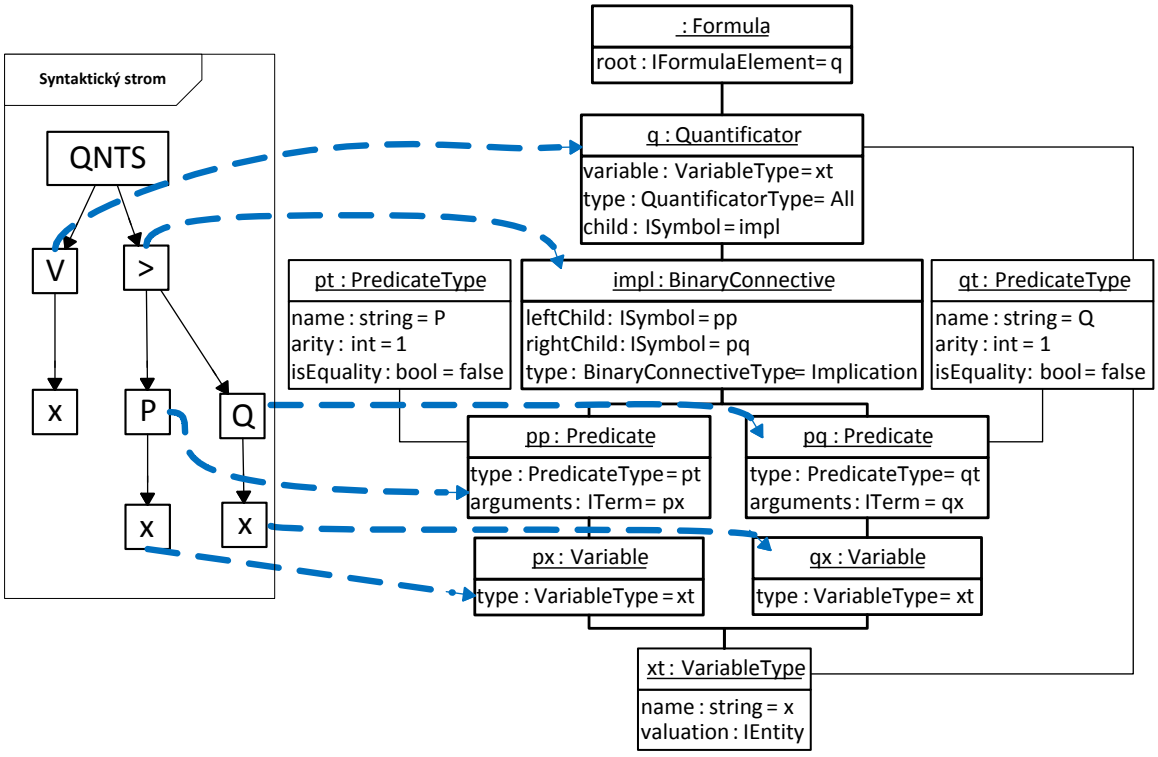
Token	Kategorie
V	QUANTIFICATOR
!	NEGATION
>	BINARY_CONNECTIVE
P	PREDICATE
(	BRACKET

Tabulka 3: Příklad „tokenů“ pro jazyk PL1

Jakmile máme k dispozici abstraktní syntaktický strom, tak naším posledním úkolem je převést tento strom do naší stromové struktury formule. Samotný převod je v zásadě velmi jednoduchý. Syntaktický strom postupně procházíme metodou prohledávání do hloubky a pro každý uzel syntaktického stromu, představující určitý token, vytváříme odpovídající instanci třídy statické struktury formule. Celý princip tohoto převodu je znázorněn na obrázku 14.

Možností, jak zkonstruovat parser je nepřeberné množství. Buď můžeme jít na celou věc tzv. „od lesa“ a zkonstruovat celý parser vlastními silami, nebo můžeme použít jeden z nástrojů, kterému předložíme bezkontextovou gramatiku a tento nástroj pak celý parser automaticky vytvoří. Pro naše účely se jeví jako přijatelnější druhá z možností. Pro konstrukci parseru na rozpoznávání formulí PL1 byl zvolen nástroj **ANTLR** (ANother Tool for Language Recognition [12]), který dokáže z předložené gramatiky vygenerovat parser do mnoha programovacích jazyků (C++, Java, C#, Python a další). Gramatiky, které byly použity pro vygenerování jednotlivých parserů nalezneme na přiloženém CD. V pozdější fázi tvoření gramatik pro systém ANTLR se u formulí s kvantifikátory ukázalo jako problematické přinutit parser generovat abstraktní syntaktický strom, který by korespondoval s naší statickou strukturou formule. Tento problém byl vyřešen přidáním imaginárního tokenu „QNTS“, který bude kořenem AST právě tehdy, když formule obsahuje kvantifikátory. Kvantifikátory jsou v tomto případě „od leva“ potomky kořene QNTS a zbytek formule je pak posledním potomkem kořene. U formulí bez kvantifikátorů je kořenem AST přímo daná formule.

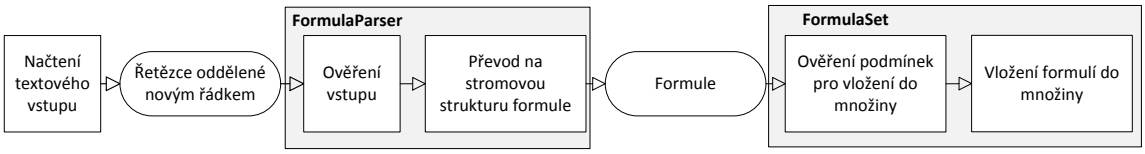
Třídní a sekvenční diagramy pro zpracování uživatelského vstupu nalezneme v příloze - 40, 50.



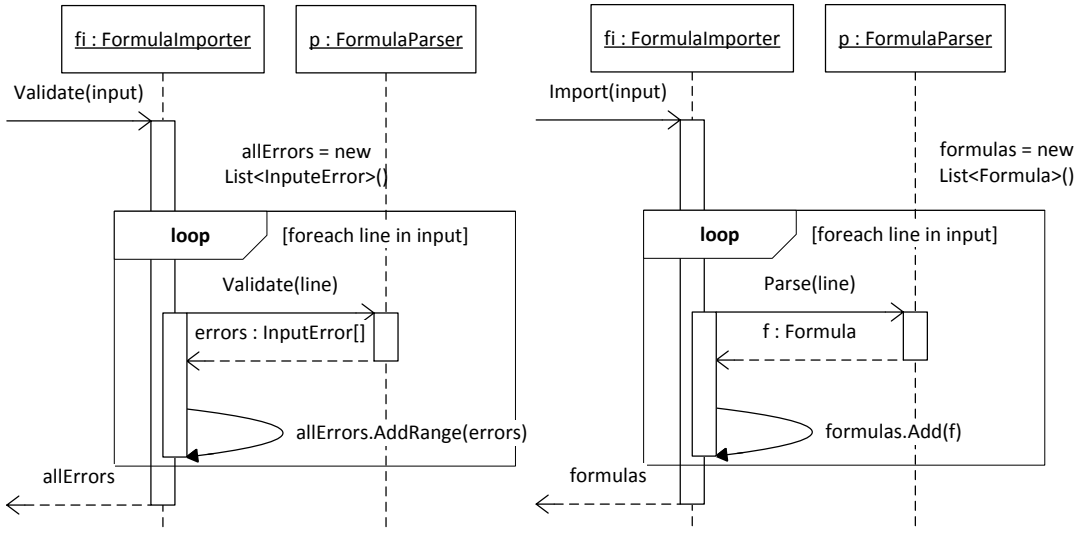
Obrázek 14: Princip převodu syntaktického stromu na strukturu formule

5.5 Import formulí

Se zpracováním uživatelského vstupu z textového řetězce přímo souvisí import formulí. Systém v obou prostředích umožní uživateli vložit více formulí najednou a to buď ze souboru nebo přímo z textového pole. Přitom bude umožněn výběr vstupního formátu - standardní zástupné symboly (V, !, ...) nebo formát TEX. Systém přitom využije stávající infrastrukturu pro převod jedné formule z textového řetězce. Systém také během importu musí zkontrolovat, zda jednotlivé formule jsou správně utvořené, vyhovují požadovaným omezením a jsou splněny podmínky pro vložení do množiny (viz kapitola 5.3). Blokové schéma průběhu importu můžeme vidět na obrázku 15. Veškerá funkcionalita importu bude zahrnuta ve třídě *FormulaImporter*. Sekvenční diagramy fáze ověření vstupu a převedení z textového řetězce na interní stromovou strukturu můžeme vidět na obrázku 16.



Obrázek 15: Blokové schéma importu formulí



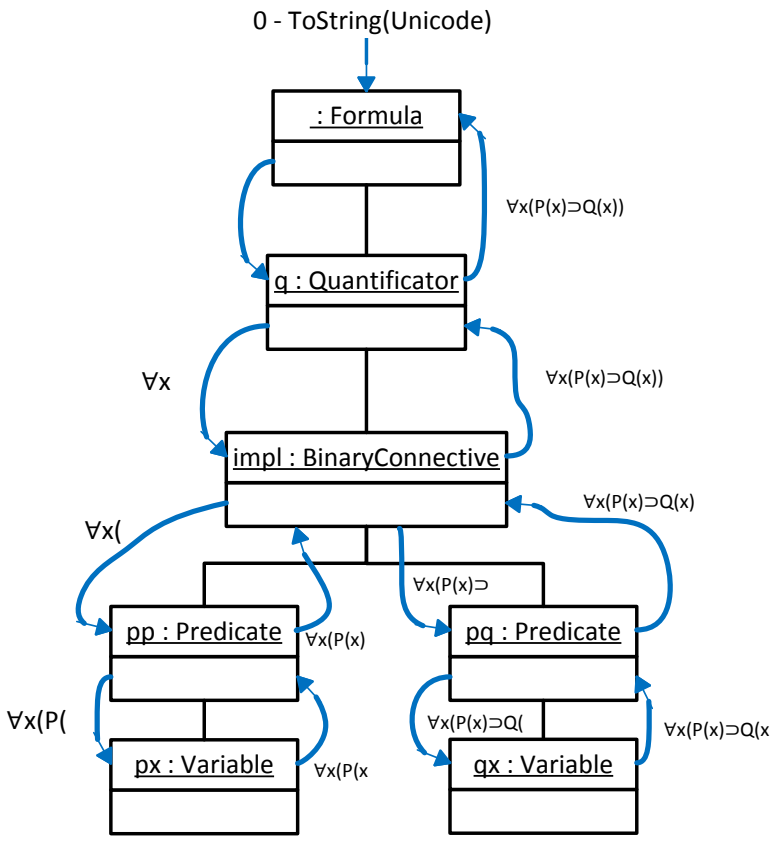
Obrázek 16: Sekvenční diagram importu formulí

5.6 Převedení formule zpět do textového řetězce

Pro účely zobrazení nebo uložení do souboru musíme být také schopni převést formule ze stromové struktury zpět na textový řetězec. Vzhledem ke způsobu reprezentace formulí

je celý proces více méně triviální. Všimněme si, že každá třída má definovánu funkci ToString(**in** format : FormulaStringFormat): **string** (viz diagramy 38, 39), kterou přezvala ze základního rozhraní *IFormulaElement*. Každý element formule tedy „zobrazí“ sám sebe a případně požádá své potomky o zobrazení do požadovaného formátu. Příklad převedení formule $\forall x(P(x) \supset Q(x))$ zpět do textového řetězce vidíme na obrázku 17. **FormulaStringFormat** je výčtovým typem pro jednotlivé typy řetězce:

- **Unicode** - tradiční zápis formule ($\forall$, $\exists$, $\supset$, ...)
- **Internal** - zápis ze zástupnými znaky (V, 1, $>$, ...)
- **Tex** - zápis ve formátu TEX(\forall, \exists, ...)



Obrázek 17: Převod formule $\forall x(P(x) \supset Q(x))$ zpět na textový řetězec

5.7 Vyhodnocení pravdivosti formule v interpretaci

Zcela zásadní pro korektní fungování celého systému je vyhodnocení pravdivosti formule v interpretaci, respektive množiny formulí. V této podkapitole si popíšeme princip algoritmu pro toto vyhodnocení. Celý algoritmus je v podstatě převedení definicí 2.11 a 2.12

do objektově orientovaného návrhu. V definici 2.11 jsme se zmínili, že kvantifikátory jsou zobecněním konjunkce, respektive disjunkce a pro konečné universum platí následující ekvivalence formulí:

- $\forall x A(x) \Leftrightarrow A_1(x) \wedge A_2(x) \wedge \ldots \wedge A_n(x)$
- $\exists x A(x) \Leftrightarrow A_1(x) \vee A_2(x) \vee \ldots \vee A_n(x)$

Řekněme, že universum bude množina $U = \{i_0, i_1, \ldots, i_n\}$. Vyhodnocení pravdivosti v interpretaci pro formuli $F = \forall x \exists y P(x, y)$ lze převést na formuli

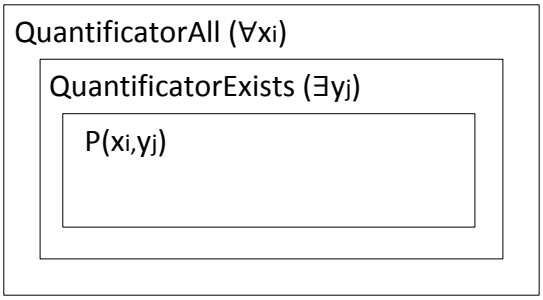
$$\begin{aligned} F' = (P(x_0, y_0) \vee P(x_0, y_1) \vee \ldots \vee P(x_0, y_j)) \wedge \\ (P(x_1, y_0) \vee P(x_1, y_1) \vee \ldots \vee P(x_1, y_j)) \wedge \ldots \wedge \\ (P(x_i, y_0) \vee P(x_i, y_1) \vee \ldots \vee P(x_i, y_j)) \end{aligned}$$

kde x_i a y_j je konkrétní valuace e přiřazená proměnným x a y. Náš algoritmus, nebo přesněji skupina algoritmů, musí tedy umět především tři věci. Tou první je generování valuací pro x_i, případně y_j, musí umět pracovat s kvantifikátory a za třetí musí umět vyhodnotit pravdivost formule pro valuace $(x_i, y_j, ...)$.

Nejprve se podíváme na práci s kvantifikátory. Vzhledem k tomu, že kvantifikátory máme povoleny pouze v hlavě formule, tak si můžeme dovolit nahradit kvantifikátory jednoduchými programovými konstrukcemi a vyhnout se tak relativně komplikovanému převodu z formule F na F'. Pro kvantifikátory můžeme využít následující dvě konstrukce:

```
function QuantificatorAll(): bool
    for x_i in Universum:
        if not vyhodnocení zbytku formule pro e(x) = x_i
            return false
    return true

function QuantificatorExists(): bool
    for x_i in Universum:
        if vyhodnocení zbytku formule pro e(x) = x_i
            return true
    return false
```

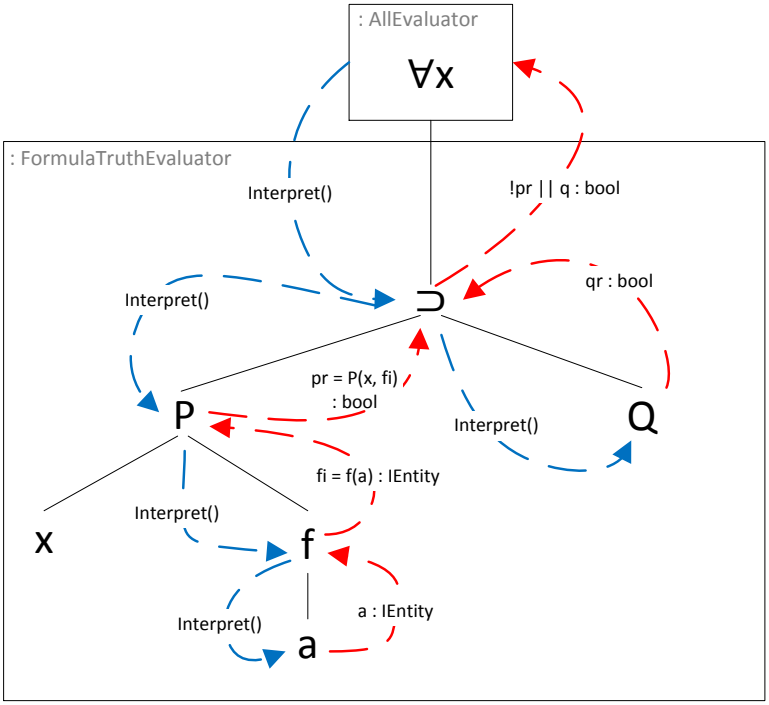
Doplnění těchto konstrukcí o možnosti zanoření pak v podstatě dostáváme náhradu za převod mezi formulemi F a F'. Blokové schéma zanoření těchto konstrukcí pro formuli $\forall x \exists y P(x, y)$ můžeme vidět na obrázku 18.



Obrázek 18: Zanoření kvantifikátorových konstrukcí pro fomuli $\forall x \exists y P(x, y)$

V našem objektově orientovaném návrhu jsou z praktických důvodů obě konstrukce reprezentované samostatnými třídami - *ExistsEvaluator* a *AllEvaluator* (viz diagram na obrázku 42). Pro upřesnění celého postupu jsou ještě přiloženy dílčí výpisy kódu - 9, 11 a 10.

Když víme, jak budeme zacházet s kvantifikátory, podívejme se nyní krátce na vyhodnocení pravdivosti formule pro konkrétní valuace $(x_i, y_j, ...)$. Opět jako v předchozích případech využijeme stromové struktury formule a aplikujeme na ní poněkud modifikovaný algoritmus průchodu do hloubky, který bere v potaz symboly podléhající interpretací a jednotlivé logické spojky. Tuto funkcionalitu v sobě ukrývá třída *FormulaTruthEvaluator*. Princip průchodu formulí je znázorněn na obrázku 19.



Obrázek 19: Průchod formulí $\forall x(P(x, f(a)) \supset Q(x))$ při vyhodnocení pravdivosti v IS

Posledním aspektem celého algoritmu je způsob generování valuací, který má na starosti třída *ValuationGenerator* (viz třídní diagram 42). Tato třída má definovánu funkci GetValuation(**int** varIndex, **int** entityIndex) : Valuation jejímž smyslem je, pro proměnnou x_i vrátit valuaci e_i. Funkce musí navíc zajistit správné generování valuací na základě „pořadí proměnných" ve formuli. U formule $\exists x \forall y P(x, y)$ je třeba generovat valuace (x_i, y_j), zatímco u formule $\exists y \forall x P(x, y)$ musíme generovat (y_i, x_j). Vezměme například formuli $\exists y \forall x P(x, y)$. Výsledkem volání funkce *GetValuation(0,5)* je valuace y_6. Příklad implementace třídy *ValuationGenerator* můžeme vidět na výpisu 8.

Aby algoritmus pro vyhodnocení pravdivosti formule v interpretaci pracoval korektně, je potřeba před spuštěním celého procesu vyhodnocení provést ještě tyto dvě úpravy formule:

- odstranění nepoužitých kvantifikátorů: $\forall x \exists y P(x) \rightarrow \forall x P(x)$
- nahrazení volných proměnných $\forall$ kvantifikátorem: $\exists x P(x, y) \rightarrow \forall y \exists x P(x, y)$

Tímto bychom celý popis algoritmu završili. Nicméně, aby byla analýza kompletní v příloze ještě nalezneme sekvenční diagram 52 celého postupu. Ještě, než se však pustíme do analýzy prostředí pro vytváření interpretačních struktur, řekneme si, jak probíhá ověření modelu množiny.

Ověření modelu množiny

Dle definice 2.12 je model množiny formulí $\{A_1, \ldots, A_n\}$ taková interpretace I, ve které jsou pravdivé všechny formule $A_1, \ldots, A_n$. Vzhledem k tomu, že v této fázi již máme algoritmus pro vyhodnocení pravdivosti jedné formule v interpretaci, je ověření modelu množiny v zásadě triviální věcí. Stačí nám tento algoritmus postupně spustit na všechny formule, celkovým výsledkem pak bude konjunkce jednotlivých pravdivostních hodnot. Průběh ověření ověření modelu množiny můžeme vidět na sekvenčním diagramu 53.

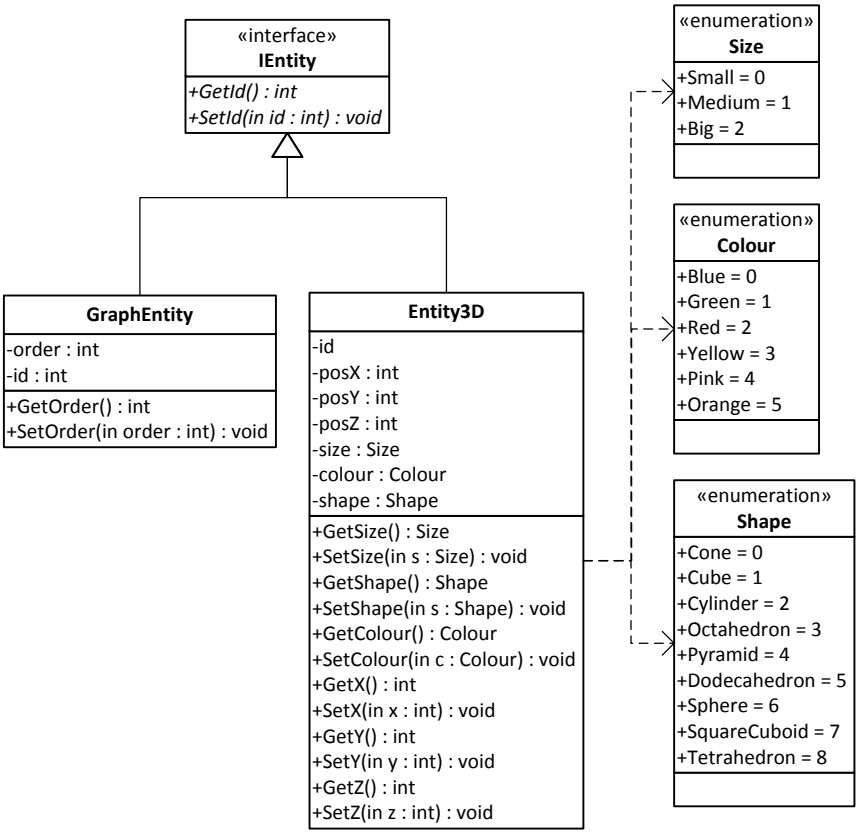
5.8 Prostředí pro vytváření interpretačních struktur

Pakliže jsme provedli analýzu obecných, respektive sdílených částí systému, nastal nyní ten pravý čas pustit se do analýzy jednotlivých prostředí. Začneme s prostředím pro vytváření interpretačních struktur. Nejprve se podíváme na universum a jeho vizualizaci v 3D prostoru, dále uvidíme jakým způsobem probíhá interpretace jednotlivých symbolů a na závěr se zastavíme u několika specifik pro ověření modelu množiny formulí v tomto prostředí.

