VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Nástroj pro dokazování platnosti úsudků pomocí Vennových diagramů

A Tool for Proving the Argument Validity Using Venn Diagrams

Daniel Honus

Bakalářská práce

Vedoucí práce: Mgr. Marek Menšík, Ph.D.

Ostrava, 2023

VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Č.j.VSB/23/046803
vsboee23013DD1

Zadání bakalářské práce

Student: **Daniel Honus**

Studijní program: B0613A140014 Informatika

Téma: Nástroj pro dokazování platnosti úsudků pomocí Vennových diagramů
A Tool for Proving the Argument Validity Using Venn Diagrams

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je naimplementovat nástroj pro podporu dokazování platnosti úsudků v predikátové logice 1. řádu (PL1) za pomocí Vennových diagramů.

Práce bude obsahovat:
1. Teorii PL1.
3. Teorii dokazování platnosti úsudků a tautologičnosti formulí v PL1.
4. Analýzu, návrh a implementaci systému pro dokazování platnosti úsudků pomocí Vennových diagramů.

Seznam doporučené odborné literatury:

[1] Vopěnka, P.: Úvod do klasické teorie množin, Fragment 2015, ISBN: 978-80-253-1251-3
[2] Švejdar, V.: Logika - neúplnost, složitost a nutnost, Academia Prague 2002, ISBN: 978-80-200-1005-6
[3] Duží, M.: Logika pro informatiky, Skripta VŠB-TU Ostrava, 2012

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

Vedoucí bakalářské práce: **Mgr. Marek Menšík, Ph.D.**

Datum zadání: 01.09.2022

Datum odevzdání: 04.07.2023

Garant studijního programu: doc. Mgr. Miloš Kudělka, Ph.D.

V IS EDISON zadáno: 30.05.2023 18:37:41

Abstrakt

Cílem této bakalářské práce je vytvořit interaktivní nástroj pro dokazování platnosti úsudků pomocí Vennových diagramů v predikátové logice 1. řádu. Práce nejdříve v první části popisuje jazyk výrokové a predikátové logiky, formalizaci přirozeného jazyka do PL1 a dokazování úsudků metodou Vennových diagramů. Pozornost je také věnována syntaktické analýze vstupních řetězců. V druhé části se zaměřuje na návrh a implementaci nástroje, který uživatelům umožňuje zadávat logické úsudky a zobrazovat Vennovy diagramy pro vizuální podporu rozhodování jejich platnosti.

Klíčová slova

predikátová logika, výroková logika, PL1, Vennovy diagramy, syntaktická analýza

Abstract

The goal of this thesis is the creation of an interactive tool for proving argument validity in first-order predicate logic using Venn diagrams. The thesis first describes the language of propositional and predicate logic, the formalization of natural language in PL1, and the proof of arguments using Venn diagrams. Syntactic analysis of input strings is also discussed. The second part focuses on the design and implementation of a tool that allows users to enter logical arguments and display Venn diagrams to visually support the reasoning for their validity.

Keywords

predicate logic, propositional logic, PL1, Venn diagrams, syntactic analysis

Poděkování

Rád bych poděkoval vedoucímu mé bakalářské práce, Mgr. Markovi Menšíkovi, Ph.D., za jeho čas a vstřícný přístup, který mi věnoval při řešení této práce.

Obsah

Seznam použitých symbolů a zkratek **7**

Seznam obrázků **8**

Seznam tabulek **9**

1 Úvod **10**

2 Teoretické základy **11**

2.1 Výroková logika 12
2.2 Predikátová logika 15
2.3 Formalizace přirozeného jazyka 17
2.4 Teorie množin 18
2.5 Aristotelova logika 19
2.6 Řešení pomocí Vennových diagramů 23
2.7 Syntaktická analýza 28