5.8.1 Universum a jeho vizualizace

První věcí, na kterou se zaměříme je objektová reprezentace individuí. Vzhledem k tomu, že algoritmus pro ověření modelu množiny jsme sestavili nezávisle na zvoleném prostředí, je pro jednotlivá individua třeba vytvořit universální rozhraní. Jak vidíme na třídním diagramu 20 tímto rozhraním je *IEntity*, které implementují dvě konkrétní třídy *GraphEntity* a

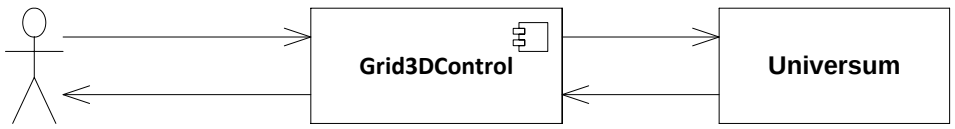
Entity3D. Třída *GraphEntity* nám poslouží později při analýze „relačních struktur“. Individua v prostředí pro vytváření interpretačních struktur PL1 budeme reprezentovat třídou *Entity3D*. Tato třída zahrnuje všechny požadované vlastnosti (typ tvaru, barva, velikost a pozice) popsané v kapitole 3.1. Navíc obsahuje atribut *id*, jednoznačně identifikující dané individuum.



Obrázek 20: Třídní diagram individuí

Celé universum budeme reprezentovat třídou *Universum* (viz třídní diagram 43), která nabízí základní sadu operací tj. - přidání, úprava nebo odebrání individua. Stejně jako u množiny formulí je i zde snaha oddělit interní reprezentaci universa od uživatelského rozhraní. Proto veškerá interakce uživatele s universem, včetně jeho vykreslení, bude probíhat přes třídu *Grid3DControl*, jak můžeme vidět na obrázku 21. Tato třída oproti „standardním“ operacím, které bychom od ní očekávali, disponuje i sadou operací, které mají za úkol informovat zbytek systému v případech, kdy uživatel změnil pozici individua v prostoru nebo vybral interpretaci konstanty.

Samotným vykreslením celé trojrozměrné scény s universem se v této práci zabývat nebudeme, to je totiž silně závislé na konkrétní zvolené technologii pro implementaci. Nicméně abychom mohli se zobrazenými tvary během analýzy pracovat, byla pro ně zavedena abstrakce v podobě třídy *VisualShape*. Předpokládá se korespondence $1:1$ instancí



Obrázek 21: Blokové schéma interakce uživatele s universem

tříd *Entity3D* a *VisualShape*. Dle požadavků musí systém uživateli umožnit plnou interakci s universem, tj. přidání, odebrání, přesunutí nebo změnu tvaru individua. Všechny tyto procesy jsou opět silně závislé na použité technologii, proto zůstaneme pouze u jejich konceptuálního popisu pomocí sekvenčních diagramů - 54, 55, 56 a 57.

Interakce uživatele s universem probíhá v několika „módech“ v závislosti na tom, zda například aktuálně probíhá interpretace konstanty. Význam těchto módů reprezentovaných typem *GridSelctionMode* je následující:

- **Normal** - standardní mód. Uživateli je umožněna plná interakce s universem
- **Constant** - mód výběru interpretace konstanty. V tomto módu musí komponenta zobrazující universum zajistit, aby uživatel nemohl měnit universum. Po kliknutí na individuum třída *Grid3DControl* vyvolá událost *OnConstantValueSelected* informující zbytek systému o vybrané interpretaci pro konstantu (viz podkapitola 5.8.2.2)
- **Function** - mód výběru funkce. I v tomto módu je uživateli zakázáno měnit universum. Po kliknutí na individuum třída *Grid3DControl* vyvolá událost *OnFunctionValueSelected* informující zbytek systému o dalším vybraném obrazu (viz podkapitola 5.8.2.3).

5.8.2 Interpretace symbolů

V této kapitole se zaměříme, na to jakým způsobem systém pracuje se symboly podléhajícími interpretaci. V první řadě je si třeba ujasnit, o které symboly se vlastně jedná a jak vypadají jejich interpretace. Z kladených omezení na formule PL1, je jasné, že systém bude pracovat pouze s unárními, respektive binárními, predikáty, unárními funkcemi a konstantami (nulární funkce). O samotné zobrazení jednotlivých interpretací se bude starat komponenta *InterpretationVisualizer* (viz třídní diagram 44). Výstup zobrazení by měl vypadat podobně jako na obrázku 5.

5.8.2.1 Predikátové symboly

Predikátové symboly v našem systému interpretujeme unárními, respektive binárními, relacemi na universu prostorových tvarů. Z požadavků již víme, že systém v obou prostředích nabídne na výběr uživateli jednu z předem připravených relací. Jednotlivé relace boudou „na venek" se zbytkem systému komunikovat skrze obecné rozhraní *IPredicateInterpretation* (viz třídní diagram 39), které má definovánu jednu funkci:

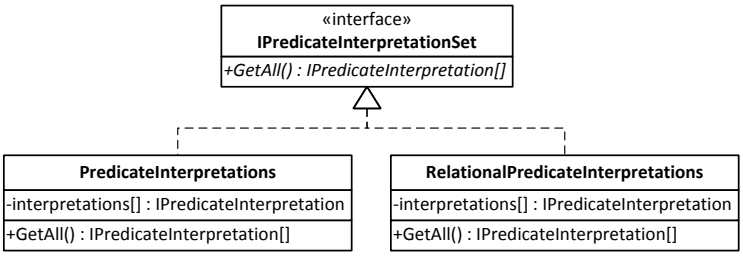
Interpret (arguments[] : IEntity) : **bool**

Kontraktem této funkce je „*Vrať hodnotu **true** právě tehdy, když argumenty patří do dané relace*". Ve fázi, kdy je tato funkce volána, se již počítá s tím, že byly vyhodnoceny všechny argumenty daného predikátu. U formule $P(f(b))$ bude funkce *Interpret* predikátového symbolů P volána až ve chvíli, kdy byly vyhodnoceny termy f a b, což lze také zřetelně vidět na obrátku 19. Příklad unární relace „x má červenou barvu" můžeme vidět na výpisu 3, některé další relace pak nalezneme v příloze na výpisu 12. Interakce mezi uživatelem a systémem během interpretace predikátových symbolů je pak znázorněna na sekvenčním diagramu 58.

```
public bool Interpret(object ctx, IEntity [] e) {
  return ((Entity3D)e[0]).GetColour() == Colour.Red;
}
```

Výpis 3: Příklad implementace relace - x má červenou barvu

Vzhledem k tomu, že binární predikátové symboly bude systém muset umět interpretovat i v prostředí pro „vytváření relačních struktur", jež ovšem bude mít svou vlastní předpřipravenou množinu interpretací, je vhodné vytvořit abstrakci nad „zdrojem" interpretací, tak abychom mohli sdílet funkcionalitu třídy *InterpretationVisualizer* v obou prostředích. Vytvoříme proto obecné rozhraní *IPredicateInterpretationSet* (viz třídní diagram 22), kdy v prostředí pro „vytváření relačních struktur" bude systém využívat třídu *RelationalPrediateInterpretation* namísto třídy *PredicateInterpretations*.



Obrázek 22: Třídní diagram „zdrojů" interpretací predikátových symbolů

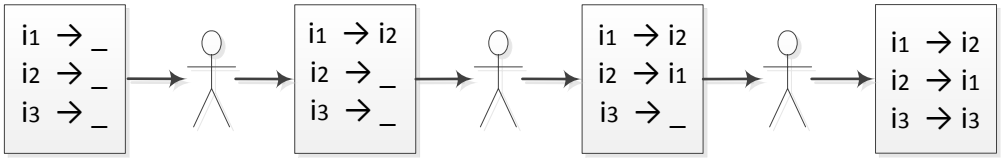
5.8.2.2 Funkční symboly s aritou $n = 0$ (konstanty)

Konstanty v PL1 interpretujeme přiřazením jednoho konkrétního individua universa. V našem případě to znamená přiřadit instanci třídy *ConstantType* jednu konkrétní instanci

třídy *Entity3D*, která jak jsme si ji řekli představuje právě individuum. Uživatel vybírá interpretaci konstanty kliknutím na individuum ve 3D zobrazení. Celý tento proces je přehledně zobrazen na sekvenčních diagramech 59, 60 a 61.

5.8.2.3 Funkční symboly s aritou $n = 1$

Posledním symbolem podléhajícím interpretaci jsou funkce s aritou $n = 1$. Tyto interpretace budeme reprezentovat pomocí „tabulky“, které ve statické struktuře odpovídá třída *FunctionTable*. Uživatel danou tabulku vytváří postupným výběrem individuí, dokud tabulka není kompletní. Blokové schéma interakce uživatele se systémem ve fázi definice funkční tabulky můžeme vidět na obrázku 23. Jednotlivé sekvenční diagramy týkající se interpretace funkcí jsou v příloze na obrázcích 44, 22 a 58. Systém navíc ještě během vytváření tabulky zobrazí v trojrozměrném universu identifikátory jednotlivých individuí, tak aby je uživatel mohl lépe identifikovat.



Obrázek 23: Blokové schéma vytváření interpretace funkčního symbolu

5.8.3 Ověření modelu množiny formulí

Poslední věcí, na kterou se v prostředí pro vytváření interpretační struktury množiny formulí podíváme, bude samotné ověření modelu. V kapitole 5.7 jsme si popsali algoritmus pro vyhodnocení pravdivosti formule v interpretaci a obecný postup při ověření modelu množiny. Aby byl systém v tomto prostředí schopen popsané algoritmy využít, musí provést jeden důležitý krok a tím je kontrola, zda uživatel interpretoval všechny potřebné symboly z množiny formulí. Speciálně musí systém zkontrolovat, zda jsou správně utvořeny tabulky pro interpretace funkcí. Poté již můžeme využít dříve vytvořenou infrastrukturu a provést samotné ověření modelu množiny. Celý tento postup je znázorněn v příloze na sekvenčním diagramu 66.

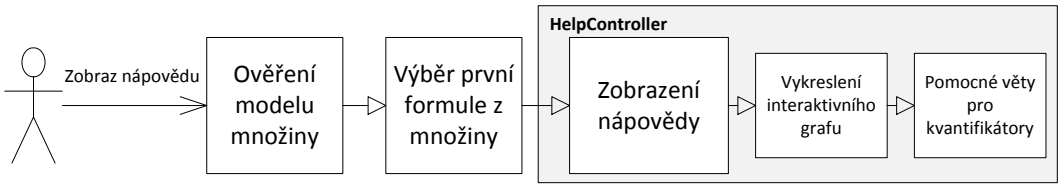
5.9 Nápověda pro vytvoření modelu

V této kapitole se zaměříme na jednotlivé aspekty nápovědy pro vytvoření modelu. Celá nápověda bude složena z několika dílčích prvků:

- Interaktivní graf se zobrazenými pravdivostními hodnotami vybrané formule při všech možných valuacích
- Pomocné věty pro formule s kvantifikátory
- Nápověda pro binární logické spojky

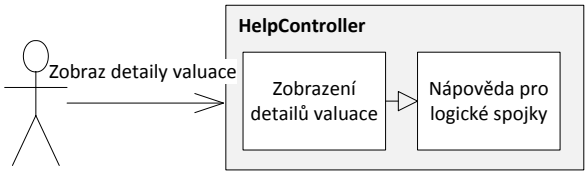
- Zobrazení výsledků funkcí a pravdivostních hodnot jednotlivých predikátů pro vybrané valuace

Systém uživateli neradí jak sestavit IS, která by byla modelem množiny formulí, ale spíše mu dopomáhá k tomu, aby zjistil, kde udělal chybu. Z tohoto důvodu je nutné, aby uživateli bylo umožněno vyvolat nápovědu pouze v případě, že je interpretační struktura kompletní. Blokové schéma prvního zobrazení nápovědy můžeme vidět na následujícím obrázku 24.



Obrázek 24: Blokové schéma zobrazení nápovědy pro vytvoření modelu

Po vykreslení interaktivního grafu a pomocných vět pro kvantifikátory, již je uživateli dovoleno zobrazovat detaily o jednotlivých valuacích (viz blokové schéma na obrázku 25). Zatímco je zobrazená nápověda, uživatel může i nadále provádět změny v interpretační struktuře, na které bude rozhraní reagovat patřičnou aktualizací interaktivního grafu. Aktualizace grafu bude také provedena v případě, že uživatel vybere jinou formuli z množiny formulí.



Obrázek 25: Blokové schéma zobrazení detailů valuace

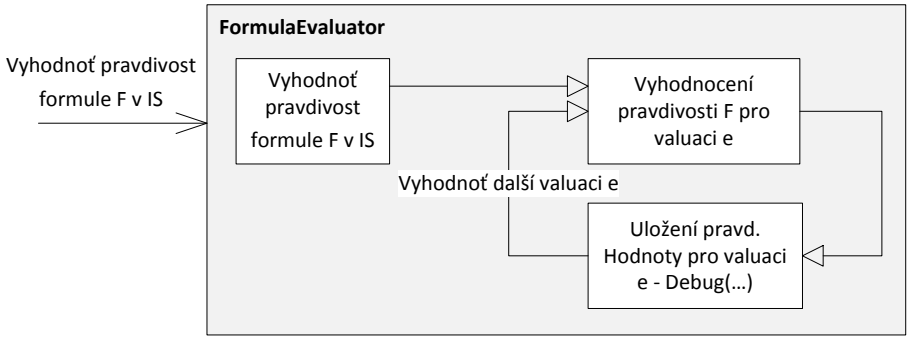
5.9.1 Získání pravdivostních hodnot formulí pro všechny valuace

Zcela zásadní pro vytvoření celé nápovědy je získání pravdivostních hodnot formulí při všech valuacích. Vzhledem k tomu, že již máme algoritmus pro ověření modelu formule, bude tento úkol pro nás relativně jednoduchý. Na třídním diagramu 42 si všimněme, že třída *FormulaSetEvaluator* v jednom z přetížení funkce *EvaluateAll* očekává jako parametr třídu implementující rozhraní *IValuationDebugger*. Toto rozhraní má definovánu funkci

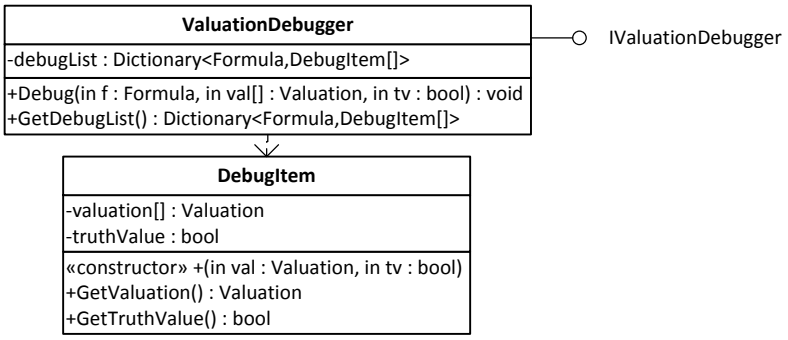
$$\text{Debug}(\text{f} : \text{Formula}, \text{val}[] \ : \ \text{Valuation}, \ \text{truthValue} \ : \ \textbf{bool})$$

která slouží právě k uložení pravdivostní hodnoty pro konkrétní valuaci určité formule. Tato funkce je volána po každém vyhodnocení pravdivostí formule pro určitou

valuaci. Blokové schéma procesu získávání pravdivostních hodnot pro jednotlivé valuace je znázorněno na obrázku 26. Interní fungování třídy *FormulaEvaluator*, která slouží pro vyhodnocení pravdivosti formule v interpretaci, v kontextu nápovědy můžeme vidět v příloze na sekvenčním diagramu 52.



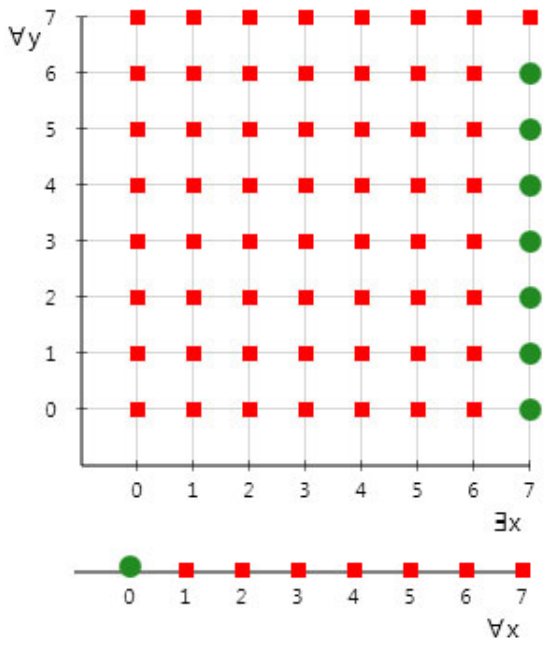
Obrázek 26: Blokové schéma získání pravdivostních hodnot při jednotlivých valuacích



Obrázek 27: Třídní diagram získání pravdivostních hodnot při jednotlivých valuacích

5.9.2 Vizualizace pravdivostních hodnot formule při všech valuacích

Pokud víme, jakým způsobem systém získá pravdivostní hodnoty formulí při jednotlivých valuacích, můžeme přistoupit k vytvoření interaktivního grafu, z takto nasbíraných dat. Způsobů jakým graf vykreslit je nepřeberné množství, od použití existujících knihoven, přes vlastní implementaci. Proto si vysvětlíme jen význam jednotlivých komponent grafu. Příklad vykresleného grafu můžeme vidět na obrázku 28. Dimenze závisí na počtu proměnných ve formuli - systém tedy bude kreslit jedno či dvou rozměrné grafy. Hodnoty osy x, případně y, představují jednotlivé valuace proměnných x_i, y_i. Barevné body, pak pravdivostní hodnotu formule při dané valuaci. Zelená pro valuace, ve kterých byla formule splněna a červená, kde splněna nebyla. Po kliknutí na určitý bod systém zobrazí detaily o vyhodnocení pravdivosti formule pro danou valuaci.



Obrázek 28: Příklad zobrazení interaktivního grafu pro 1 a 2 kvantifikátory

5.9.3 Pomocné věty pro formule s kvantifikátory

V kapitole Funkční požadavky 3.1 jsme definovali znění jednotlivých pomocných vět pro formule s kvantifikátory. Tyto věty budou zobrazeny pod interaktivním grafem, aby bylo uživateli jasné, jak by měl tento graf vypadat, aby interpretační struktura byla modelem dané formule. Všechny takto definované věty si nyní zobecníme, tj. vytvoříme z nich šablony, které lze použít pro všechny typy formulí, nezávisle na pořadí proměnných:

- $\forall v^1$ - formule musí být splněna pro všechna ohodnocení $e(v_i^1)$
- $\exists v^1 A$ - formule musí být splněna alespoň pro jedno ohodnocení $e(v_i^1)$
- $\forall v^1 \forall v^2$ - formule musí být splněna pro všechna ohodnocení $e(v_i^1)$, $e(v_j^2)$
- $\forall v^1 \exists v^2$ - pro každé ohodnocení $e(v_i^1)$ musí existovat alespoň jedno ohodnocení $e(v_j^2)$, ve kterém je formule splněna
- $\exists v^1 \forall v^2$ - musí existovat alespoň jedno ohodnocení $e(v_i^1)$, ve kterém je formule splněna pro všechna ohodnocení $e(v_j^2)$
- $\exists v^1 \exists v^2$ - formule musí být splněna alespoň pro jedno ohodnocení $e(v_i^1)$, $e(v_j^2)$

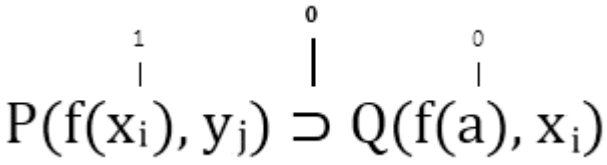
Abychom vybrali správnou větu pro danou formuli, systém musí zjistit jaké je pořadí kvantifikátorů. Toto zle provést velmi jednoduše tak, že procházíme strom formule od

jeho kořene a zastavíme se, jakmile narazíme na první vrchol, který není instancí třídy *Quantificator*. Dle zjištěného pořadí kvantifikátorů vybereme korektní větu, ve které pak nahradíme symboly v^1 nebo v^2 skutečnými názvy proměnných. Jako příklad uveďme formuli $\forall y \exists x P(x, y)$, pro kterou systém vybere větu $\forall v^1 \exists v^2$. Nahrazením symbolů v^i pak dostaneme tuto větu:

„pro každé ohodnocení $e(y_i)$ musí existovat alespoň jedno ohodnocení $e(x_j)$, ve kterém je formule splněna“

5.9.4 Nápověda pro binární logické spojky

Dílčí nápovědu pro binární logické spojky vytvoříme s využitím třídy *FormulaTruthEvaluator*, která vyhodnocuje pravdivostní hodnotu formule pro valuaci. Zde však použijeme funkci GetTruthValueDebug(), vracející i pravdivostní hodnoty logických spojek a predikátů. Takto získané pravdivostní hodnoty pak vykreslíme, způsobem uvedeným na obrázku 29. Při vykreslování z formule odstraníme kvantifikátory a proměnným x, případně y, přiřadíme patřičné indexy i a j v závislosti na jejich pořadí ve formuli. Například pro $\forall x \forall y(P(f(x), y) \supset Q(f(a), x))$ přiřadíme proměnné x index i a y index j.



Obrázek 29: Nápověda pro binární logické spojky

5.9.5 Zobrazení dalších detailů vyhodnocení pravdivosti formule při dané valuaci

Po výběru valuace z interaktivního grafu systém uživateli zobrazí další detaily o vyhodnocení pravdivosti formule při dané valuaci. Konkrétně se jedná o výsledky funkčních zobrazení a pravdivostní hodnoty predikátů. Vezměme formuli $\forall x \forall y(P(f(x), y) \supset Q(f(a), x))$. Universum zadejme jako množinu $U = \{i_1, i_2, i_3\}$. Funkční zobrazení f interpretujeme tabulkou $[i_1 \to i_2, i_2 \to i_3, i_3 \to i_1]$ a jako interpretaci konstanty a zvolíme individuum i_2. Uživatel se nyní rozhodne zobrazit detaily o vyhodnocení pravdivosti formule pro valuaci $e(x_3) = i_2$ a $e(y_1) = i_1$. Systém uživateli zobrazí následující:

$$f(a) = i_3 \quad f(x_3) = i_1 \quad P(i_1, y_1) = 0 \quad Q(i_3, x_1) = 1$$

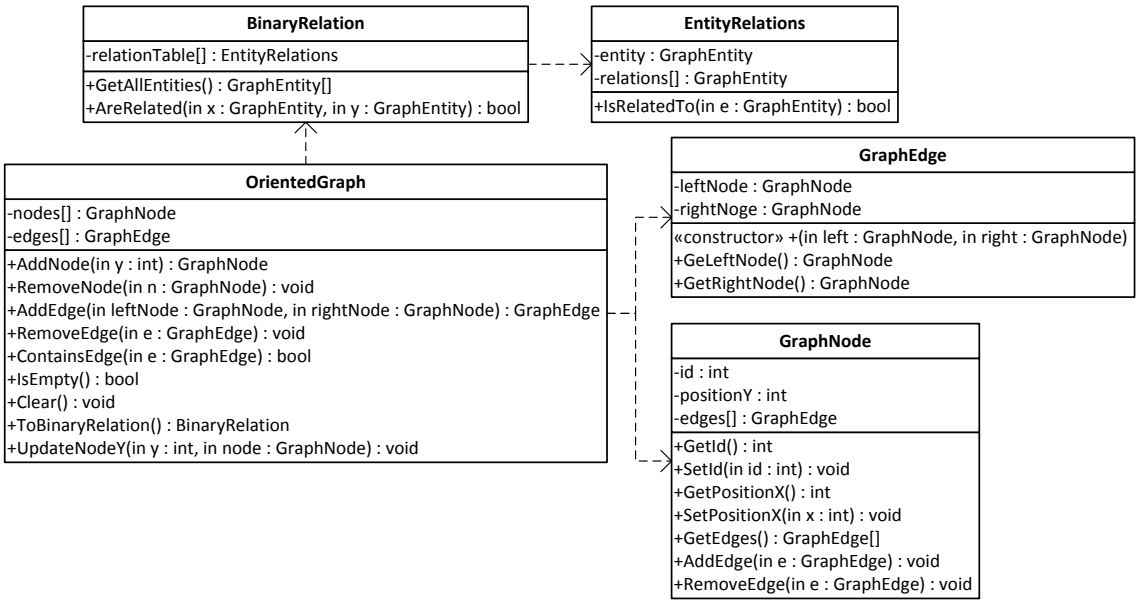
Zobrazené pravdivostní hodnoty predikátových symbolů jsou pouze příkladem. Jejich skutečná pravdivostní hodnota bude samozřejmě záviset za zvolené interpretaci. Pravdivostní hodnoty predikátových symbolů a výsledky funkčních zobrazení získáme analogickým postupem, jako při vyhodnocení pravdivostní hodnoty formule, uvedeným na obrázku 19.

5.10 Prostředí pro modelování vlastností relací

Druhým typem prostředí, které bude systém obsahovat je prostředí pro modelování vlastností relací. Jak jsme si již řekli v kapitole specifikace požadavků, tak „vlastnosti relací“ budeme modelovat pomocí orientovaného grafu zasazeného do 2-rozměrného kartézského systému souřadnic, tak abychom byli schopni modelovat i relace typu „větší než“ a další. Uspořádání vrcholů grafu budeme brát dle jejich pozice na ose y.

Pro reprezentaci množiny formulí, uživatelský vstup a ověření, zda struktura splňuje požadované vlastnosti využijeme již existující infrastrukturu popsanou v kapitolách 5.3, 5.4 a 5.7.

Prvním aspektem, který je potřeba si ujasnit je reprezentace orientovaného grafu $G = (V, E)$, jehož statickou strukturu můžeme vidět na obrázku 30. Jednotlivé vrcholy grafu $v_i \in V$ reprezentujeme třídou *GraphNode*, která má dva hlavní atributy - *id* a *positionY*. Atribut *id* je číslo jednoznačně identifikující daný vrchol grafu, zatímco *positionY* je pozice vrcholu na ose y kartézského systému. Hrany $e_i \in E$ jsou pak představeny třídou *GraphEdge* se dvěma atributy *leftNode* a *rightNode* udávající orientaci hrany. Veškeré operace nad orientovaným grafem probíhají skrze třídu *OrientedGraph*.



Obrázek 30: Třídní diagram orientovaného grafu

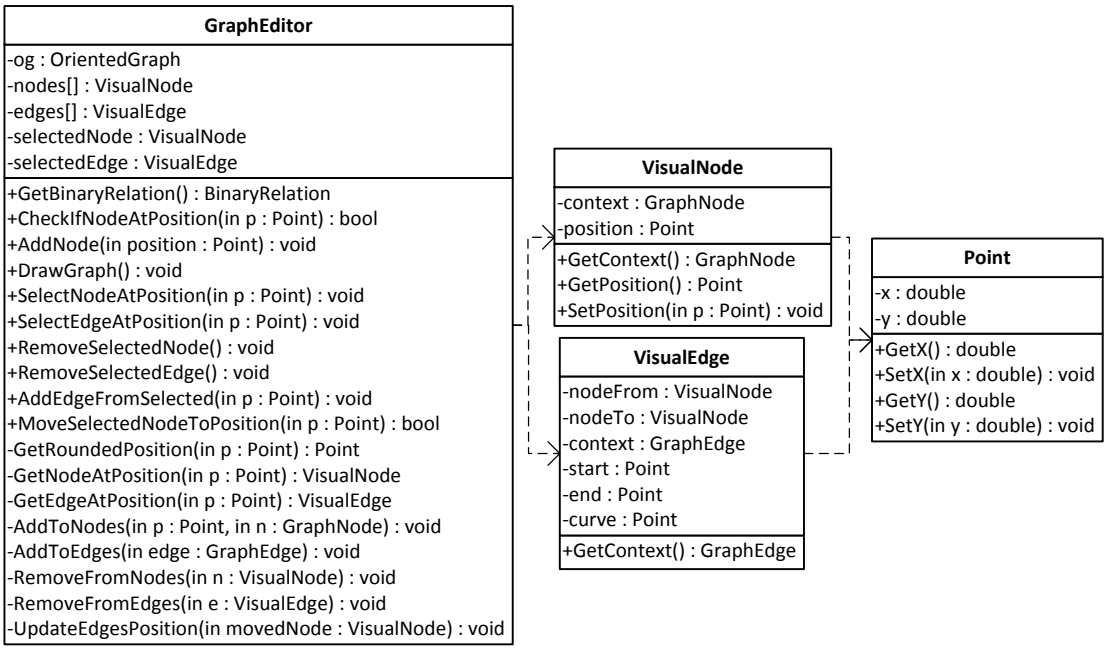
5.10.1 Vizualizace relační struktury

Nyní již víme, jak bude systém interně reprezentovat orientovaný graf. Podíváme se, jak lze na konceptuální úrovni uživateli umožnit pracovat s takovýmto grafem. Tzn. jak vytvořit jednoduchý editor orientovaných grafů. Veškerá interakce uživatele s grafem bude probíhat přes třídu *GraphEditor*, jak je znázorněno na blokovém schéma 31. Tato

třída má na starost jak vykreslení „vodících čar“ os kartézského systému, tak i vykreslení vrcholů a hran.

Obrázek 31: Blokové schéma interakce uživatele s orientovaným grafem

Dále je také potřeba oddělit visuální a interní reprezentaci hran a vrcholů. Na třídním diagramu 32 můžeme vidět, že tato abstrakce je provedena skrze třídy *VisualNode*, respektive *VisualEdge*. S takto definovanou statickou strukturou se můžeme nyní zaměřit na způsob kreslení hran a vrcholů.



Obrázek 32: Třídní diagram editoru orientovaného grafu

Vrcholy systém vykreslí jako kružnice se středem umístěným na zvolené pozici os x a y. Vzhledem k tomu, že hodnoty os x a y standardně nabývají reálných hodnot, systém provede jejich zaokrouhlení, tak aby korespondovaly s nejbližším průsečíkem vodicích čar kartézského systému. Tímto zajistíme poněkud jednoznačnější vykreslení vrcholů vzhledem k jejich pozici na ose y. Pro účely zaokrouhlení můžeme využít následující rovnici:

$$roundedPos = round(x/step) * step$$

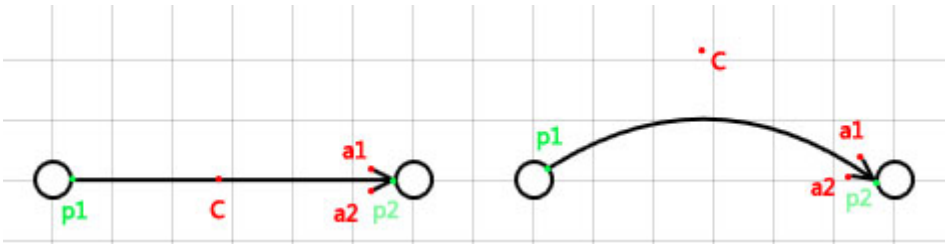
kde *round* je funkce pro standardní zaokrouhlení na celé číslo, *x* je původní pozice v reálných hodnotách a *step* je velikost kroku vodicích čar kartézského systému.

Hrany grafu systém kreslí pomocí orientovaných šipek. Proces tohoto vykreslení však není tak jednoduchý, jak by se mohlo na první pohled zdát. V první řadě musíme znát počáteční a koncové body - A, B. Systém musí zjistit pozice počátečního a koncového vrcholu dané šipky, které jsou uloženy v atributu *position* třídy *VisualNode*. Následně je potřeba provést posunutí těchto bodů za hranice kružnic vrcholů, tak aby nedocházelo k jejich překrytí šipkou. Pro tyto potřeby definujeme jednoduchou funkci pro výpočet bodu na přímce:

```
Point GetPointAtLine(double x1, double y1, double x2 , double y2, double distance)
{
    var v = new Vector(x2 − x1, y2 − y1); // Vektor primky
    v.Normalize(); // Normalizace vektoru
    return new Point(x1 + distance ∗ v.X, y1 + distance ∗ v.Y); // Bod na primce
}
```

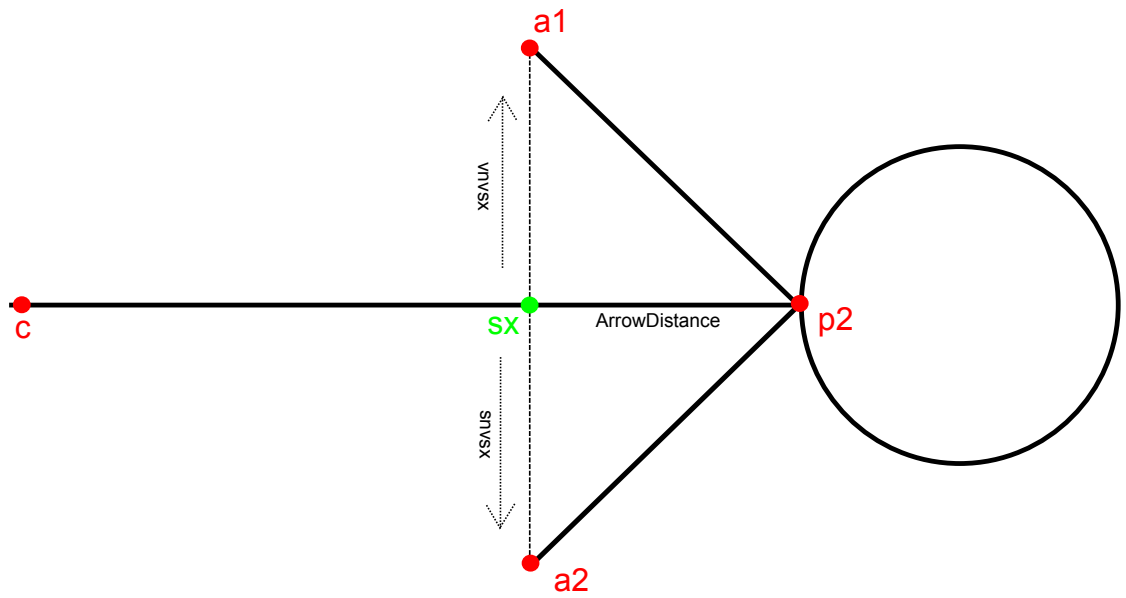
Výpis 4: Funkce pro výpočet bodu na přímce

kde parametry $x1, x2, y1, y2$ jsou jednotlivé složky bodů A, B a $distance$ je poloměr kružnice vrcholu. Pokud máme takto posunuté body, říkejme jim p_1 a p_2, můžeme přistoupit k samotnému kreslení hran grafu. Jako nejvhodnější se jeví použití tzv. „kvadratické Bézierovy křivky“ [14]. Jejich výhodnou je, že nám umožňují relativně jednoduché kreslení přímek, tak i mírně zakřivených čar, které se nám hodí pro zpřehlednění složitých struktur. Způsobem, jakým se taková křivka vykresluje, se zde zabývat nebudeme, nicméně pro její definici je třeba znát celkem tři body - počáteční p_1, koncový p_2 a tzv. bod zakřivení c. Zjištění bodu c pro přímky provede systém s pomocí funkce $GetPointAtLine$, kde parametr $distance$ je poloviční vzdálenost mezi body p_1 a p_2. Vzdálenost mezi body p_1 a p_2 vypočítáme pomocí známé rovnice $|AB| = \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2}$. Bod c pro „zakřivené čáry“ bude definován uživatelem. Příklad vykreslení hran můžeme vidět na obrázku 33.



Obrázek 33: Příklad vykreslení hran grafu pomocí kvadratických Bézierových křivek

Posledním problémem je zjištění bodů a_1 a a_2, tak abychom mohli vykreslit dvě přímky tvořící šipku. Tyto dva body lze relativně lehce určit pomocí vektorového počtu. Slovní popis postupu by však byl zbytečně komplikovaný proto se podívejme na ukázkovou implementaci na výpisu 5 a na ilustrační obrázek 34. Veškeré sekvenční diagramy interakce uživatele s komponentou orientovaného grafu nalezneme v příloze na obrázcích 74, 75, 76, 77 a 79.



Obrázek 34: Ilustrační obrázek pro výpočet bodů šipky

```
Point[] GetArrowPoints(Point p2, Point c)
{
    // Bod x pixelu od stredu vrcholu uzlu
    Point sx = GetPointAtLine(p2.X, p2.Y, c.X, c.Y, ArrowDistance);
    // Vektor smerem ke stredu vrcholu uzlu
    Vector vsx = new Vector(end.X − sx.X, end.Y − sx.Y);
    vsx.Normalize();

    // Dva normalove vektory z bodu sx (na kazdou stranu)
    Vector vnvsx = new Vector(−vsx.Y, vsx.X);
    Vector rnvsx = new Vector(vsx.Y, −vsx.X);

    vnvsx.Normalize();
    rnvsx.Normalize();

    // Vypocet bodu a1, a2 (X = A + tu)
    Point a1 = new Point(sx.X + ArrowSize ∗ vnvsx.X, sx.Y + ArrowSize ∗ vnvsx.Y);
    Point a2 = new Point(sx.X + ArrowSize ∗ rnvsx.X, sx.Y + ArrowSize ∗ rnvsx.Y);
    return new[]{a1,a2};
}
```

Výpis 5: Funkce pro výpočet bodů šipky

5.10.2 Ověření zda struktura splňuje požadované axiómy

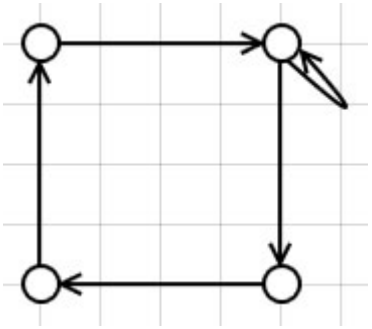
Ověření, zda struktura splňuje požadované axiómy systém provede pomocí známého algoritmu pro ověření modelu množiny formulí (viz kapitola 5.7). Abychom však mohli tento algoritmus využít, musíme převést orientovaný graf na kompatibilní strukturu.

Na jednotlivé vrcholy v_i orientovaného grafu budeme nahlížet jako na prvky universa. Orientované hrany e_i pak lze, na tomto universu, považovat za definici binární relace „x je v relaci s y“. Pracovně tuto relaci nazveme R.

Algoritmus pro ověření modelu množiny formulí očekává, že jednotlivá individua budou implementovat rozhraní *IEntity*. Toto zajistíme triviálním převodem z instancí třídy *GraphNode* na instance třídy *GraphEntity*. Binární relaci R budeme reprezentovat pomocí tabulky. Ve statické struktuře této tabulce odpovídá třída *BinaryRelation*. Počet řádků a sloupců takové tabulky bude odpovídat počtu uzlů v orientovaném grafu. Hodnoty buněk budou nabývat 0 nebo 1 (true / false), v závislosti na tom, zda jsou dané dva vrcholy v_i a v_j spojeny orientovanou hranou. Formálně můžeme hodnoty buněk definovat funkcí

$$f(i,j) = \begin{cases} 0 & \text{if } (v_i, v_j) \notin E \\ 1 & \text{if } (v_i, v_j) \in E \end{cases}$$

kde i, j jsou čísla řádků, respektive sloupců, tabulky a E je množina hran orientovaného grafu. Příklad převedení grafu z obrázku 35 můžeme vidět na tabulce 4.



Obrázek 35: Příklad orientovaného grafu

	v_0	v_1	v_2	v_3
v_0	0	1	0	0
v_1	0	0	1	0
v_2	0	0	1	1
v_3	1	0	0	0

Tabulka 4: Reprezentace binární relace tabulkou

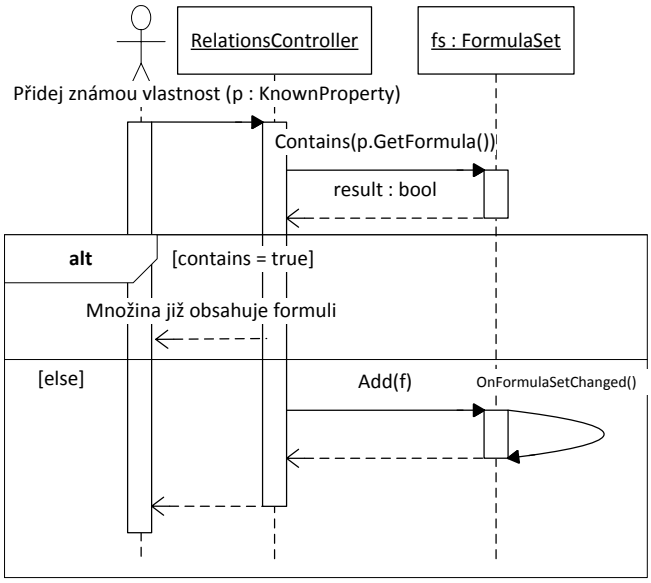
5.10.3 Známé vlastnosti relací

Vzhledem k tomu, že uživatelé budou asi nejčastěji modelovat „standardní“ vlastnosti relací, jako reflexivita či symetrie, tak pro zjednodušení práce, nabídne systém sadu připravených vlastností relací. Zvolenou vlastnost pak uživatel vloží do množiny formulí pouhým kliknutím na tlačítko a nebude nucen opakovaně formule vepisovat. Seznam všech těchto vlastností nalezneme v kapitole 2.6. Abychom pokročilejším uživatelům

umožnili úpravu této sady vlastností, uložíme jí ve formátu XML. Strukturu takovéhoto souboru si můžeme prohlédnout na výpisu 6. Způsob vložení vlastnosti do množiny formulí je pak znázorněn na sekvenčním diagramu 36.

```
<?xml version="1.0" encoding="utf-8" ?>
<knownProperties>
  <property name="Reflexivita" formula="VxR(x,x)" />
  <property name="I-reflexivita" formula="Vx!R(x,x)" />
  ...
</knownProperties>
```

Výpis 6: Struktura uložení známých vlastností relací



Obrázek 36: Sekvenční diagram vložení známé vlastnosti do množiny formulí

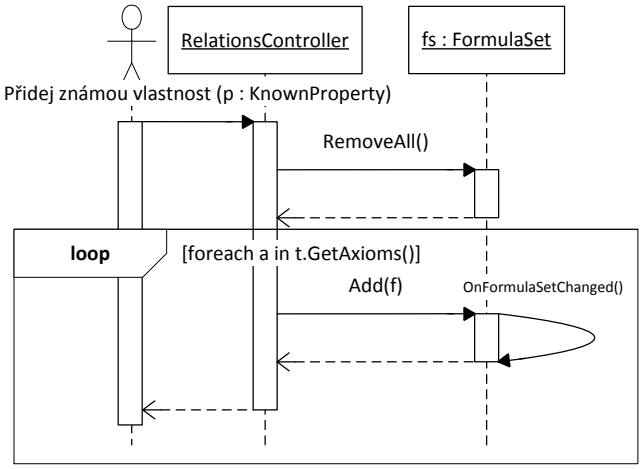
5.10.4 Speciální axiómy známých teorií

Pokud bude chtít uživatel modelovat strukturu, splňující například speciální axiómy teorie ostrého uspořádání, je stejně jako u „standardních" vlastností relací vhodné, aby nemusel jednotlivé axiómy ručně vepisovat, ale mohl je vložit pouhým kliknutím. I zde pro možnost pozdějších úprav uložíme speciální axiómy do formátu XML (viz výpis 7).

```
<?xml version="1.0" encoding="utf-8" ?>
<knownTheories>
  <theory name="Ostre usporadani (v3)" interpretationOfR="LessThan">
    <specialAxiom>VxVy(R(x,y)>!R(y,x))</specialAxiom>
    <specialAxiom>VxVyVz((R(x,y)*R(y,z))>R(x,z))</specialAxiom>
  </theory>
```

```
    ...
</knownTheories>
```

Výpis 7: Struktura uložení speciálních axiómů známých teorií



Obrázek 37: Sekvenční diagram vložení speciálních axiómů do množiny formulí

5.11 Specifikace struktury XML souborů

Úplně poslední věcí, na kterou se pří analýze systému zaměříme, je uložení a načtení celé struktury (tzn. formule, universum a interpretační struktura) do, respektive z, formátu XML. Vzhledem k tomu, že slovní popis takovéto struktury je problematický, počítá se s tím, že čtenář k plnému porozumění struktuře využije přiložené výpisy 13, 14 a soubor *FOLModeler.xsd*, nacházející se na přiloženém CD, který formálně celou strukturu formálně definuje.

Kořenovým elementem celého XML souboru je ***folmodeler*** obsazující tři atributy. Prvním je atribut ***version***, který má vždy hodnotu „2.0“. Dále atribut ***taskType*** s hodnotami *InterpretationStructures* nebo *RelationStructures*, v závislosti na prostředí. Posledním atributem je jmenný prostor ***xmlns*** nutný pro ověření struktury souboru pomocí technologie XSD. Element folmodeler může mít celkem tři potomky:

- **formulaSet** - jednotlivé formule množiny
- **interpretations** - interpretace predikátových a funkčních symbolů
- **universum** - individua se všemi vlastnostmi. V prostředí pro vytváření IS množiny formulí má tento element ještě tři atributy - *sizex*, *sizey* a *sizez* určující rozměry vykreslené 3D mřížky

Potomky elementu ***formulaSet*** budou pouze elementy ***formula*** obsahující formuli ve formátu se zástupnými symboly. Například formule $\forall x P(f(x))$ bude uložena takto:

<formula>VxP(f(x))</formula>

Potomky elementu ***interpretations*** budou čtyři typy elementů, každý dle patřičného symbolu podléhajícího interpretaci:

- **unaryPredicate** a **binaryPredicate** - atribut *value* s názvem dané relace
- **unaryFunction** - s potomky <map from="a"to="b"/> reprezentující řádky funkční tabulky a $\rightarrow$ b
- **constant** - atribut *entityID* označující interpretované individuum

Každý z těchto elementů bude mít navíc ještě atribut ***name*** s názvem daného symbolu. Poslední základní element ***universum*** má poněkud komplikovanější strukturu. V prostředí pro vytváření interpretačních struktur množiny formulí může obsahovat elementy typu:

<entity size="big" shape="pyramid"color="yellow" x="5" y="4" z="2" id="3" />

kde *size*, *shape*, *color*, ... jsou jednotlivé vlastnosti prostorových tvarů. V prostředí pro modelování vlastností relačních struktur element universum obsahuje: <nodes>

```
    <node id="0"position="1740,1860"/>
    ...
</nodes>
<edges>
    <edge from="3"to="1" start="1890,2010"end="1890,1870"curvePoint="1890,1935"/>
    ...
</edges>
```

kde elementy ***nodes*** a ***edges*** představují jednotlivé hrany či vrcholy orientovaného grafu. Tímto bychom měli mít alespoň hrubou představu o struktuře XML souborů systému. Samotnému způsobu převodu objektové struktury do XML souborů se v této práci věnovat nebudeme, jelikož se stejně jako v předchozích případech jedná o postup silně závislý na zvoleném prostředí pro implementaci.