3 Návrh a analýza **30**

3.1 Cíl práce 30
3.2 Použité technologie 30
3.3 Třídní diagram 32
3.4 Sekvenční diagram 33
3.5 Vymezení podmnožiny jazyka 33
3.6 Syntaktická analýza 35
3.7 Proces řešení 36
3.8 Vizualizace a interaktivita Vennových diagramů 37
3.9 Interaktivní ověření platnosti 38
3.10 Uživatelské rozhraní 39

4 Závěr **41**

Literatura **42**

Přílohy **42**

A Uživatelská příručka **43**

B Návod pro spuštění **47**

Seznam použitých zkratek a symbolů

VL	–	Výroková logika
PL	–	Predikátová logika
PL1	–	Predikátová logika 1. řádu
LL	–	Syntaktický analyzátor shora-dolů
API	–	Application Programming Interface
HTTP	–	Hypertext Transfer Protocol
DOM	–	Document Object Model

Seznam obrázků

2.1 Logický čtverec 19
2.2 Eulerův diagram 21
2.3 Vennův diagram korespondující k předchozímu Eulerovu diagramu 21
2.4 Oblasti Vennova diagramu 22

3.1 Třídní diagram 32
3.2 Sekvenční diagram 33
3.3 Základní Vennův diagram 37
3.4 Diagram v případě, že není možno umístit křížek 38
3.5 Interaktivní řešení 39
3.6 Uživatelské rozhraní 40

A.1 Špatně zadaná premisa 44
A.2 Zadání premis 44
A.3 Kroky vedoucí k řešení 45
A.4 Interaktivní mód 45
A.5 Zadání řešení 46
A.6 Srovnání výsledku 46

Seznam tabulek

2.1 Pravdivostní tabulka 13
2.2 Figury kategorických sylogismů 20

3.1 Alternativní symboly 34

A.1 Alternativní symboly 43

Kapitola 1

Úvod

Cílem této bakalářské práce je implementovat nástroj, s jehož pomocí lze v PL1 ověřovat platnost úsudků zakreslitelných Vennovými diagramy. Výsledný program je webová aplikace umožňující zadávat úsudky v PL1 a následně získávat informace o jejich platnosti a řešení. Nástroj vypisuje kroky, které vedly k výsledku, což uživateli vyjasní případné nesrovnalosti. Konečným výsledkem je skupina Vennových diagramů, pomocí kterých bude výsledek zobrazen.

Tento nástroj je potenciálně užitečný pro studium logiky, matematiky nebo informatiky pro pochopení logického vyplývání. Proces řešení je názorně zachycen pomocí množin. Navíc tím, že nástroj umožňuje zobrazit kroky vedoucí k výsledku, může být využit pro výuku, například v předmětech zabývajících se úvodem do logiky.

V první části práce se budu věnovat základní teorii VL, PL1, Aristotelově logice a Vennovým diagramům. Zmíním také syntaktickou analýzu, která umožňuje zpracování vstupních řetězců. V druhé části se zaměřím na návrh a implementaci nástroje.

Kapitola 2

Teoretické základy

V této práci se budu zabývat především úsudky PL1 a následně ověřováním platnosti těchto úsudků pomocí metody Vennových diagramů. Pro účely dokazování platnosti úsudků je zapotřebí formálního jazyka. Ten mimo jiné umožňuje popsat vztahy mezi objekty a využít nad nimi logických operací. Přestože je tato práce zaměřena na PL1, je pro její pochopení vhodné porozumět základům VL. Pro ověřování platnosti úsudků je důležité zvolit vhodný logický systém, jelikož ne všechny deduktivně správné úsudky lze ověřit pomocí libovolného logického systému kvůli rozdílům v expresivitě.

Metody dokazování platnosti ve VL a PL se dělí na syntaktické a sémantické. Sémantické metody se zaměřují na význam a pravdivost výroků. Mezi sémantické metody patří například tabulková metoda nebo Vennovy diagramy. Syntaktické metody nezávisí na významu výroků. Syntaktické metody, jako například rezoluční metoda, se také využívají při automatickém dokazování.