6 Závěr

Cílem této práce bylo vytvoření návrhu a implementace systému v jazyce C# pro podporu výuky předmětu Matematická logika. Navržený systém měl obsahovat dvě prostředí. Grafické prostředí pro vytváření interpretační struktury množiny formulí PL1 a prostředí pro modelování důležitých vlastností relací.

Můžeme říci, že cíl této práce byl splněn a studenti mají nyní k dispozici nástroj, který by jim měl usnadnit pochopení základů predikátové logiky prvního řádu. Díky prostředí pro modelování důležitých vlastností relací, může tento nástroj nalézt uplatnění i při výuce jiných předmětů.

V rámci budoucího vývoje systému se nabízí hned několik oblastí, na které bychom se mohli zaměřit. Bylo by například vhodné doplnit funkce pro nalezení modelu množiny formulí. Tento problém je pro konečná universa algoritmicky řešitelný, nicméně řešení „hrubou silou“ není zcela praktické, jelikož možnosti, jak v našem systému sestavit universum, se teoreticky pohybují ve stovkách miliard kombinací. Dále by bylo například vhodné doplnit nástroj o prostředí pro výuku obecné rezoluční metody. V neposlední řadě se také nabízí napojení na e-learningový systém eLogika.

7 Literatura

[1] BARKER-PLUMMER, Dave; BARWISE, Jon; ETCHEMENDY, John. Tarski's World : Revised and Expanded : University of Chicago press, 2007. 144 s. ISBN 1575864843.

[2] *JANČAR, Petr.:* **Teoretická informatika,** skriptum VŠB-TU Ostrava, 2010

[3] *DUŽÍ, Marie:* **Matematická logika,** skriptum VŠB-TU Ostrava, 2003

[4] *MATĚJIČNÝ, Vladimír:* **3D vizualizace modelů v PL1,** bakalářská práce VŠB-TU Ostrava, 2010

[5] **Parciální funkce** [online]. Dostupné z URL: <http://www.cojeco.cz/index.php?id_desc=70539> [cit. 2013-07-05]

[6] *VONDRÁK, Ivo:* **Úvod do softwarového inženýrství**, skriptum VŠB-TU Ostrava, 2002

[7] *BUCHALCEVOVÁ, Alena; PAVLÍČKOVÁ, Jarmila; PAVLÍČEK, Luboš:* **Základy softwarového inženýrství - materiály ke cvičení. 1.vyd.**, Vysoká škola ekonomická Praha, 2007

[8] **Use Case** [online]. Dostupné z URL: <http://en.wikipedia.org/wiki/Use_case>. [cit. 2013-07-05]

[9] **UML basics: The sequence diagram** [online]. Dostupné z URL: <http://www.ibm.com/developerworks/rational/library/3101.html> [cit. 2013-07-05]

[10] **UML basics: The class diagram** [online]. Dostupné z URL: <http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/>. [cit. 2013-07-05]

[11] **UML basics: The component diagram** [online]. Dostupné z URL: <http://www.ibm.com/developerworks/rational/library/dec04/bell/> [cit. 2013-07-05]

[12] **ANTLR - ANother Tool for Language Recognition** [online]. Dostupné z URL: <http://www.antlr.org/> [cit. 2013-07-05]

[13] **Depth-first search** [online]. Dostupné z URL: < http://en.wikipedia.org/wiki/Depth-first_search> [cit. 2013-07-05]

[14] **Bézier curve** [online]. Dostupné z URL: < http://en.wikipedia.org/wiki/B%C3%A9zier_curve> [cit. 2013-07-05]

Přílohy

A Specifikace základních případů užití

01 - Vložení formule do množiny	
Cíl případu:	Systém vloží formuli zadanou uživatelem do množiny formulí
Vstupní podmínky:	-
Výstupní podmínky:	Systém vložil novou formuli do množiny formulí
Scénář případu:	1. Systém uživateli zobrazí pole pro textový vstup 2. Uživatel do textového pole zadá formuli PL1 ve formátu se zástupnými symboly (viz tabulka 1) a potvrdí vstup stisknutím klávesy „Enter“ nebo kliknutím na tlačítko přidat 3. Systém provede kontrolu zda se jedná o správně utvořenou formuli PL1 vyhovující omezením 4. DOKUD se nejedná o správně utvořenou formuli PL1 nebo formule nevyhovuje omezením: (a) Systém žádá uživatele o napravení chyby a zároveň zobrazí, na které pozici se chyba nachází (b) Uživatel provede opravu a potvrdí vstup stisknutím klávesy „Enter“ nebo kliknutím na tlačítko přidat (c) Systém provede kontrolu zda se jedná o správně utvořenou formuli PL1 vyhovující omezením 5. Systém provede kontrolu, zda množina formulí již neobsahuje vkládanou formuli (a) Při neúspěšné kontrole systém zobrazí chybovou zprávu a scénář užití se vrací do výchozího bodu 1 6. Systém provede kontrolu konzistence arity predikátových a funkčních symbolů (a) Při neúspěšné kontrole systém zobrazí chybovou zprávu a scénář užití se vrací do výchozího bodu 1 7. Systém vloží novou formuli do množiny formulí

Tabulka 5: Specifikace případu užití „01 - Vložení formule do množiny“

02 - Import formulí	
Cíl případu:	Systém vloží jednu či více formulí do množiny formulí
Vstupní podmínky:	-
Výstupní podmínky:	Systém vložil či nevložil formule do množiny formulí
Scénář případu:	1. Systém uživateli zobrazí dialog pro import formulí 2. Uživatel vybere jeden ze vstupních formátů - standardní zástupné symboly nebo formát TEX 3. Uživatel vybere vstupní soubor 4. Systém načte importovaný soubor 5. Systém provede kontrolu, zda jsou importované formule správně utvořenými formulemi PL1 a vyhovují zadaným omezením 6. KDYŽ některá z formulí není správně utvořená nebo nevyhovuje omezení, systém zobrazí chybovou zprávu s popisem, kde a u které formule se chyba nachází (a) Uživatel provede opravu opravu chybných formulí. Scénář užití se vrací do bodu 5 7. Systém provede kontrolu, zda množina formulí neobsahuje některou z importovaných formulí (a) Při neúspěšné kontrole systém zobrazí chybovou zprávu (b) Uživatel provede opravu opravu chybných formulí. Scénář užití se vrací do bodu 5 8. Systém provede kontrolu konzistence arity predikátových a funkčních symbolů (a) Při neúspěšné kontrole systém zobrazí chybovou zprávu (b) Uživatel provede opravu opravu chybných formulí. Scénář užití se vrací do bodu 5 9. Systém vloží importované formule do množiny formulí

Tabulka 6: Specifikace případu užití „02 - Import formulí“

03 - Interpretace množiny formulí	
Cíl případu:	Kompletní interpretace všech symbolů množiny formulí
Vstupní podmínky:	1. V množině formulí musí být alespoň jedna formule 2. Neprázdné universum
Výstupní podmínky:	Uživatel interpretoval symboly podléhající interpretaci
Scénář případu:	1. Systém uživateli zobrazí nabídku interpretací symbolů 2. Uživatel pro všechny predikátové symboly vybere jednu z předdefinovaných interpretací 3. Uživatel klikne na tlačítko pro vytvoření interpretace funkčního symbolu s aritou $n = 1$ 4. Systém zobrazí universum a okno s vytvářenou interpretací funkčního symbolu 5. Uživatel dokončí interpretaci funkčního symbolu postupným výběrem jednotlivých individuí 6. Uživatel klikne na tlačítko pro výběr interpretace konstanty 7. Systém zobrazí universum 8. Uživatel vybere interpretaci konstanty kliknutím na dané individuum 9. Uživatel klikne na tlačítko pro dokončení interpretace 10. Systém uzavře nabídku pro interpretaci symbolů

Tabulka 7: Specifikace případu užití „03 - Interpretace množiny formulí“

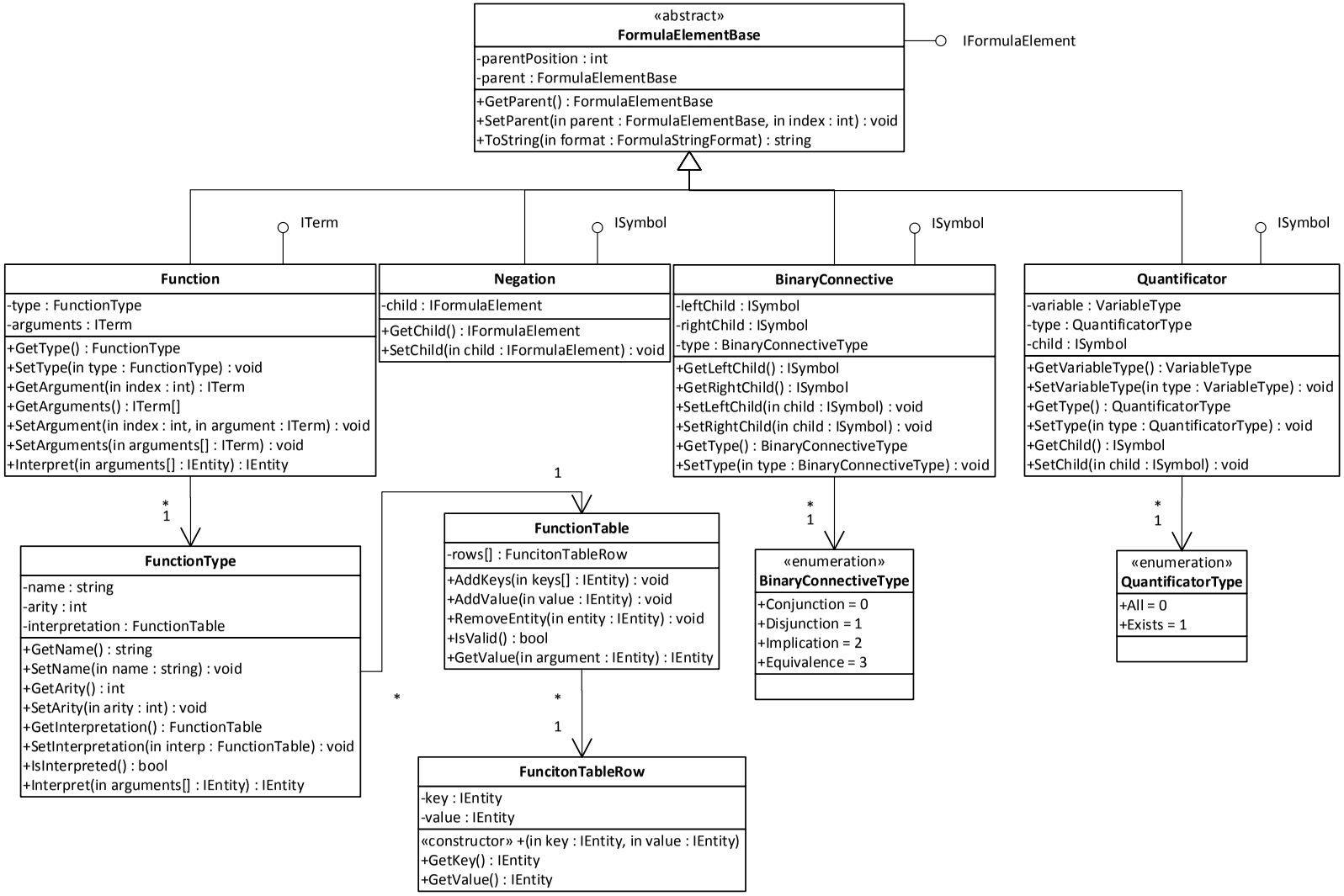
04 - Ověření modelu množiny formulí	
Cíl případu:	Ověření, zda je daná interpretační struktura modelem množiny formulí
Vstupní podmínky:	1. V množině formulí musí být alespoň jedna formule
Výstupní podmínky:	Systém vyhodnotil zda je IS modelem množiny formulí
Scénář případu:	1. Uživatel klikne na tlačítko pro ověření modelu množiny formulí 2. Systém zkontroluje zda je vytvořená interpretační struktura kompletní 3. KDYŽ je IS nekompletní systém zobrazí chybovou zprávu a celý scénář je ukončen 4. Systém provede kontrolu zda je uživatelem vytvoření IS modelem množiny formulí 5. Systém uživateli zobrazí výsledek kontroly

Tabulka 8: Specifikace případu užití „04 - Ověření modelu množiny formulí“

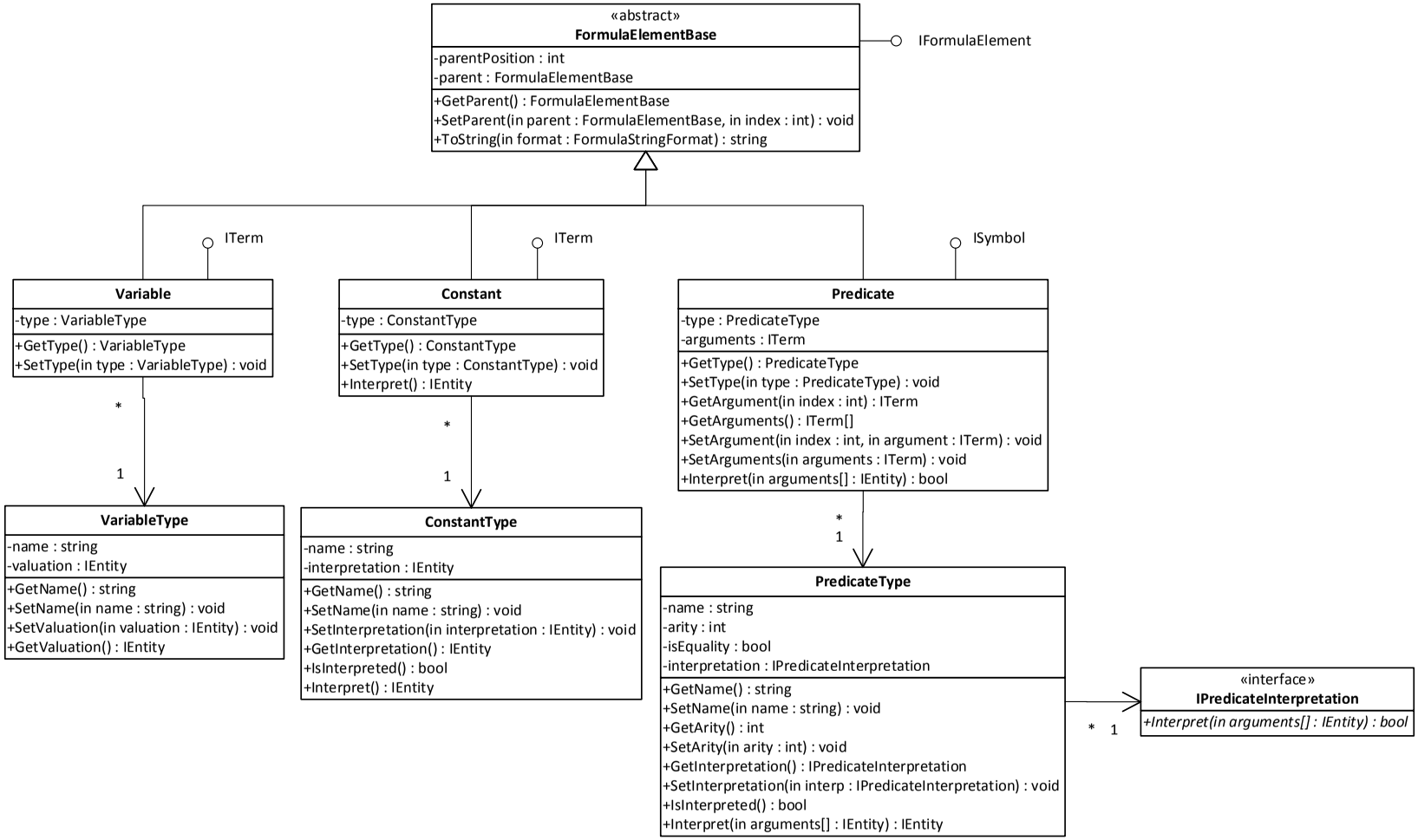
05 - Zobrazení nápovědy pro vytvoření modelu	
Cíl případu:	Systém zobrazí nápovědu pro vytvoření modelu
Vstupní podmínky:	1. V množině formulí musí být alespoň jedna formule 2. Vytvořená IS je kompletní 3. Neprázdné universum
Scénář případu:	1. Uživatel klikne na tlačítko pro zobrazení nápovědy 2. Systém zobrazí pravdivostní hodnoty jednotlivých formulí 3. Uživatel vybere jednu konkrétní formuli 4. Systém zobrazí pravdivostní hodnoty formule pro jednotlivé valuace 5. Uživatel vybere jednu konkrétní valuaci 6. Systém zobrazí: • všechny vlastnosti individuí x_i, případně y_j • výsledky funkčních zobrazení. Například pro formuli $\forall x P(f(x))$ toto bude výsledek zobrazení f pro konkrétní valuaci $e(x_i)$ • pravdivostní hodnoty jednotlivých predikátových symbolů formule • nápovědu pro binární logické spojky • pomocné věty pro formule s kvantifikátory 7. Uživatel změní interpretaci predikátového symbolu 8. Systém aktualizuje všechny dílčí nápovědy pro aktuálně vybranou formuli

Tabulka 9: Specifikace případu užití „05 - Zobrazení nápovědy pro vytvoření modelu“

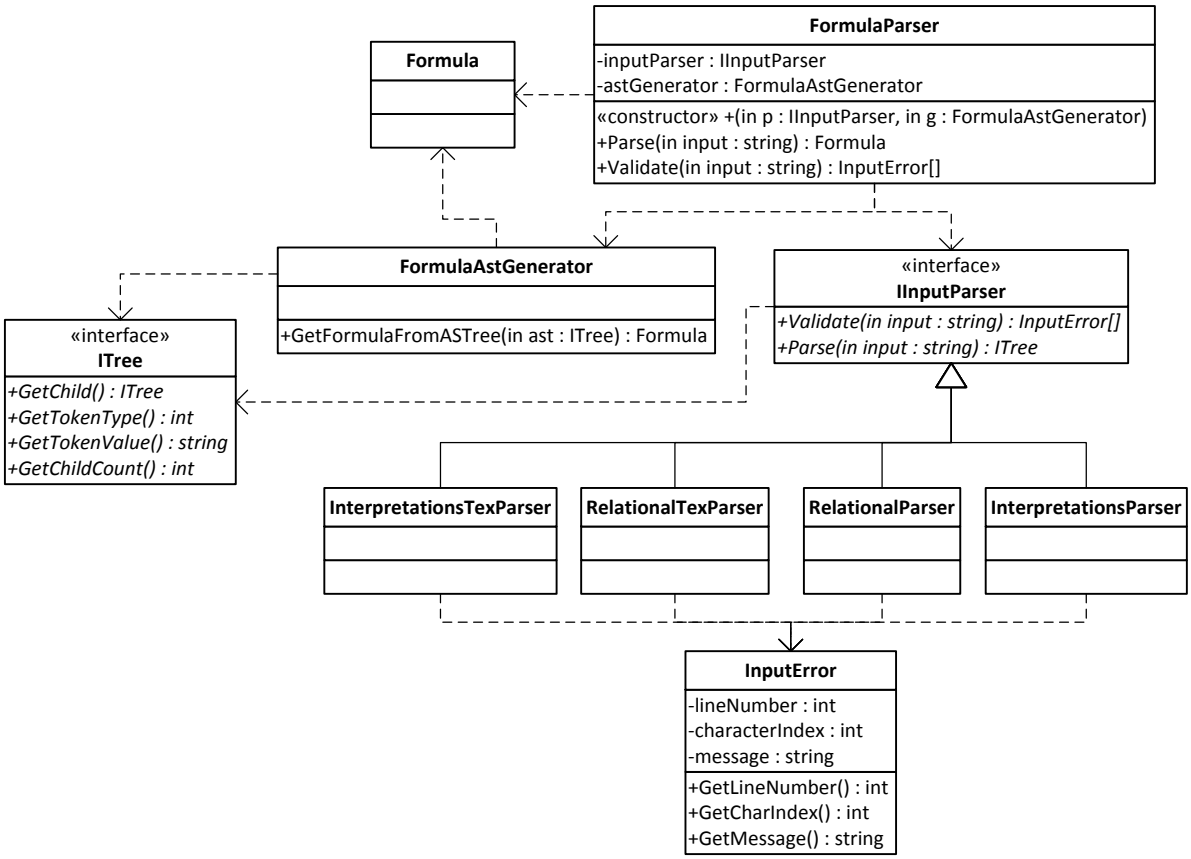
B Třídní diagramy



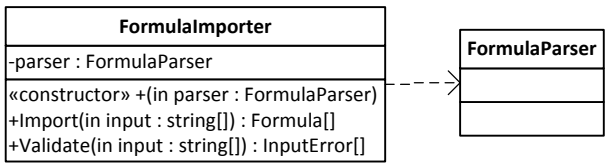
Obrázek 38: Třídní diagram formule (část 1/2)



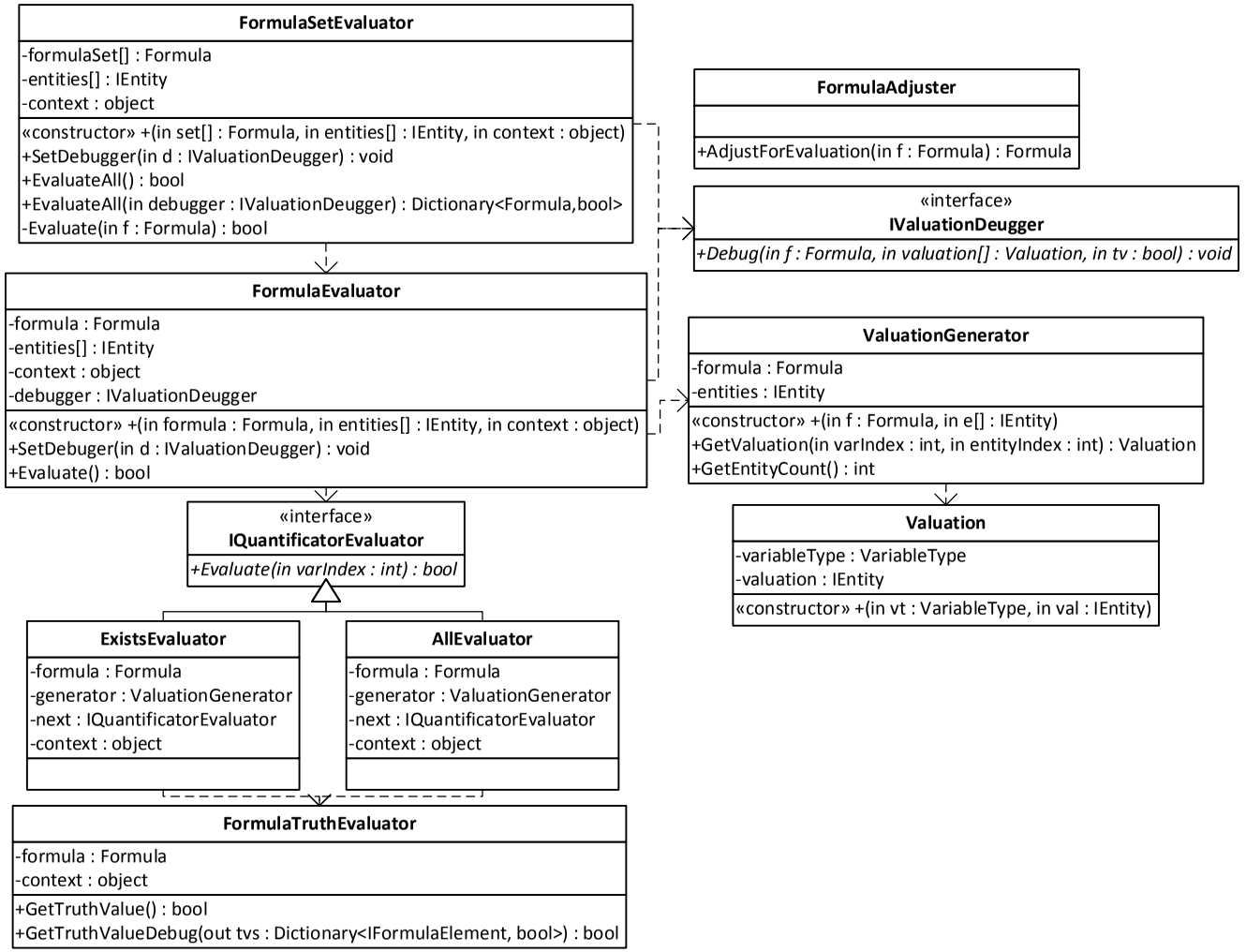
Obrázek 39: Třídní diagram formule (část 2/2)



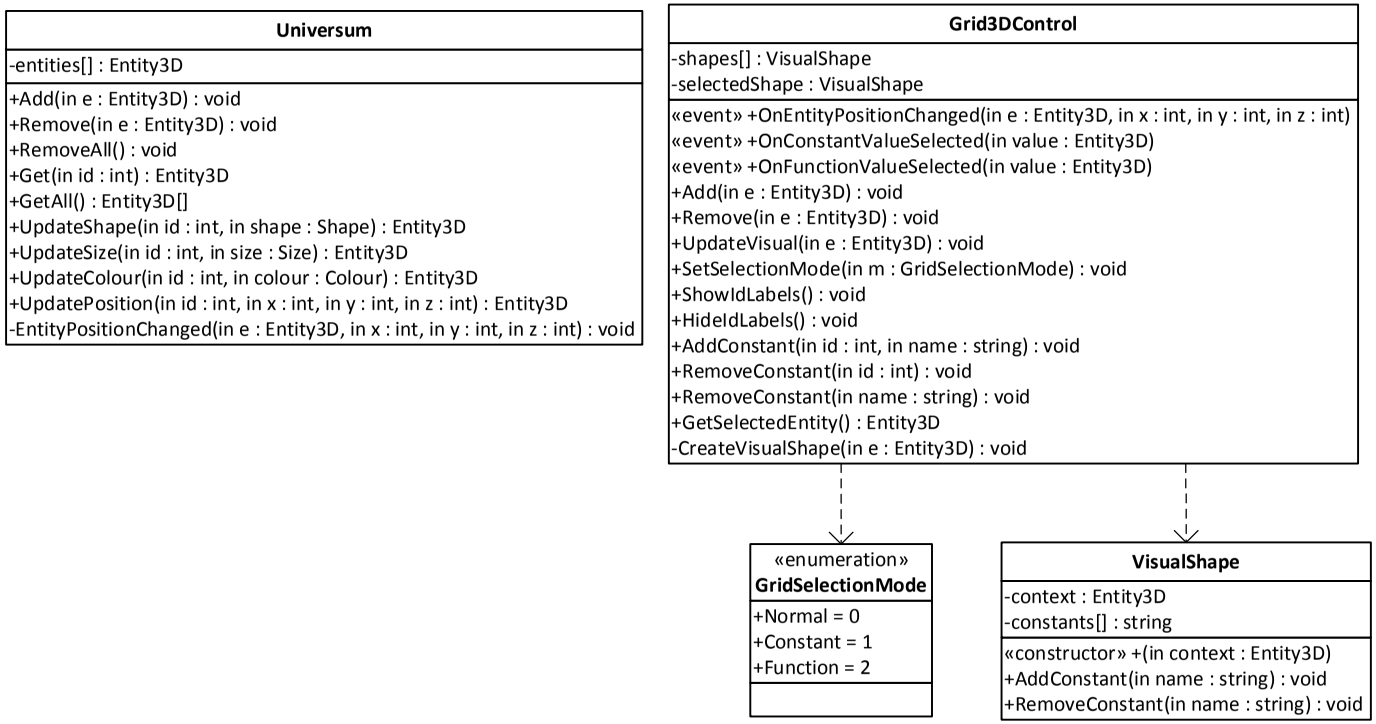
Obrázek 40: Třídní diagram zpracování uživatelského vstupu z textového řetězce



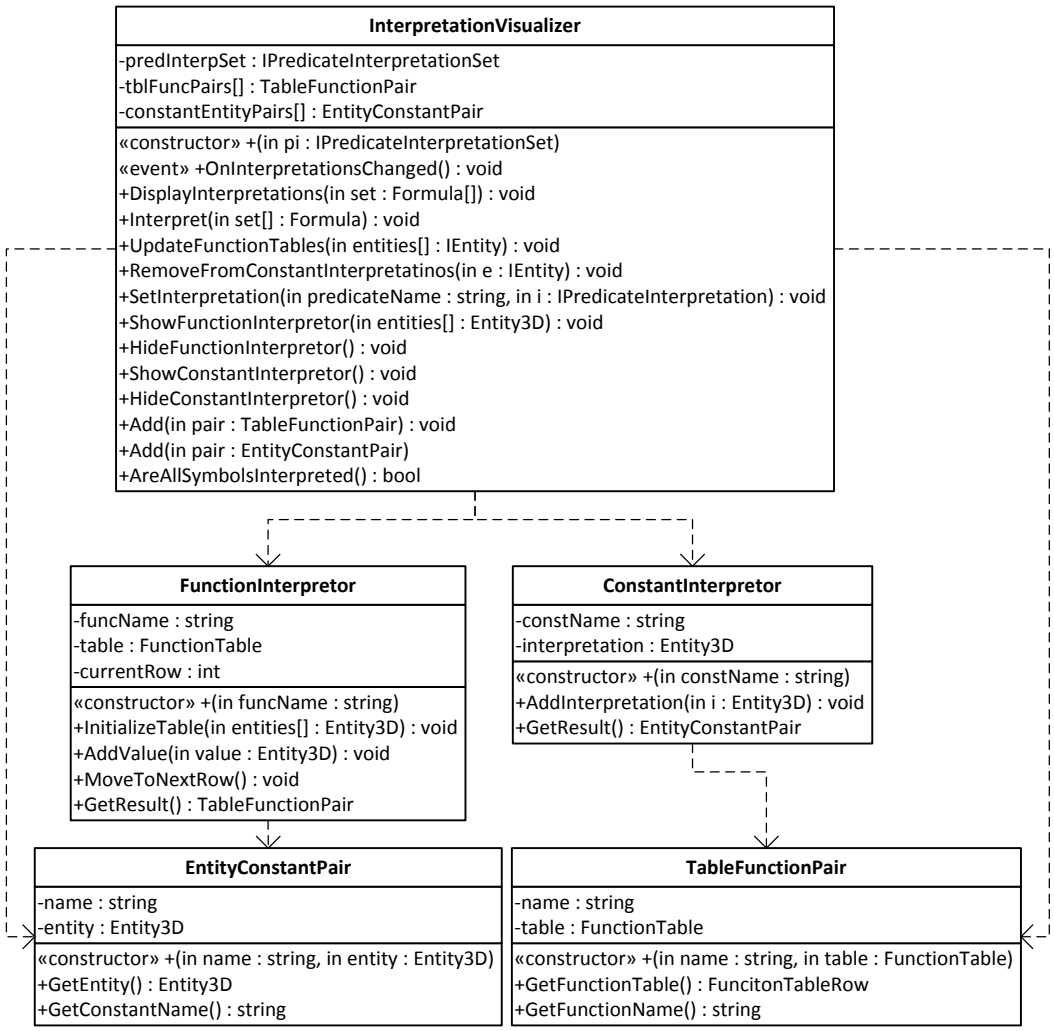
Obrázek 41: Třídní diagram importu formulí



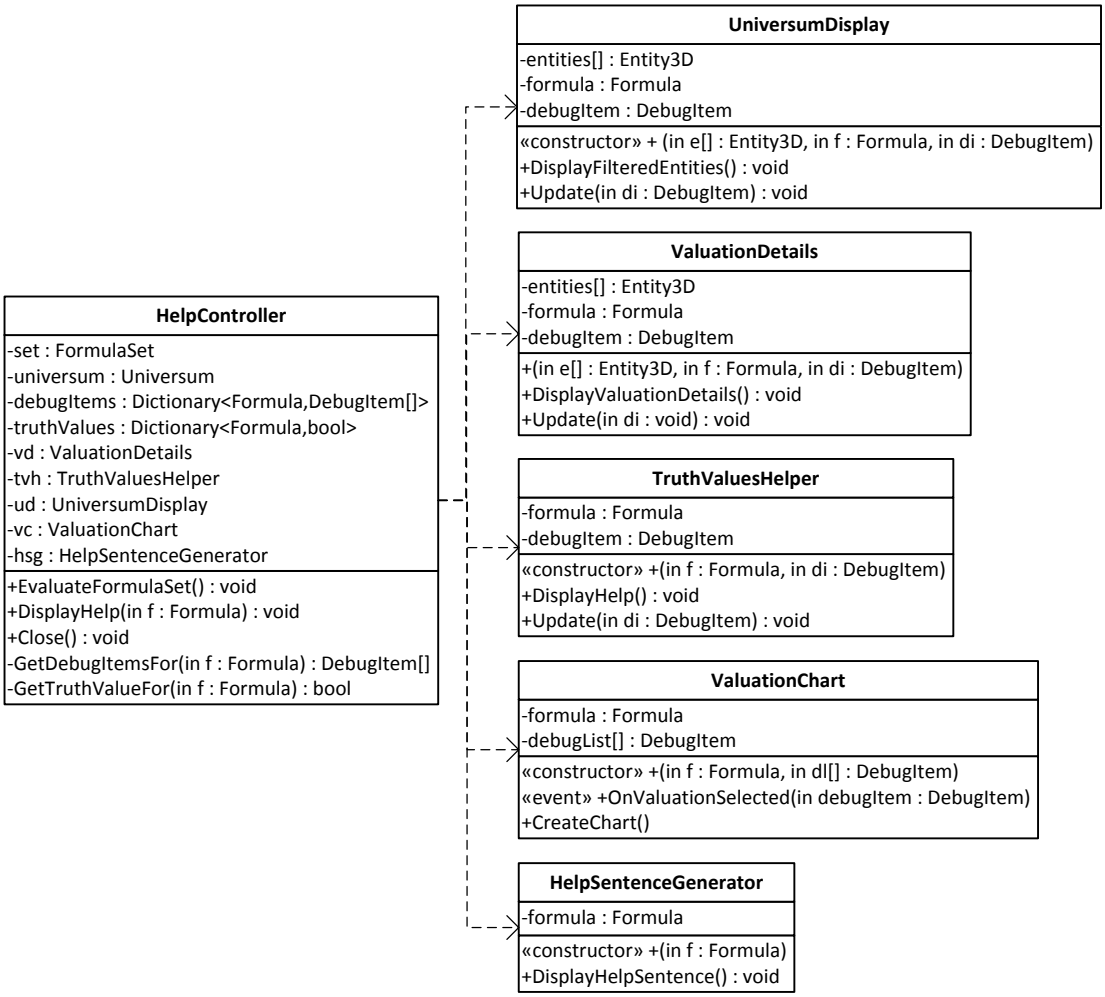
Obrázek 42: Třídní diagram ověření modelu množiny formulí



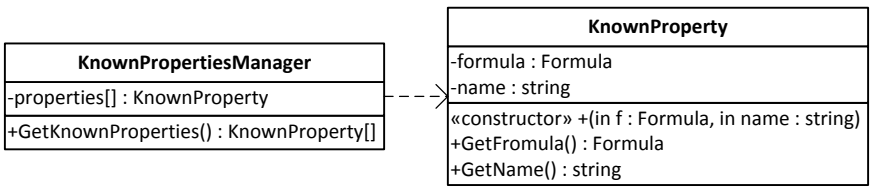
Obrázek 43: Třídní diagram universa



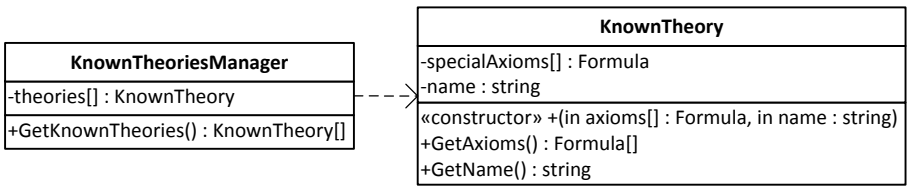
Obrázek 44: Třídní diagram interpretace symbolů



Obrázek 45: Třídní diagram nápovědy pro vytvoření modelu

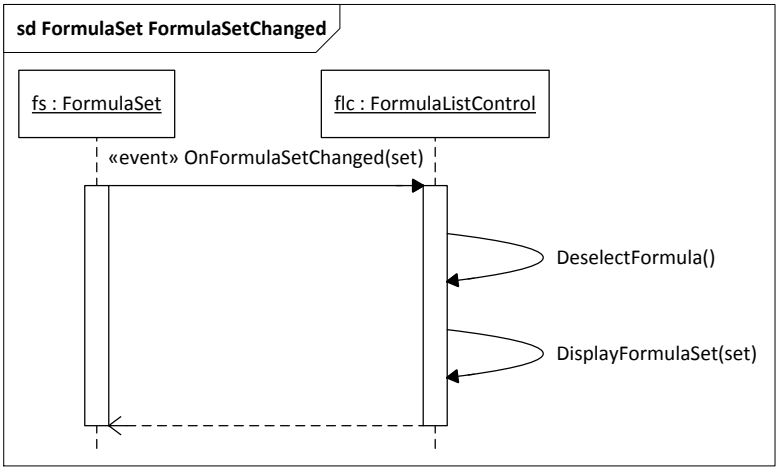


Obrázek 46: Třídní diagram známých vlastností relací

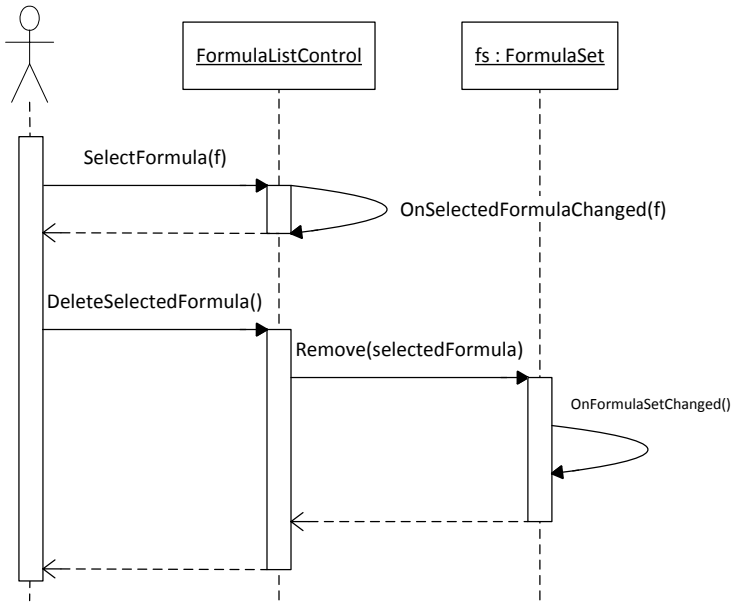


Obrázek 47: Třídní diagram speciálních axiómů známých teorií

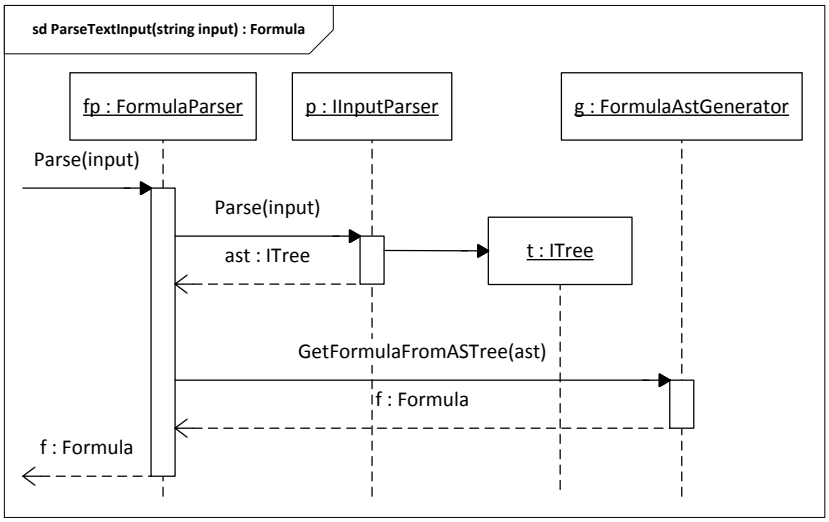
C Sekvenční diagramy



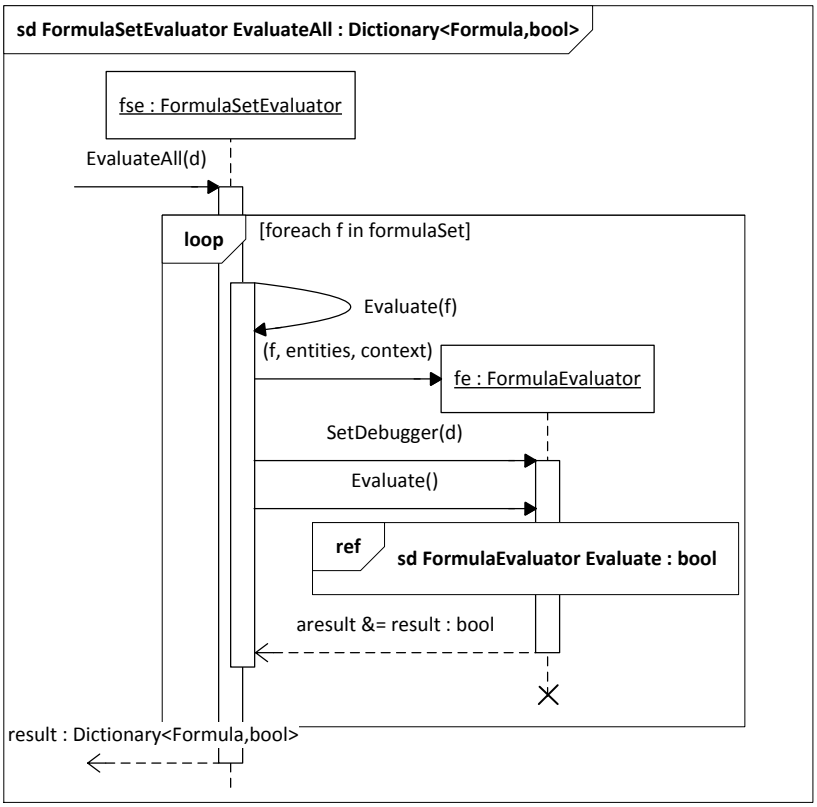
Obrázek 48: Reakce rozhraní pro zobrazení množiny formulí na změnu množiny



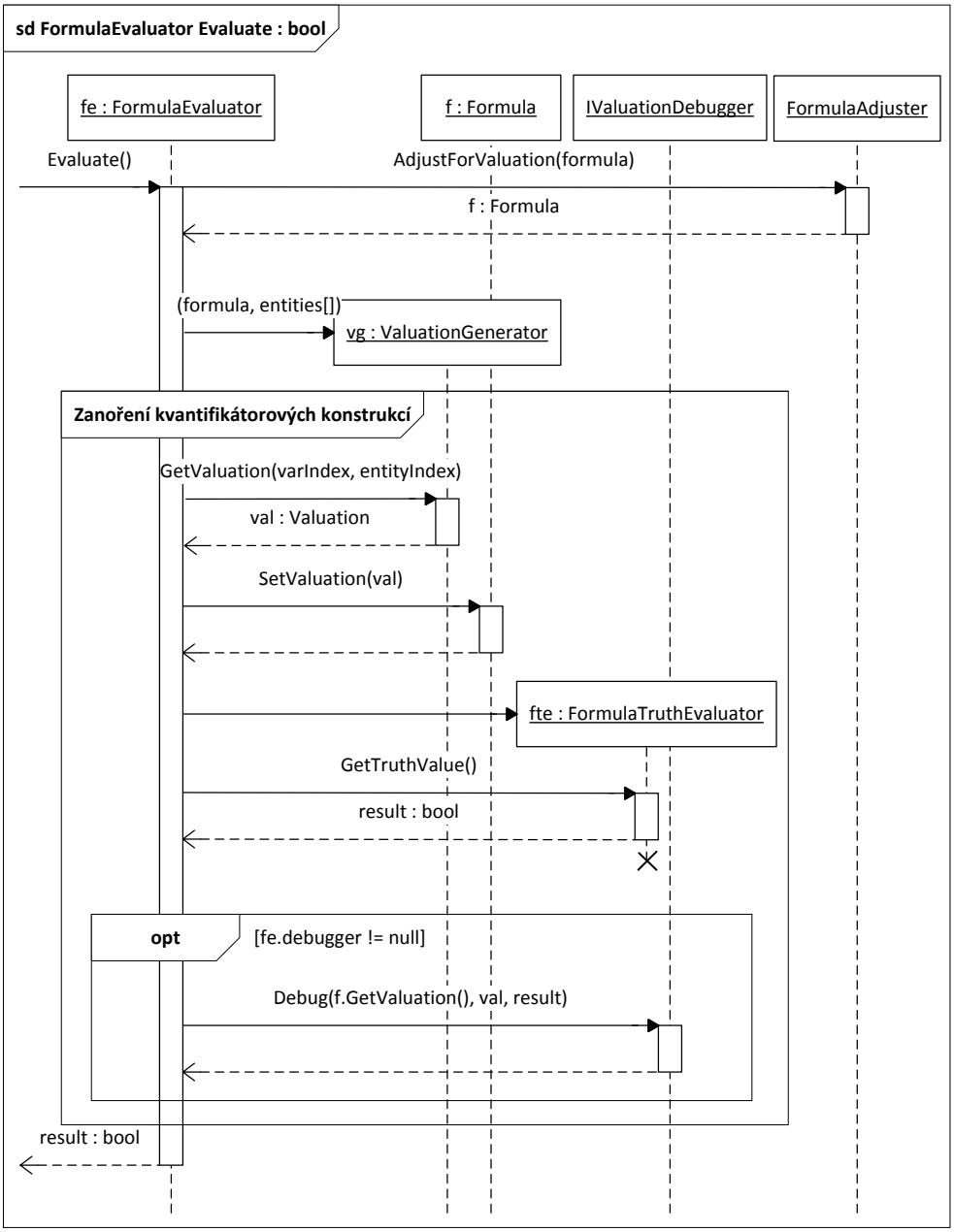
Obrázek 49: Odebrání formule z množiny



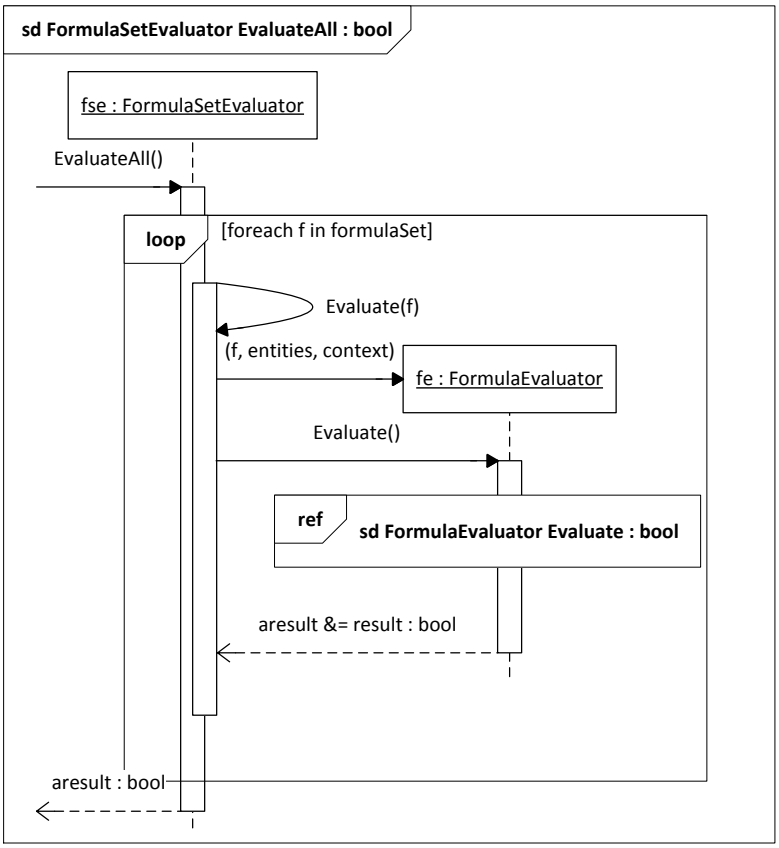
Obrázek 50: Sekvenční diagram zpracování uživatelského vstupu z textového řetězce