„*Obecně můžeme* ***úsudek*** *charakterizovat následujícím schématem: Na základě pravdivosti výroků (soudů, tvrzení)* V_1*, ...,* V_n *soudím, že je pravdivý rovněž výrok V. Zapisujeme schematicky:* V_1*, ...,* V_n */ V nebo častěji:*“[5]

premisa P_1
premisa P_2
...
premisa P_n
$\therefore$ Závěr Z

Definice 1 Deduktivně platný úsudek [5, s. 6]

Úsudek P_1*, ...,* P_n */* Z je deduktivně platný (*správný*), značíme P_1*, ...,* $P_n \models Z$, jestliže závěr Z analyticky vyplývá z předpokladů P_1*, ...,* P_n, tj. za všech okolností takových, že jsou pravdivé všechny předpoklady P_1*, ...,* P_n, je (za těchto okolností) pravdivý i závěr Z.

Závěr Z je tedy vždy pravdivý v případě, že jsou pravdivé všechny předpoklady. V definici však hovoříme o podmíněné pravdivosti, nikoliv o aktuální pravdivosti [3]. Například závěr platného úsudku:

Jestliže sněží, je zima.
Sněží.

Je zima.

z premis vyplývá i v případě, že momentálně nesněží, nebo není zima.

2.1 Výroková logika

V této kapitole jsem čerpal z [5] [10].

Výroková logika (VL) je formální logika, která analyzuje věty až do úrovně elementárních výroků, které jsou samostatné a nedělitelné. *Výrok* je tvrzení, o němž má smysl prohlásit, zda je pravdivé či nepravdivé. VL nezkoumá strukturu těchto samotných výroků. Elementární výroky nemají žádnou vnitřní stavbu a jejich atributy jsou pouze pravdivostní hodnoty [5, s. 5].

V rámci výrokové logiky rozlišujeme mezi jednoduchými a složenými výroky. Jednoduchý (elementární) výrok není složený z jiných výroků. Složený výrok se pak skládá z jednoduchých. Prostředkem pro složení jednoduchých výroků do složeného výroku jsou výrokové spojky. *„V sémantice výrokového logiky budeme zkoumat pravdivostní hodnotu složeného výroku v závislosti na pravdivostních hodnotách jeho složek. Celé naše pojetí sémantiky naprosto rozhodujícím způsobem ovlivní přirozený požadavek, aby hodnota složeného výroku nezáležela na ničem jiném než na pravdivostních hodnotách jeho složek. Přijetím tohoto předpokladu umožní mj. popsat pravdivostní hodnoty složených výroků pomocí tabulek pravdivostních hodnot"*[10]. Jedná se o princip *kompozicionality.*

2.1.1 Sémantický výklad

Formální jazyk je určen abecedou a gramatikou. Jazyk teorie je množina všech (dobře utvořených) formulí jazyka.[5, s. 56]. Jazyk výrokové logiky tedy udává všechny dobře utvořené formule výrokové logiky. Gramatika jazyka výrokové logiky z její abecedy rekurzivně definuje pravidla, podle kterých se vytváří dobře utvořené formule.

Definice 2 *[5, s. 15] Abeceda jazyka výrokové logiky je množina následujících symbolů:*

- *Výrokové symboly: p, q, r, ... (případně s indexy)*
- *Pomocné symboly (závorky): (,), případně [,],{,}*
- *Symboly logických spojek (funktorů): ($\neg, \wedge, \vee, \supset, \equiv$)*

Tyto logické spojky nazýváme po řadě negace, konjunkce, disjunkce, implikace, ekvivalence. Jsou seřazeny sestupně dle jejich priority.

Definice 3 *Gramatika jazyka výrokové logiky induktivně definuje nekonečnou množinu formulí:[5, s. 15]:*

1. *Výrokové symboly jsou formule (báze definice)*
2. *Jsou-li výrazy A, B formule, pak jsou formulemi i výrazy* $(\neg A)$, $(A \wedge B)$, $(A \vee B)$, $(A \supset B)$, $(A \equiv B)$
3. *Jiných formulí výrokové logiky, než podle bodů (1), (2) není (uzávěr definice).*

Jazyk výrokové logiky *je množina všech dobře utvořených formulí výrokové logiky.*

2.1.2 Pravdivostní vyhodnocení

Definice 4 *[5, s. 16]* ***pravdivostní vyhodnocování formulí***

Pravdivostní ohodnocení (valuace) *výrokových symbolů je zobrazení v, které ke každému výrokovému symbolu přiřazuje pravdivostní hodnotu, tj. hodnotu z množiny {1,0}, která kóduje množinu pravda, nepravda.*

Pravdivostní funkce *formule výrokové logiky je funkce w, která ke každému pravdivostnímu ohodnocení výrokových symbolů přiřazuje pravdivostní hodnotu celé formule. Tato hodnota je určena takto:*

1. *Pravdivostní hodnota elementární formule je rovna valuaci výrokového symbolu, tj.* $w(p)_v = v(p)$ *pro všechny výrokové proměnné p.*
2. *Jsou-li dány pravdivostní funkce formulí A, B, pak pravdivostní funkce formulí* $\neg A$, $A \wedge B$, $A \vee B$, $A \supset B$, $A \equiv B$ *jsou dány následující tabulkou:*

A	B	$\neg A$	$A \wedge B$	$A \vee B$	$A \supset B$	$A \equiv B$
1	1	0	1	1	1	1
1	0	0	0	1	0	0
0	1	1	0	1	1	0
0	0	1	0	0	1	1

Tabulka 2.1: Pravdivostní tabulka

Pro ověření platnosti a tautologičnosti výroku lze tedy využít tabulkové metody. Tabulka obsahuje sloupce pro výrokové symboly a řádky pro veškeré kombinace jejich pravdivostních hodnot.

Definice 5 [5, s. 20]

Každé ohodnocení v výrokových symbolů obsažených ve formuli A, pro které je hodnota pravdivostní funkce rovna 1, tedy $w(A)_v = 1$, se nazývá **model** této **formule**.

Formule A výrokové logiky je **splnitelná**, je-li $w(A)_v = 1$ pro nějaké ohodnocení v, neboli existuje aspoň jeden model formule A.

Formule A výrokové logiky je **tautologií**, je-li $w(A)_v = 1$ pro všechna ohodnocení v, neboli každé ohodnocení je modelem formule A. Skutečnost, že formule A je tautologií, označujeme zápisem $\vDash A$.

Formule A výrokové logiky je **kontradikcí**, jestliže neexistuje takové ohodnocení výrokových symbolů, pro které by hodnota pravdivostní funkce formule A byla rovna 1, tj. $w(A)_v = 0$ pro všechna ohodnocení v, formule nemá model.

Množina formulí M je **splnitelná**, jestliže existuje valuace v taková, že $w(A)_v = 1$ pro *každou* formuli $A \in M$. Takové ohodnocení v se pak nazývá **model množiny M**.

Příkladem platného úsudku ve VL je například následující:

Jestliže jsem ryba (r), umím plavat (p).
Jestliže nejsem ryba, nemám rád vodu (v).

Umím plavat, nebo nemám rád vodu.

$r \supset p$
$\neg r \supset \neg v$

$p \vee \neg v$

Výroková logika avšak není mimo jiné dostačující pro úsudky, ve kterých na sebe mají jednotlivé vnitřní komponenty vazby. Například formalizace výroku „Všichni lidé jsou savci“ není ve VL dostačující, jelikož je potřeba kvantifikovat objekty (lidi a savce) a vyjádřit vztah mezi nimi. V predikátové logice by formalizace zmíněného výroku vypadala následovně:

$$\forall x[C(x) \supset S(x)] \tag{2.1}$$

kde,

- x je předmětová (individuová) proměnná v dané předmětné oblasti,
- C, S jsou určité vlastnosti předmětů z universa diskursu, v našem případě se jedná o vlastnosti „být člověkem“ a „být savcem“,
- zápis $\forall x[]$ značí, že pro všechna individua z předmětné oblasti platí to, co je uvedeno v hranatých závorkách

Popis převzat z [5, s. 68]

Podobně, (zjevně) platný úsudek:

p: Plazi neumí létat.
q: Želvy jsou plazi.
---
r: Želvy neumí létat.

nelze ve výrokové logice přesně vyjádřit. Termíny „želvy“ a „plazi“ se nachází ve výrocích p i q a existuje mezi nimi vazba. Úsudek v jeho formalizaci také vyžaduje využití kvantifikátorů.