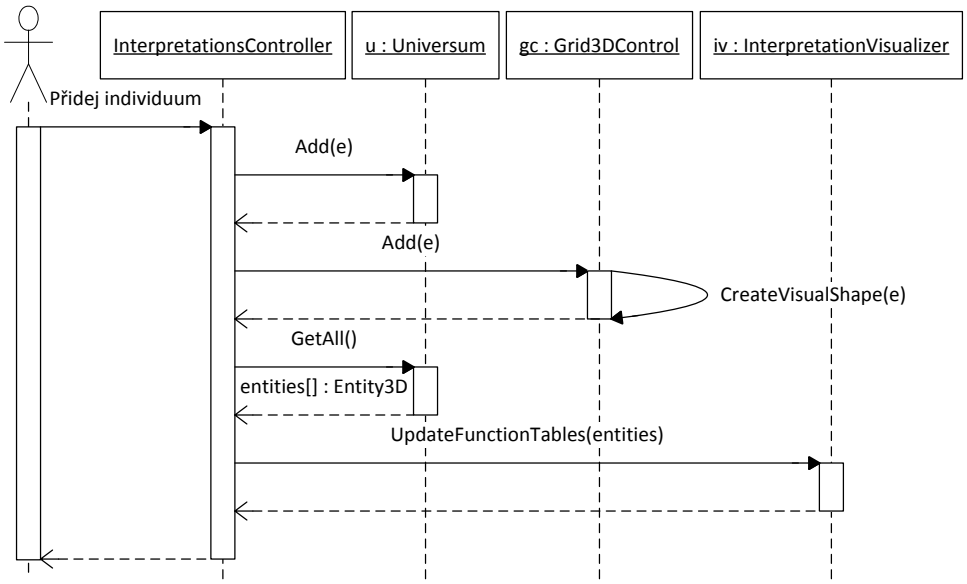
Obrázek 51: Sekvenční diagram ověření modelu množiny pro potřeby nápovědy



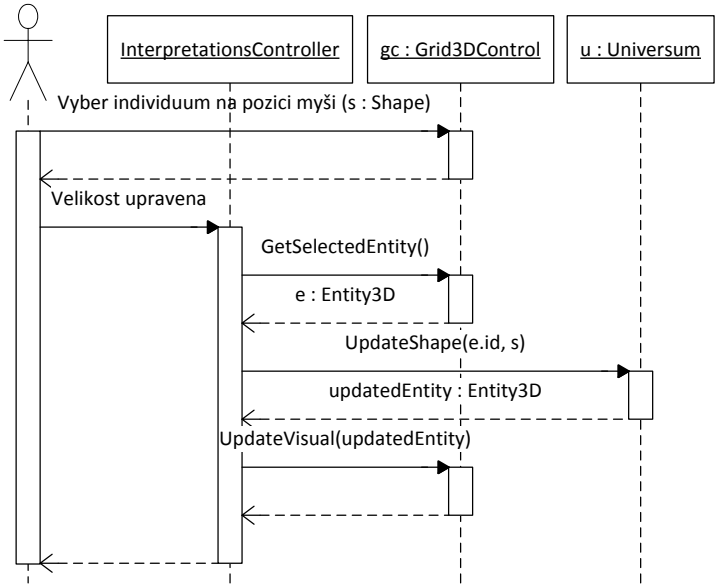
Obrázek 52: Sekvenční diagram vyhodnocení pravdivosti formule v interpretaci



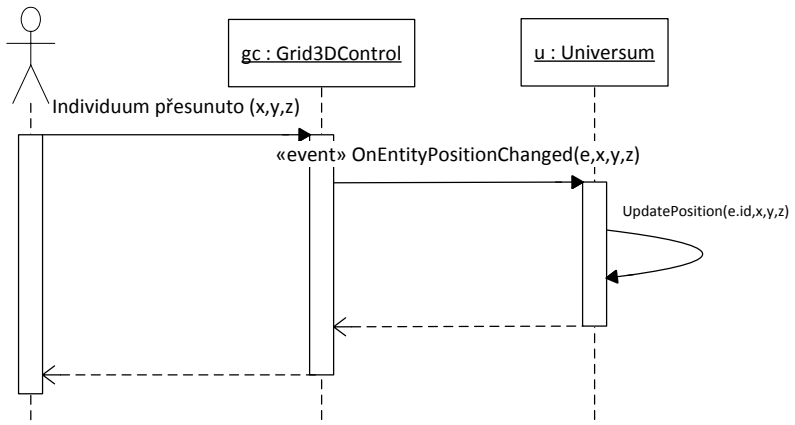
Obrázek 53: Sekvenční diagram ověření modelu množiny



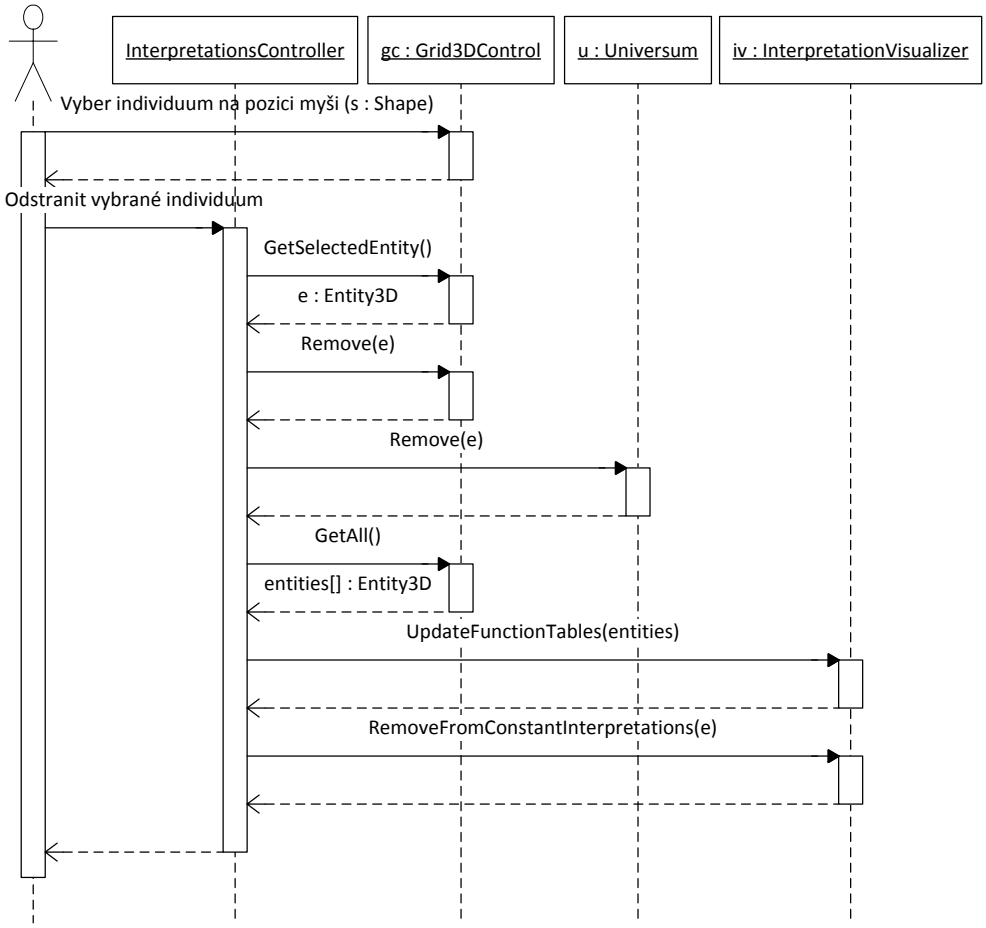
Obrázek 54: Sekvenční diagram přidání individua do universa



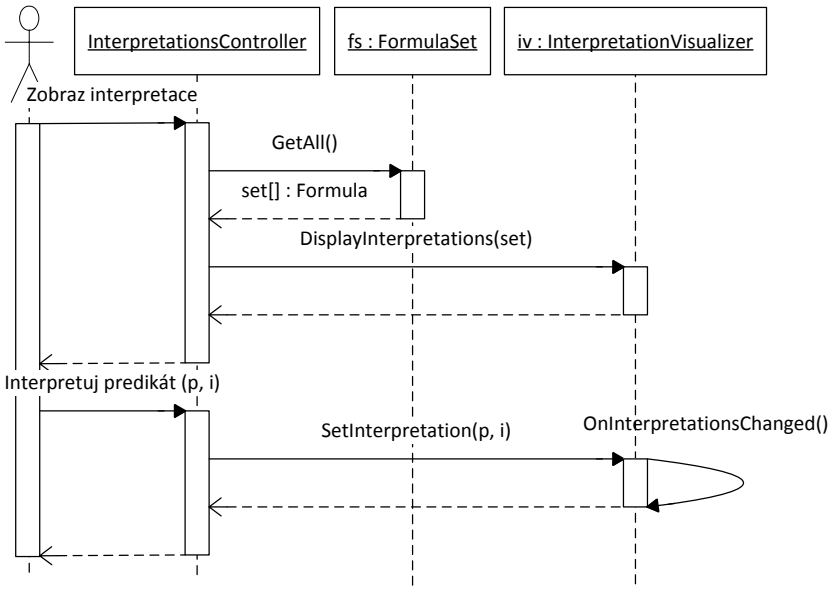
Obrázek 55: Sekvenční diagram úpravy vlastnosti individua



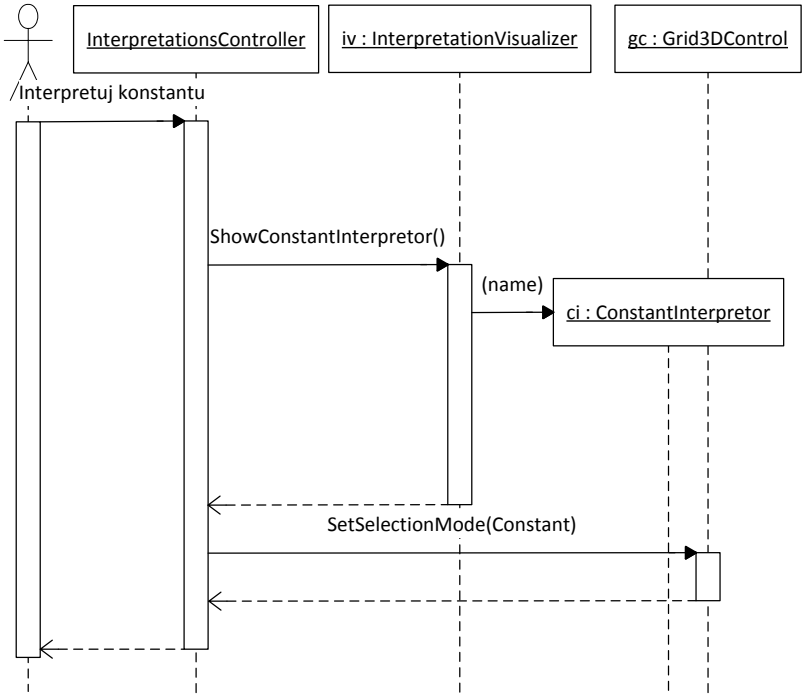
Obrázek 56: Sekvenční diagram změny pozice individua



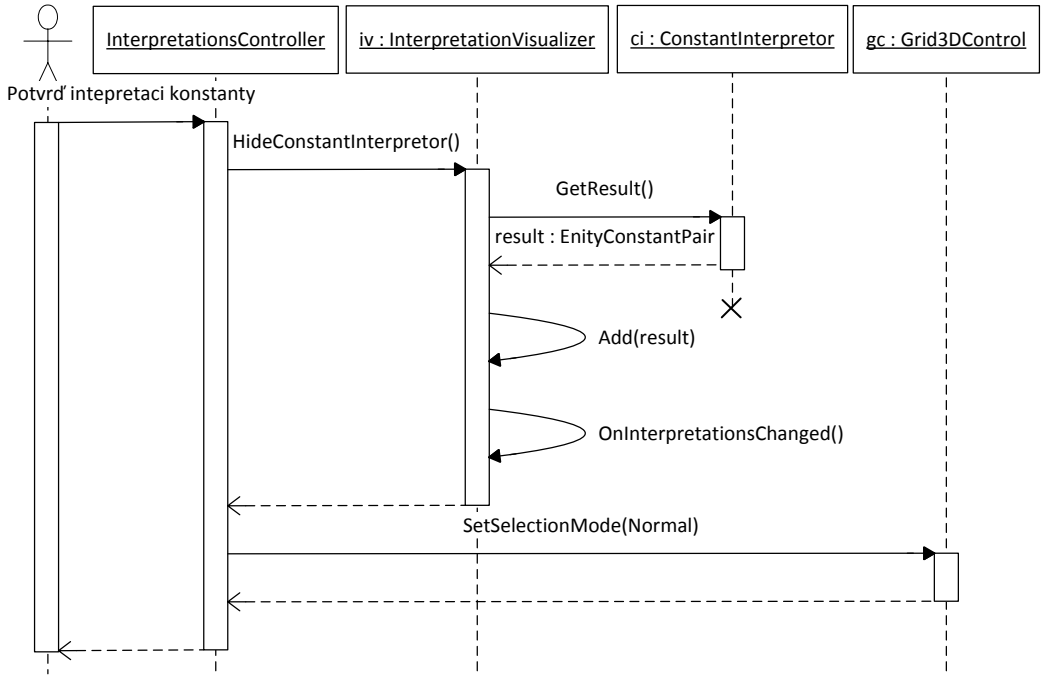
Obrázek 57: Sekvenční diagram odebrání individua z universa



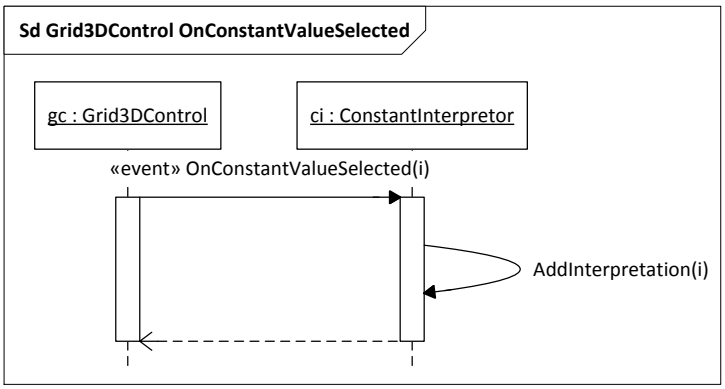
Obrázek 58: Sekvenční diagram interpretace predikátových symbolů



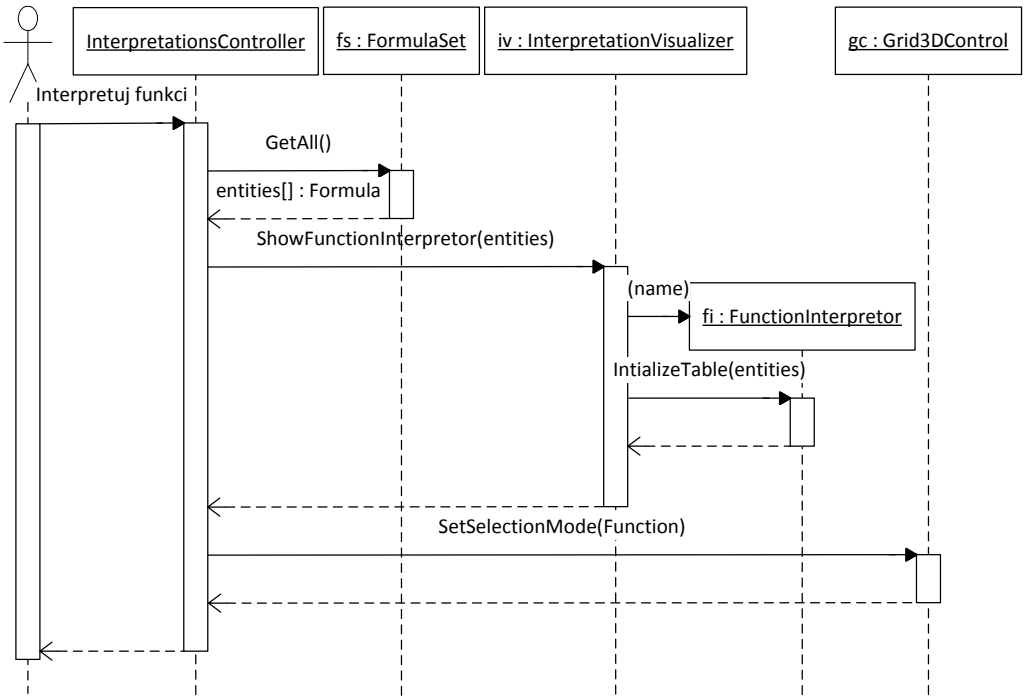
Obrázek 59: Sekvenční diagram výběru interpretace konstanty (část 1/3)



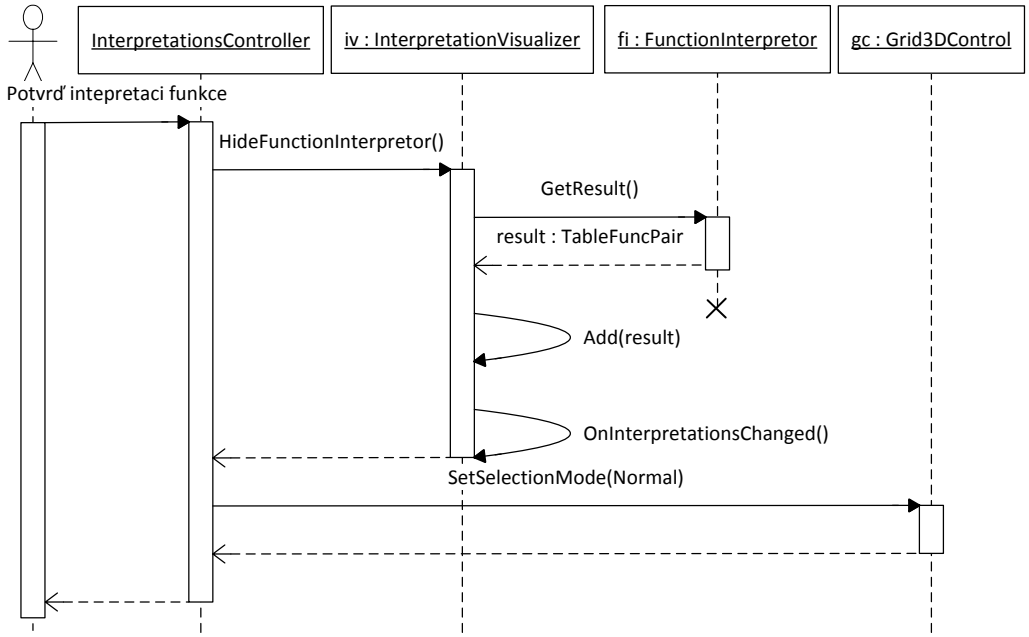
Obrázek 60: Sekvenční diagram výběru interpretace konstanty (část 2/3)



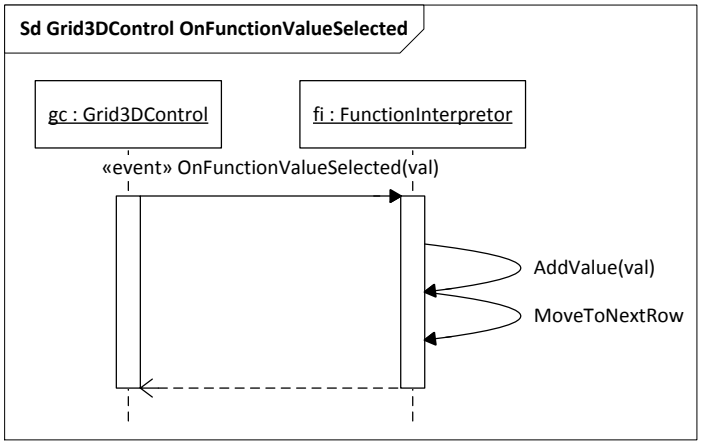
Obrázek 61: Sekvenční diagram výběru interpretace konstanty (část 3/3)



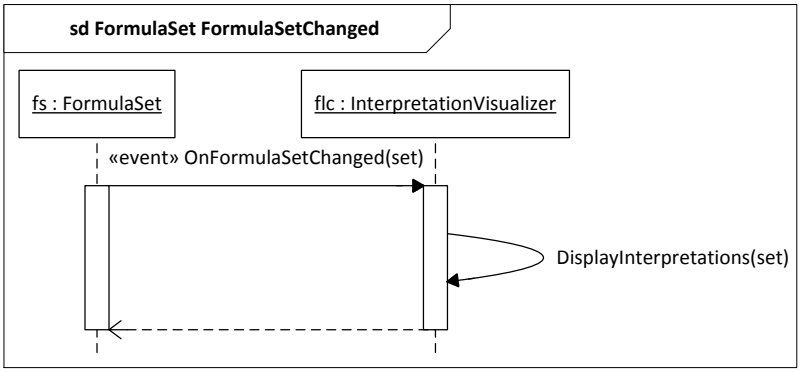
Obrázek 62: Sekvenční diagram výběru interpretace funkce (část 1/3)