2.2 Predikátová logika

V této kapitole jsem čerpal z [5] [4]

Jelikož ne všechny úsudky lze dokazovat pomocí výrokové logiky, budeme se zabývat jejím rozšířením - predikátovou logikou. Tato práce se zabývá predikátovou logikou 1. řádu. Predikátová logika *„formalizuje úsudky o vlastnostech předmětů a vztazích mezi předměty pevně dané předmětné oblasti (univerza)“* [5, s. 68].

Definice 6 Universum:

- Universum je neprázdná množina individuí (prvků).

2.2.1 Jazyk predikátové logiky

Nejprve je nutno definovat jazyk predikátové logiky 1. řádu. Ten oproti jazyku výrokové logiky mj. vyjadřuje individua universa diskursu – *termy* (konstanty, proměnné a funkční termy) – a *predikátové symboly* jako P, Q jako vlastnosti jednotlivých individuí a vztahy mezi nimi.

Definice 7 *[5, s. 69] Abeceda predikátové logiky*

1. *Logické symboly*

   (a) *předmětové (individuové) proměnné: x, y, z, ... (příp. s indexy)*

   (b) *symboly pro spojky: $\neg$, $\wedge$, $\vee$, $\supset$, $\equiv$*

   (c) *symboly pro kvantifikátory*

      i. *$\forall$: Všeobecný (univerzální) kvantifikátor $\Rightarrow$ „Pro všechny prvky“*

      ii. *$\exists$: Existenční kvantifikátor $\Rightarrow$ „Existuje prvek“*

   (d) *případně binární predikátový symbol = (predikátová logika s rovností)*

   Speciální symboly (určují specifiku jazyka)

   - *predikátové symboly velkým písmenem: P, Q, R (příp. s indexy)*
   - *funkční symboly: f, g, h, ... (příp. 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 funkčního symbolu nebo predikátu.*
   - *Pomocné symboly (závorky):* $(,)$ (*případně i* $[,], \{,\}$)

Definice 8 *[5, s. 69] Gramatika*

1. ***termy:***

   (a) *každý symbol proměnné je atomický term*

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

   (c) *jen výrazy dle a. a b. jsou termy*

2. ***atomické formule***

   - *je-li* P *n-ární predikátový symbol a jsou-li* $t_1, ..., t_n$ *termy, pak výraz* $P(t_1, ..., t_n)$ *je atomická formule*
   - *jsou-li* t_1 *a* t_2 *termy, pak výraz* $(t_1 = t_2)$ *je atomická formule*

3. ***(složené) formule***

   (a) *každá atomická formule je formule*

   (b) *je-li výraz A formule, pak* $\neg A$ *je formule*

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

   (d) *je-li x proměnná a A formule, pak výrazy* $\forall x\ A$ *a* $\exists x\ A$ *jsou formule*

   (e) *jen výrazy dle a. – d. jsou formule*

Přestože výskyt proměnných může být volný i vázaný, zabývá se tato práce pouze uzavřenými formulemi.

Definice 9 *[5, s. 70] Výskyt proměnné x ve formuli A je vázaný, jestliže je součástí nějaké podformule* $\exists x\ B(x)$ *nebo* $\forall x\ B(x)$ *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. Formule se nazývá uzavřenou, neobsahuje-li žádnou volnou proměnnou.