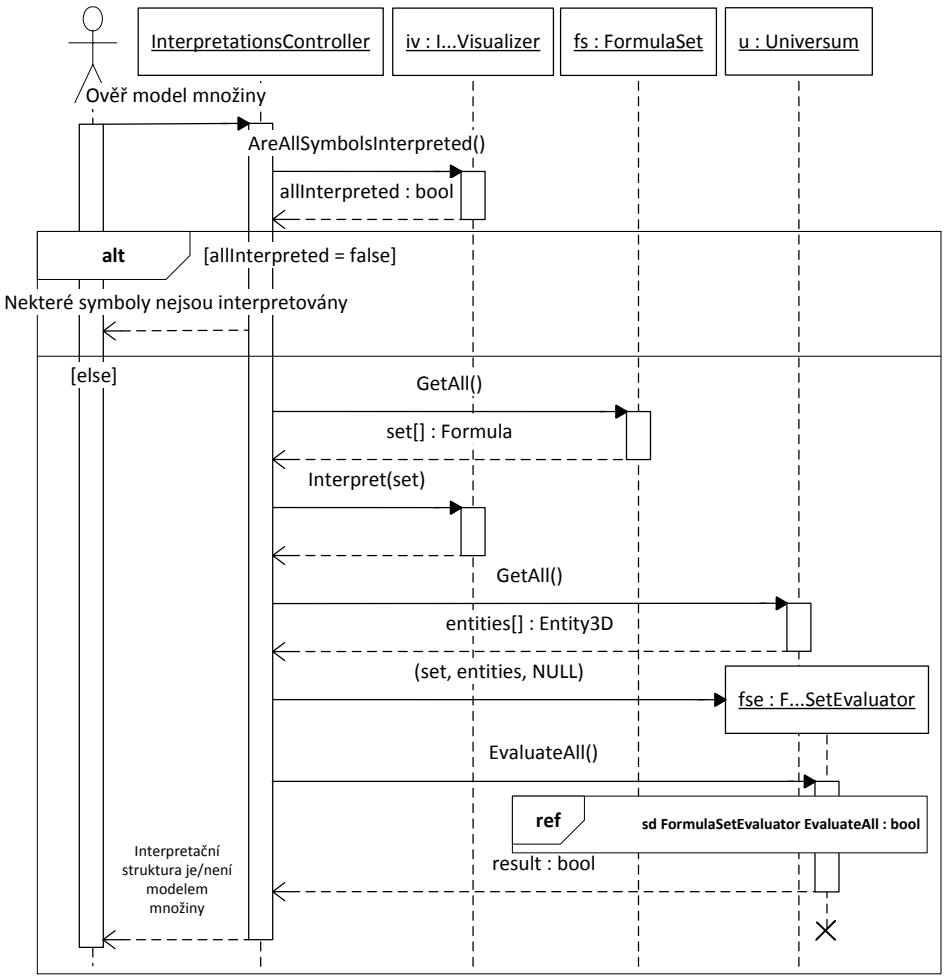
Obrázek 63: Sekvenční diagram výběru interpretace funkce (část 2/3)



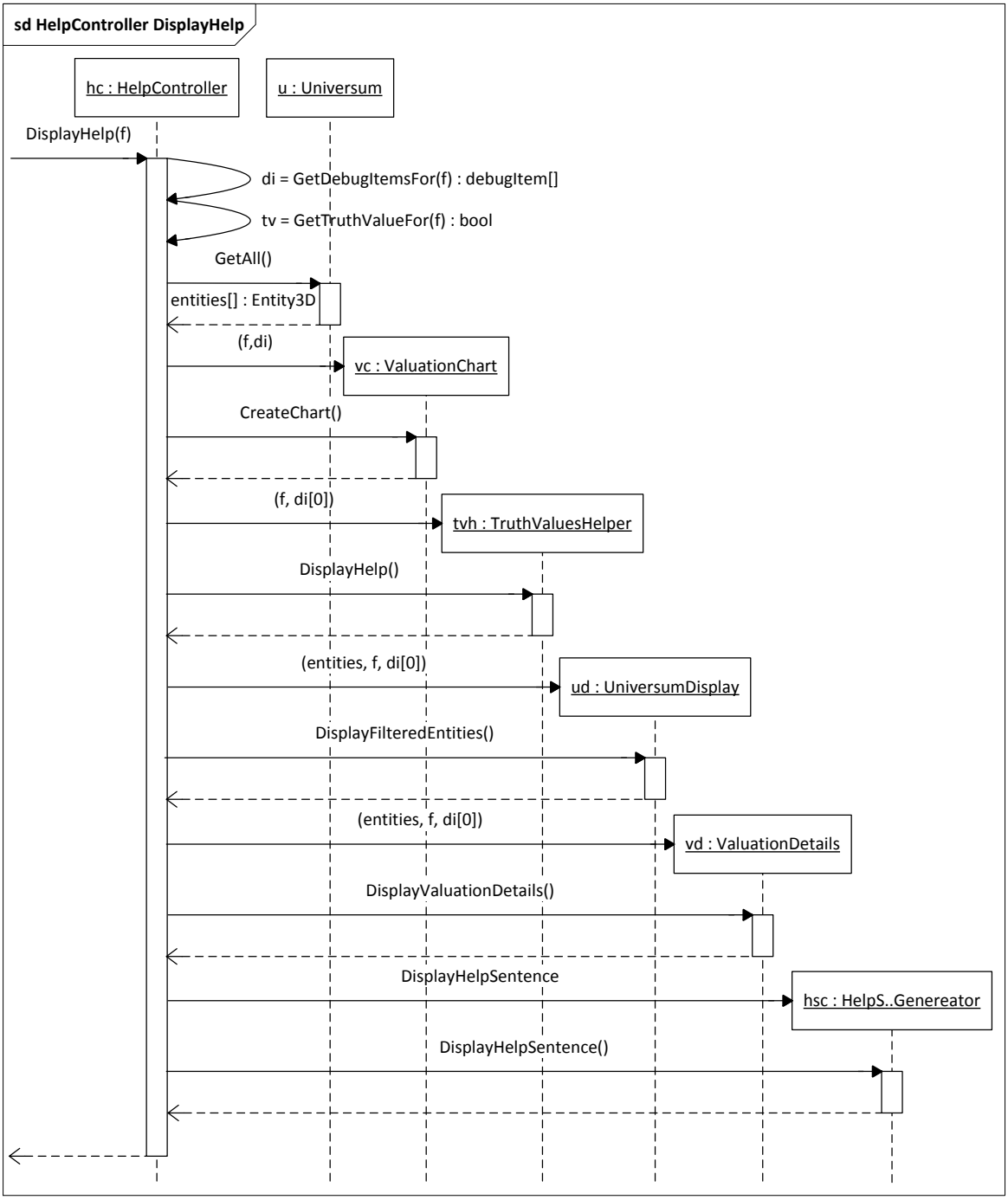
Obrázek 64: Sekvenční diagram výběru interpretace funkce (část 3/3)



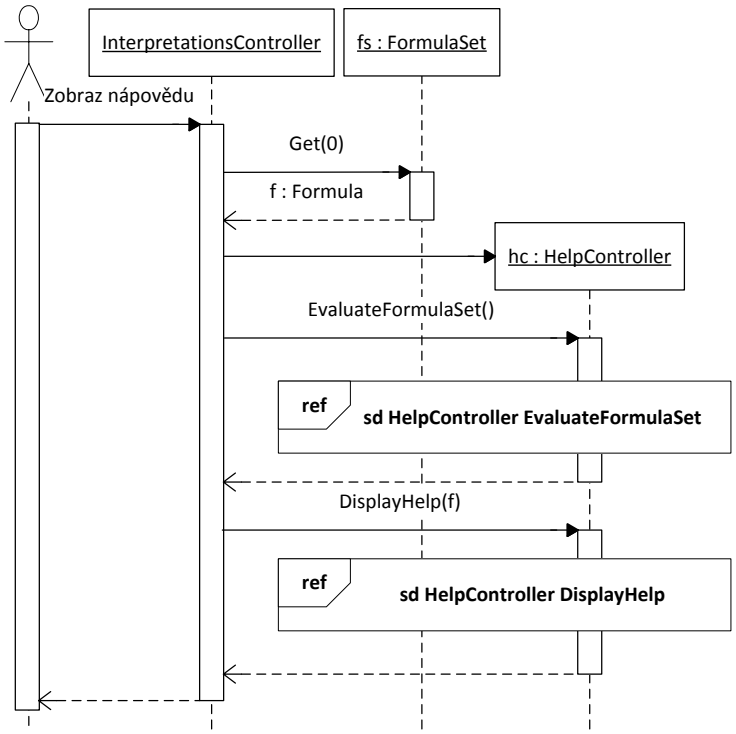
Obrázek 65: Sekvenční diagram reakce zobrazení interpretací na změnu množiny formulí



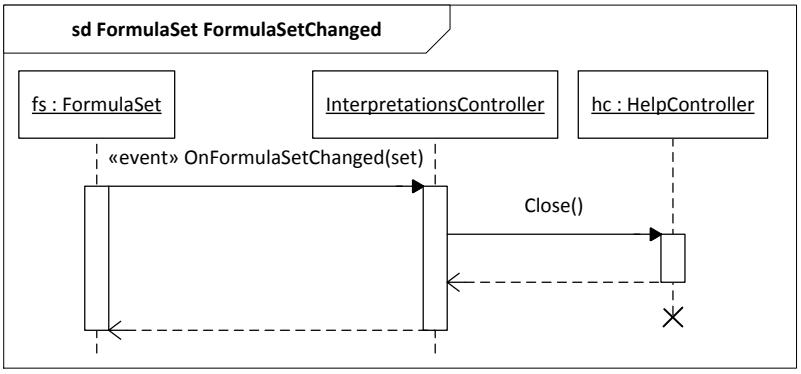
Obrázek 66: Sekvenční diagram ověření modelu množiny v prostředí pro vytváření interpretačních struktur



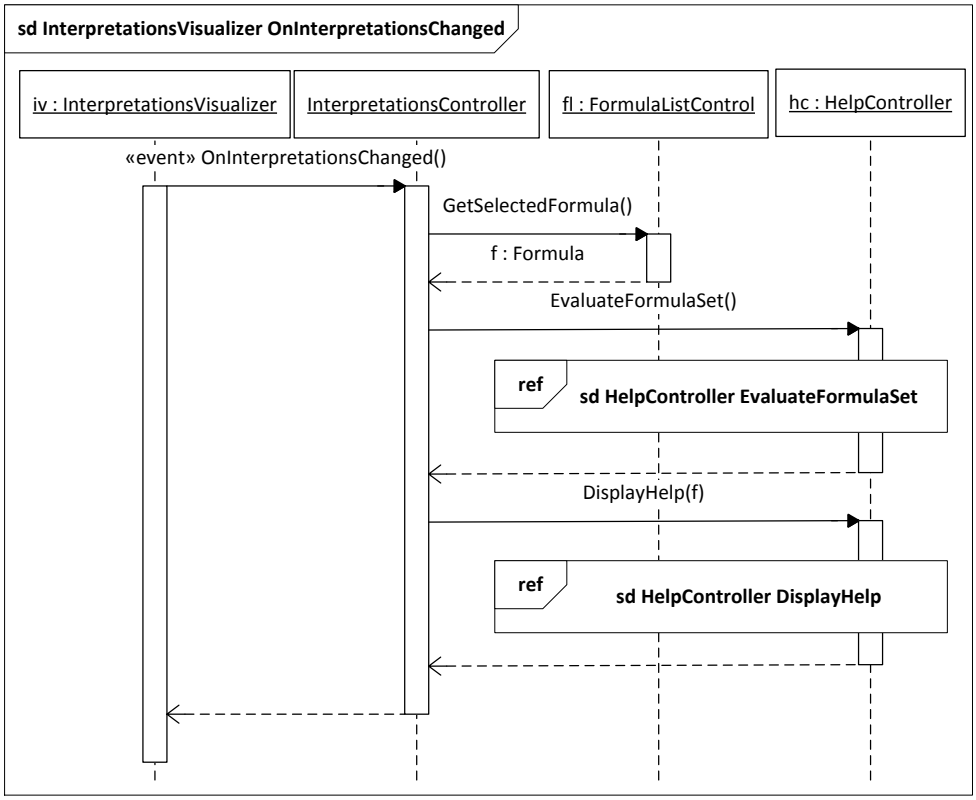
Obrázek 67: Sekvenční diagram zobrazení nápovědy pro vytvoření modelu



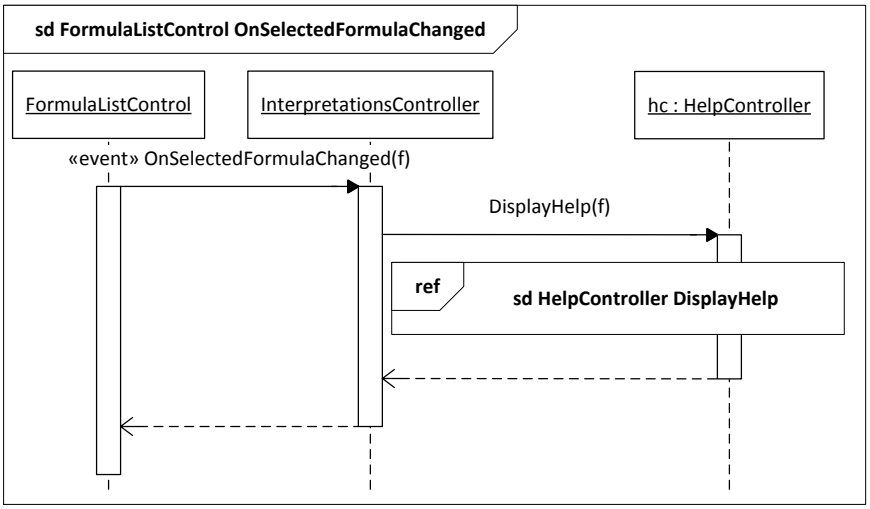
Obrázek 68: Sekvenční diagram zobrazení nápovědy pro konkrétní formuli



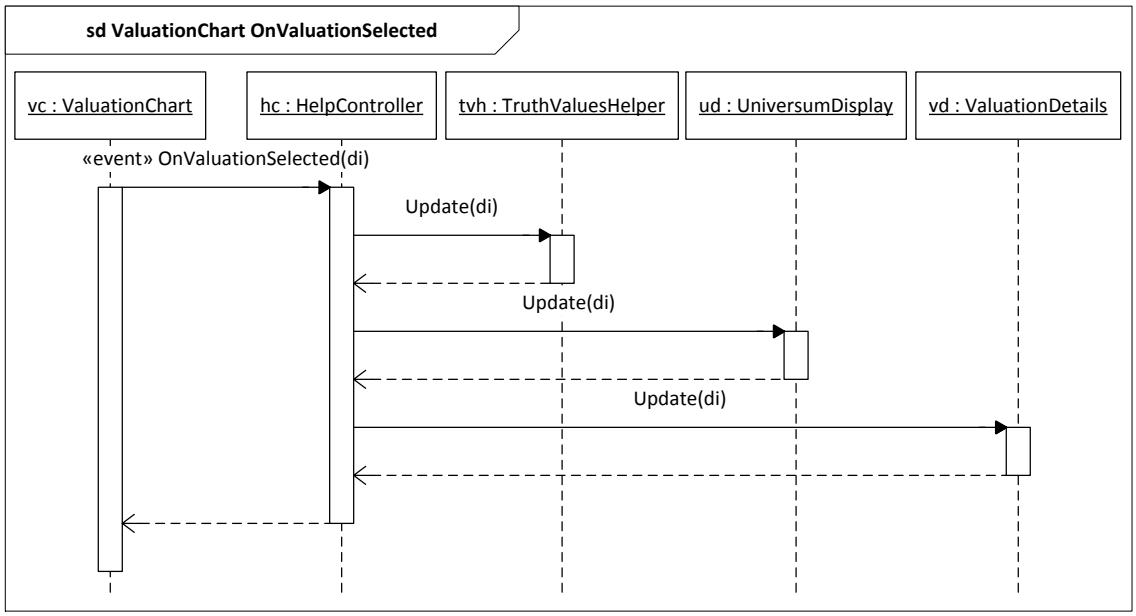
Obrázek 69: Sekvenční diagram reakce komponenty nápovědy na změnu množiny formulí



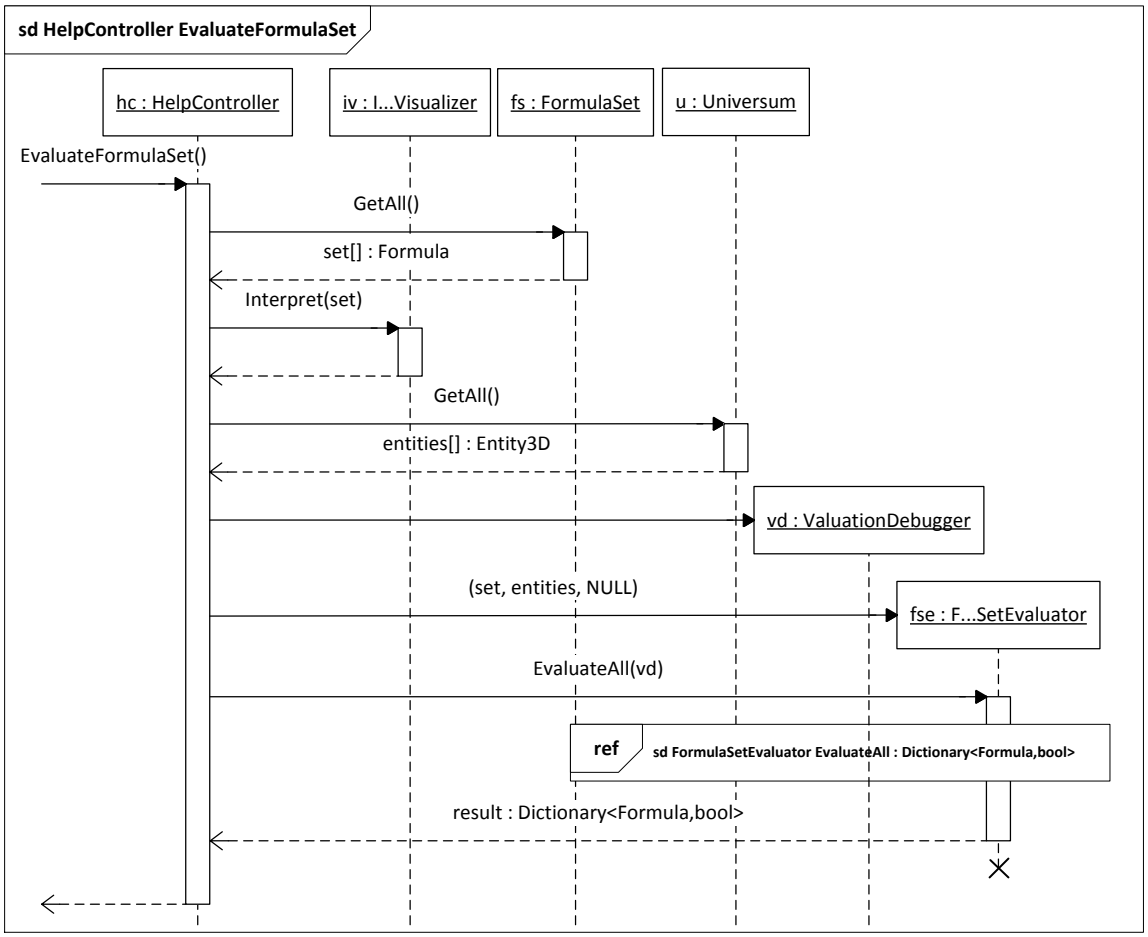
Obrázek 70: Sekvenční diagram reakce komponenty nápovědy na změnu interpretací



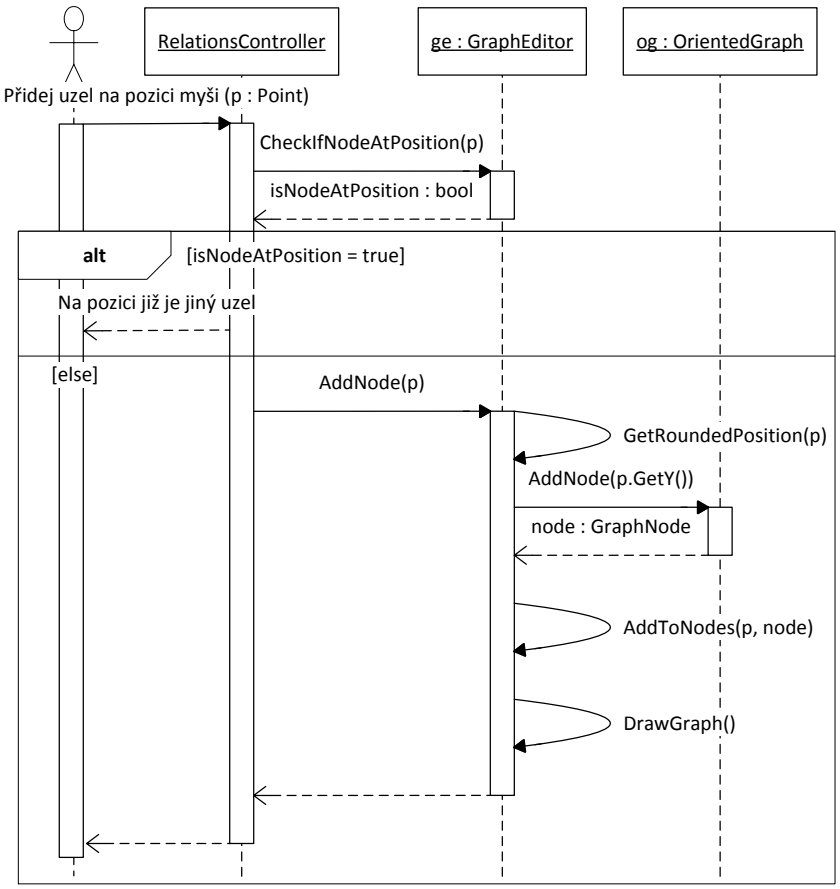
Obrázek 71: Sekvenční diagram reakce komponenty nápovědy na výběr formule



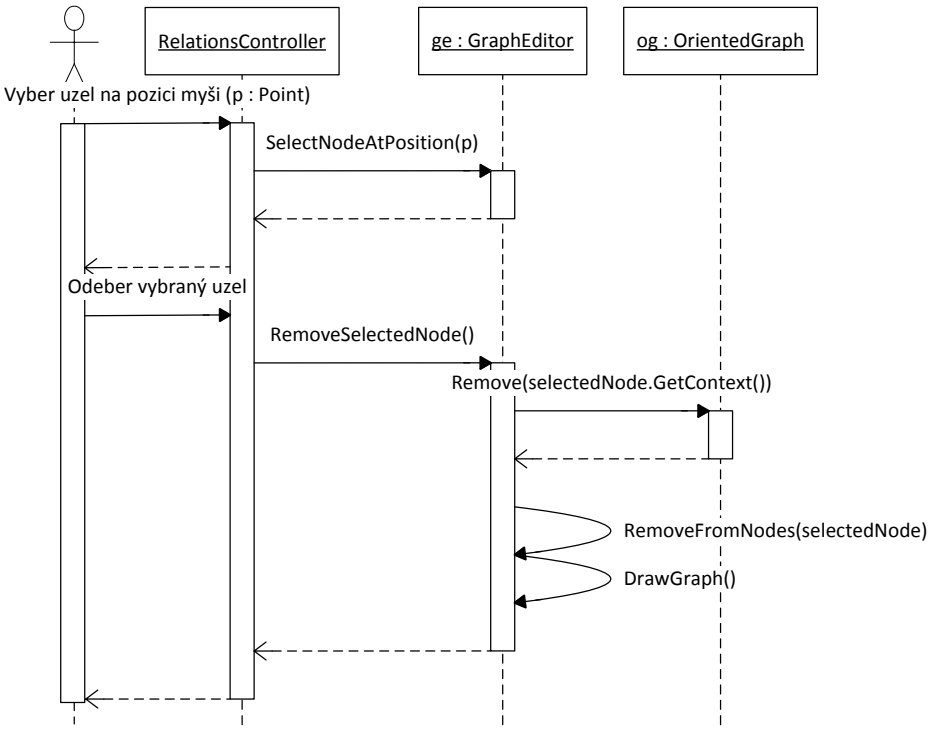
Obrázek 72: Sekvenční diagram reakce komponenty nápovědy na výběr konkrétní valuace