2.3 Formalizace přirozeného jazyka

V této kapitole jsem čerpal z [5] [4]

Abychom mohli vyhodnotit a správně zakreslit výrazy, je nutno je nejprve převést do predikátové logiky. Při tomto převodu nejprve určíme klíčová slova vyjadřující kvantifikátory a samotná individua universa. Například výraz „Všichni lidé jsou smrtelní“ může být do predikátové logiky převeden jako:

$$\forall x \; [C(x) \supset S(x)] \tag{2.2}$$

V této formuli je použit **všeobecný kvantifikátor**, který udává že formule platí pro všechna individua celého universa. Klíčová slova jako „všichni“, „nikdo“, „každý“ apod. lze formalizovat všeobecným kvantifikátorem. Po všeobecném kvantifikátoru $\forall$ následuje (většinou) formule ve tvaru implikace ($\supset$), kdežto po existenčním kvantifikátoru $\exists$ (většinou) formule ve tvaru konjunkce ($\wedge$) [5]. Tyto tvary jsou ovlivněny universem diskursu. Všeobecný kvantifikátor kvantifikuje *všechny objekty v universu*. Předchozí výraz však říká, že **pokud** je objekt člověkem, **poté** je smrtelný. Kdybychom použili konjunkci, znamenalo by to, že **všechny** objekty jsou smrtelní lidé.

Výrazy jako „někteří“, „existují“ a „jsou takoví, že“ překládáme pomocí **existenčního kvantifikátoru**. Například větu „Někteří lidé mají modré oči“ lze převést jako:

$$\exists x \; [C(x) \wedge M(x)] \tag{2.3}$$

Při převodu do predikátové logiky také využíváme již zmíněných logických symbolů, tj. negace, konjunkce, disjunkce, implikace a ekvivalence. Jejich výskyt udávají slovní spojení jako „není“, „a“, „nebo“, „právě tehdy, když“ apod.

Třetí intuitivní vyjadřovací prostředek získáme, jestliže si uvědomíme, že v určité skupině můžeme vytknout jednotlivé objekty např. „kočka“ z množiny „savci“. Mohou tedy být určeny individuové **konstanty** [2.2.1]. V teorii množin je konstantou např. prázdná množina [10].

Pro správnou interpretaci a konečné vyhodnocení formulí je třeba určit universum diskursu. Například, pokud mluvíme o lidech, můžeme zvolit univerzum diskursu jako množinu všech lidí. Relace mezi těmito prvky vyjadřují vztah, který vůči sobě mají. Avšak např. interpretace formule, která je analýzou věty „Všichni lidé jsou smrtelní“, může být „Všichni lidé jsou stateční“. V této interpretaci by tato formule nebyla pravdivá.

Pravdivost i tautologičnost formule tedy můžeme určit pouze pro danou interpretaci. Obsahuje-li však formule nějaké volné proměnné, můžeme vyhodnotit její pravdivost v dané interpretaci pouze v závislosti na ohodnocení (valuaci) volných proměnných [5].

2.4 Teorie množin

Jako prostředek pro dokazování pravdivosti tvrzení za použití Vennových diagramů je tento nástroj založen na základních vlastnostech teorie množin. Těmto vlastnostem, jejich využití v dokazování platnosti PL1 logiky a procesu dokazování úsudků, je nutno porozumět dříve, než se můžeme věnovat samotným Vennovým diagramům.

Definice 10 [9, s. 24] *„Vyložíme-li nějaké dané ostře vydělené seskupení objektů jako objekt jediný, vytvoříme z tohoto seskupení množinu. Objekty náležící do tohoto seskupení se nazývají prvky takto vytvořené množiny.“*

- Množina je jednoznačně určena seskupením svých prvků (extenzionalita množiny).
- Zápis A $\in$ B značí, že prvek A je prvkem množiny B (A náleží B).
- Zápis A $\notin$ B značí, že objekt A *není* prvkem množiny B.
- Zápis A $\subseteq$ B značí, že $A,\ B$ jsou množiny a každý prvek množiny A je též prvkem množiny B. A je tedy **podmnožinou** B. $(\forall x(x \in A \Rightarrow x \in B))$
- Zápis A $\subset$ B (A $\subseteq$ B a A $\neq$ B) značí, že množina A je **vlastní podmnožinou** B.