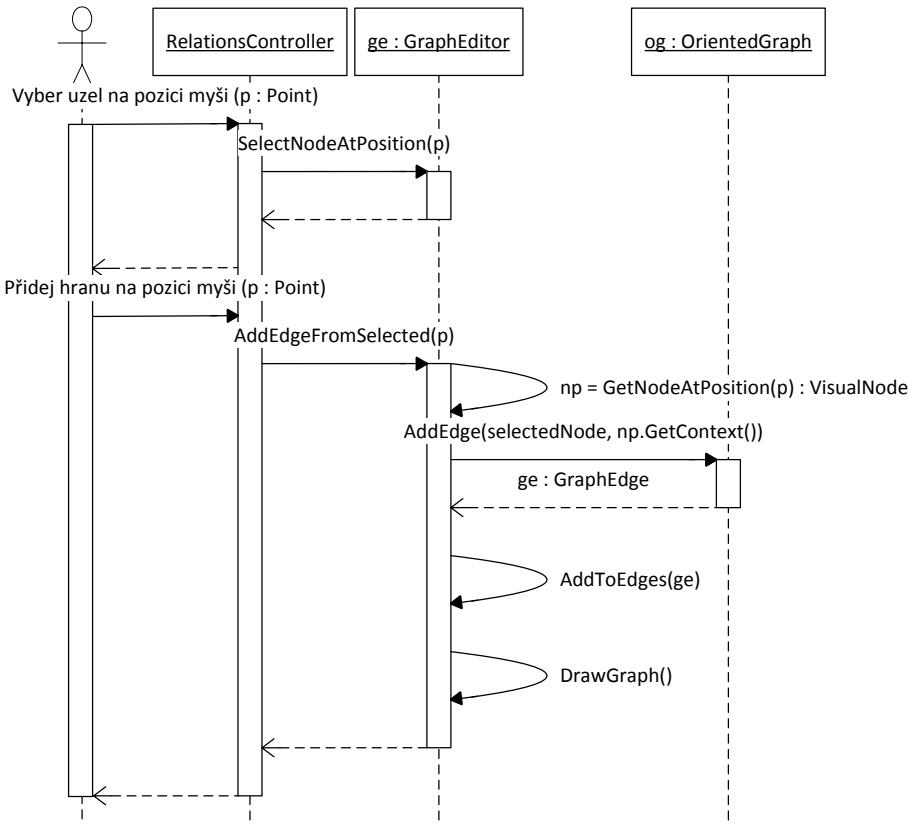
Obrázek 73: Sekvenční diagram ověření modelu množiny pro potřeby nápovědy



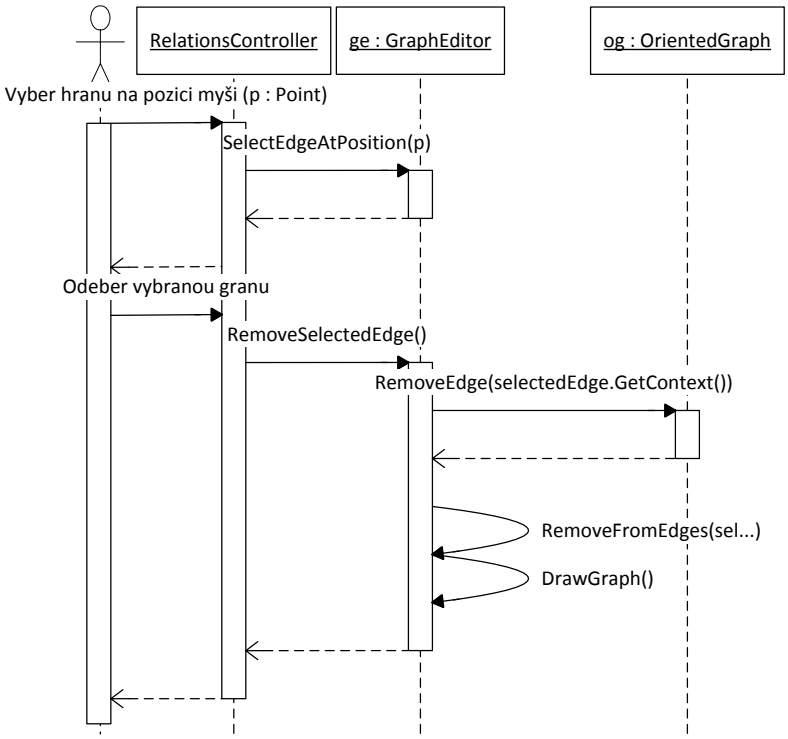
Obrázek 74: Sekvenční diagram přidání uzlu do grafu



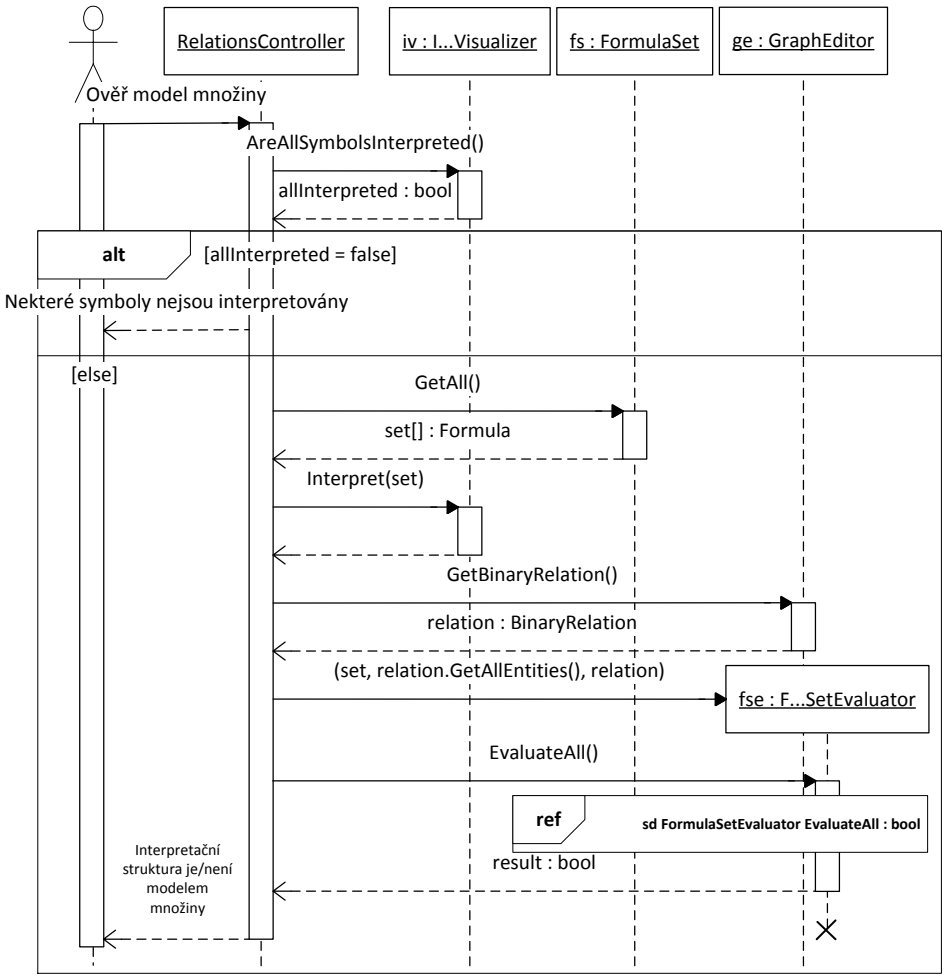
Obrázek 75: Sekvenční diagram odebrání uzlu z grafu



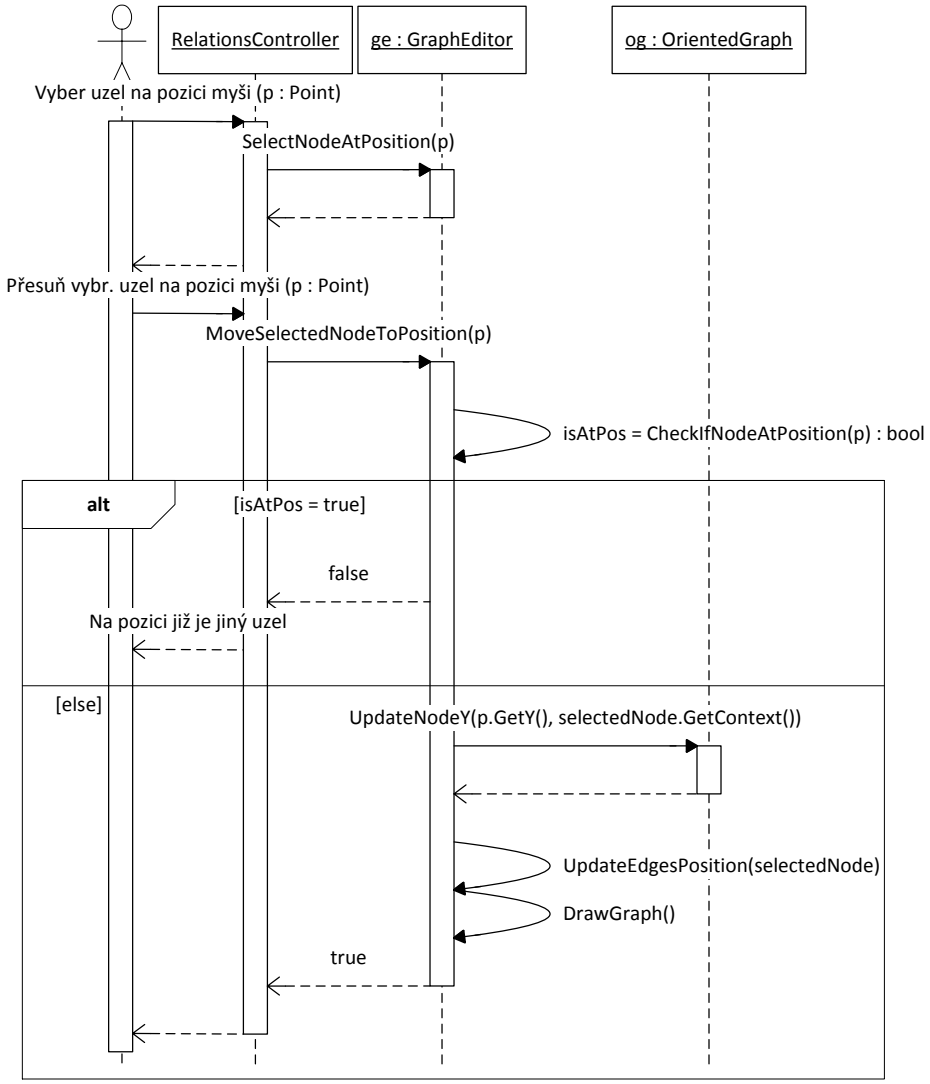
Obrázek 76: Sekvenční diagram přidání hrany do grafu



Obrázek 77: Sekvenční diagram odebrání hrany z grafu



Obrázek 78: Sekvenční diagram ověření, zda daná struktura splňuje požadované axiomy



Obrázek 79: Sekvenční diagram přesunu uzlu na jinou pozici

D Výpisy zdrojových kódů

```
class ValuationGenerator {
    private IEntity [] _entities ; // Universum
    private VariableType[] _variables ; // Poradi promennych ve formuli

    public ValuationGenerator(Formula f, IEntity [] entities ) {
        _entities = entities ;
        _variables = f.GetQuantificators().Select(q => q.GetVariableType());
    }

    public Valuation GetValuation(int varIndex, int entityIndex) {
        return new Valuation(_variables[varIndex], _entities [entityIndex])
    }
}
```

Výpis 8: Způsob generování valuací

```
class FormulaEvaluator {
    private Formula _formula;
    private object _context;
    private IEntity [] _entities ;
     ...

    public bool Evaluate() {
        // Uprava formule pred vyhodnocenim pravdivosti v IS
        ...

        if (_formula.HasQuantificators())
            return EvaluateWithQuantificators();
        return EvaluateOnlyConstants();
    }

    private bool EvaluateWithQuantificators() {
        var generator = new ValuationGenerator(_formula, _entities);
        IQuantificatorEvaluator qe = null;

        foreach (Quantificator q in _formula.GetQuantificators().Reverse()) {
            IQuantificatorEvaluator current;

            if (q.GetType() == QuantificatorType.Exists)
                current = new ExistsEvaluator(generator, _formula, qe, _context);
            else
                current = new AllEvaluator(generator, _formula, qe, _context);

            qe = current;
        }

        return eq.Evaluate(0);
    }

    private bool EvaluateOnlyConstants() {
        return new FormulaTruthEvaluator(_formula, _context).GetTruthValue()
```

```
        }
    }
```

Výpis 9: Implementace vyhodnocení pradivosti formule v interpretaci s kvantifikátory

```
class ExistsEvaluator : IQuantificatorEvaluator {
    private ValuationGenerator _generator
    private Formula _formula;
    private IQuantificatorEvaluator _next;
    private object _context;
    ...

    public bool Evaluate(int varIndex) {
        for (int i=0; i<_generator.GetEntityCount(); i++) {
            Valuation v = _generator.GetValuation(varIndex, i);
            _formula.SetValuation(v);

            bool iResult;
            if (next != null) // Nasleduje dalsi kvantifikator
                iResult = next.Evaluate(varIndex + 1);
            else // Toto je posledni kvantifikator v rade
                iResult = new FormulaTruthEvaluator(_formula, _context);

            if (iResult) // Formule je alespon jednou splnena
                return true;
        }
        return false; // Formule neni splnena
    }
}
```

Výpis 10: Implementace existenčního kvantifikátoru

```
class AllEvaluator : IQuantificatorEvaluator {
    private ValuationGenerator _generator
    private Formula _formula;
    private IQuantificatorEvaluator _next;
    private object _context;
    ...

    public bool Evaluate(int varIndex) {
        for (int i=0; i<_generator.GetEntityCount(); i++) {
            Valuation v = _generator.GetValuation(varIndex, i);
            _formula.SetValuation(v);

            bool iResult;
            if (next != null) // Nasleduje dalsi kvantifikator
                iResult = next.Evaluate(varIndex + 1);
            else // Toto je posledni kvantifikator v rade
                iResult = new FormulaTruthEvaluator(_formula, _context).GetTruthValue();

            if (!iResult) // Formul neni splnena
                return false;
```

```
        }
        return true; // Formule je splnenapro vsechna ohodnoceni
    }
}
```

Výpis 11: Implementace všeobecného kvantifikátoru

```
Polygons = new[]
{
    Shape.Dodecahedron,
    Shape.Octahedron,
    Shape.Tetrahedron,
    Shape.Pyramid,
    Shape.Cube,
    Shape.SquareCuboid
};
CircleBase = new[] { Shape.Cylinder, Shape.Cone };
SquareBase = new[]
{
    Shape.Cube,
    Shape.SquareCuboid,
    Shape.Pyramid
};
TriangleSurface = new[] {Shape.Pyramid, Shape.Tetrahedron};

// x ma cervenou barvu
public bool Interpret(object ctx, IEntity [] e) {
  return ((Entity3D)e[0]).GetColour() == Color.Red;
}
// x je kuzel
public bool Interpret(object ctx, IEntity [] e) {
  return ((Entity3D)e[0]).GetShape() == Shape.Cone;
}
// x ma malou velikost
public bool Interpret(object ctx, IEntity [] e) {
  return ((Entity3D)e[0]).GetSize() == Size.Small;
}
// x je mnohouhelnik
public bool Interpret(object ctx, IEntity [] e) {
  return Polygons.Contains(((Entity3D)e[0]).GetShape());
}
// x ma kruhovou podstavu
public bool Interpret(object ctx, IEntity [] e) {
  return CircleBase.Contains(((Entity3D)e[0]).GetShape());
}
// x mactvercovou podstavu
public bool Interpret(object ctx, IEntity [] e) {
  return SquareBase.Contains(((Entity3D)e[0]).GetShape());
}
// x ma plastje cely nebo castecne slozen z trojuhelniku
public bool Interpret(object ctx, IEntity [] e) {
  return TriangleSurface.Contains(((Entity3D)e[0]).GetShape());
}
// x je vyse nebo stejne vysoko jako y
```

```
public bool Interpret(object ctx, IEntity [] e) {
  return ((Entity3D)e[0]).GetY() >= ((Entity3D)e[1]).GetY();
}
// x je nad nebo pod y
public bool Interpret(object ctx, IEntity [] e) {
  return (((Entity3D)e[0]).GetY() > ((Entity3D)e[1]).GetY()) || (((Entity3D)e[0]).GetY() < ((
      Entity3D)e[1]).GetY())
}
// x je vedle y
public bool Interpret(object ctx, IEntity [] e) {
  return (
            var e0 = ((Entity3D)e[0]);
            var e1 = ((Entity3D)e[0]);
           (
                ((e0.GetX() + 1) == e1.GetX()) || ((e0.GetX() - 1) == e1.GetX()) &&
                (e0.GetY() == e1.GetY()) && (e0.GetZ() == 0.GetZ())
            )
           ||
           (
                ((e0.GetY() + 1) == e1.GetY()) || ((e0.GetY() - 1) == e1.GetY()) &&
                (e0.GetX() == e1.GetX() ) && (e0.GetZ() == e1.GetZ())
           )
           ||
           (
                (e0.GetZ() + 1) == e1.GetZ()) || ((e0.GetZ() - 1) == e1.GetZ()) &&
                (e0.GetX() == e1.GetX()) && (e0.GetY() == e1.GetY())
           )
}
// <
public bool Interpret(object ctx, IEntity [] e) {
   return ((BinaryRelation)ctx).AreRelated(e[0], e[1]) && e[0].PositionY < e[1].PositionY;
}
// x je v relaci s y
public bool Interpret(object ctx, IEntity [] e) {
   return ((BinaryRelation)ctx).AreRelated(e[0], e[1])
}
// =
public bool Interpret(object ctx, IEntity [] e) {
  return object.ReferenceEquals(e[0], e[1]);
}
```

Výpis 12: Příklady implementace relací

```
<?xml version="1.0" encoding="utf-8"?>
<folmodeler version="2.0" taskType="InterpretationStructures" xmlns="http://homel.vsb.cz/~mat675/
    FOLModeler.xsd">
  <formulaSet>
    <formula>VxP(f(x))</formula>
    <formula>P(a)</formula>
  </formulaSet>
  <interpretations>
    <unaryPredicate name="p" value="HasColorGreen" />
    <unaryFunction name="f">
      <map from="1" to="6" />
```

```
      <map from="2" to="6" />
      <map from="3" to="6" />
      <map from="4" to="2" />
      <map from="5" to="2" />
      <map from="6" to="2" />
      <map from="7" to="3" />
    </unaryFunction>
    <constant name="a" entityID="6" />
  </interpretations>
  <universum sizex="8" sizey="5" sizez="5">
    <entity size="big" shape="cone" color="pink" x="0" y="0" z="1" id="1" />
    <entity size="big" shape="tetrahedron" color="pink" x="7" y="2" z="4" id="2" />
    <entity size="big" shape="pyramid" color="yellow" x="5" y="4" z="2" id="3" />
    <entity size="medium" shape="cone" color="pink" x="5" y="2" z="3" id="4" />
    <entity size="small" shape="squarecuboid" color="red" x="4" y="1" z="4" id="5" />
    <entity size="big" shape="cube" color="blue" x="7" y="2" z="0" id="6" />
    <entity size="medium" shape="cylinder" color="green" x="0" y="1" z="1" id="7" />
  </universum>
</folmodeler>
```

Výpis 13: Příklad XML struktury v prostředí pro vytváření IS

```
<?xml version="1.0" encoding="utf-8"?>
<folmodeler version="2.0" taskType="RelationStructures" xmlns="http://homel.vsb.cz/~mat675/
    FOLModeler.xsd">
  <formulaSet>
    <formula>VxVyVz((R(x,y)*R(y,z))&gt;R(x,z))</formula>
  </formulaSet>
  <interpretations>
    <binaryPredicate name="R" value="IsInRelation" />
  </interpretations>
  <universum>
    <nodes>
      <node id="0" position="1740,1860" />
      <node id="1" position="1890,1860" />
      <node id="2" position="1740,2010" />
      <node id="3" position="1890,2010" />
    </nodes>
    <edges>
      <edge from="3" to="1" start="1890,2010" end="1890,1870" curvePoint="1890,1935" />
      <edge from="1" to="3" start="1890,1860" end="1890,2000" curvePoint="1890,1935" />
      <edge from="3" to="2" start="1890,2010" end="1750,2010" curvePoint="1815,2010" />
      <edge from="2" to="3" start="1740,2010" end="1880,2010" curvePoint="1815,2010" />
      <edge from="2" to="0" start="1740,2010" end="1740,1870" curvePoint="1740,1935" />
      <edge from="0" to="2" start="1740,1860" end="1740,2000" curvePoint="1740,1935" />
      <edge from="0" to="1" start="1740,1860" end="1880,1860" curvePoint="1815,1860" />
      <edge from="1" to="0" start="1890,1860" end="1750,1860" curvePoint="1815,1860" />
      <edge from="2" to="1" start="1740,2010" end="1882.92893218813,1867.07106781187"
          curvePoint="1815,1935" />
      <edge from="1" to="2" start="1890,1860" end="1747.07106781187,2002.92893218813"
          curvePoint="1815,1935" />
      <edge from="0" to="3" start="1740,1860" end="1882.92893218813,2002.92893218813"
          curvePoint="1815,1935" />
```

```
        <edge from="3" to="0" start="1890,2010" end="1747.07106781187,1867.07106781187"
              curvePoint="1815,1935" />
        <edge from="1" to="1" start="1887,1863" end="1900,1864" curvePoint="1950,1920" />
        <edge from="3" to="3" start="1887,2013" end="1900,2014" curvePoint="1950,2070" />
        <edge from="2" to="2" start="1737,2013" end="1750,2014" curvePoint="1800,2070" />
        <edge from="0" to="0" start="1737,1863" end="1750,1864" curvePoint="1680.5,1907.98" />
      </edges>
    </universum>
  </folmodeler>
```

Výpis 14: Příklad XML struktury v prostředí pro modelování vlastností relací

E Obsah CD

Adresář / Soubor	Popis
/ANTLR Gramatiky	Gramatiky systému ANTLR [12] použité pro vytvoření parserů uživatelského vstupu
/Install	Instalovatelná verze aplikace
/Zdrojové kódy	Zdrojové kódy aplikace FOLModeler a TEXsoubor textu práce
/FOLModeler.xsd	Definice XML struktury soborů
/dokumentace.pdf	Uživatelská dokumentace nástroje FOLModeler
/modelovani_is_v_pl1.pdf	Elektronická verze práce

Tabulka 10: Popis obsahu přiloženého CD