Pro zjednodušení úvah týkajících se množin je užitečné zavést tzv. **prázdnou množinu**, to je taková která nemá žádné prvky. Budeme využívat znak $\emptyset$. Platí, že $\emptyset \neq \{\emptyset\}, \emptyset \neq \{\{\emptyset\}\}$.

Existuje souvislost mezi množinami a Vennovými diagramy, které využijeme při implementaci nástroje a následně také při rozhodování platnosti úsudků. Unární predikátové symboly lze reprezentovat jako množiny Vennova diagramu. Každá z množin je podmnožinou všech individuí (universa). Např. vlastnost „být člověkem“ můžeme považovat za množinu „lidé“. Logické spojky budou převedeny do množinové notace.

- Konjunkci $A \wedge B$ lze interpretovat průnikem $A \cap B = \{x : x \in A \wedge x \in B\}$,
- disjunkci $A \vee B$ jako sjednocení $A \cup B = \{x : x \in A \vee x \in B\}$,
- negaci jako doplněk množiny vzhledem k universu diskurzu $\overline{A} = \{x : x \notin A\}$.
- implikaci $A \supset B$ jako podmnožinu $(\subseteq)$
- ekvivalenci jako A = B

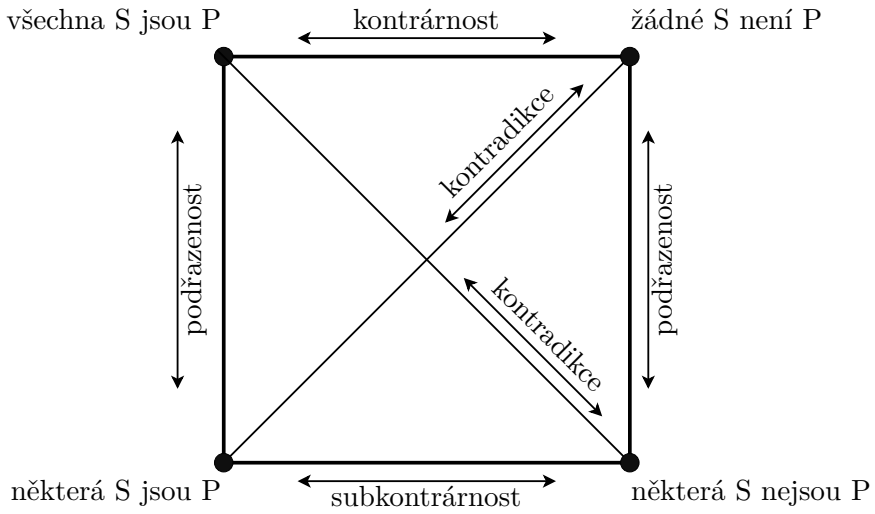
2.5 Aristotelova logika

Tato kapitola vychází z [10] [5] [3]

Aristotelova logika je pouze fragmentem predikátové logiky. Zkoumá specifické výroky, pro které se používá historický pojem „soud“, mající strukturu S–P, sestávající z tzv. subjektu a predikátu. Tyto Aristotelem popsané výroky, včetně jejich vzájemných vztahů, lze zobrazit tzv. logickým čtvercem (angl. „square of opposition“) 2.1.

U soudů z výrokového čtverce jsou rozeznávány dva druhy vlastností. Kvantita soudu určuje, zdali se jedná o obecný (kvantifikovaný pomocí všeobecného kvantifikátoru), nebo částečný (pomocí existenčního kvantifikátoru) soud. Jejich kvalita soudy rozděluje na kladné (pokud se v něm nevyskytuje negace) a záporné (pokud se v něm vyskytuje negace). Na základě těchto vlastností vznikají čtyři druhy soudů, reprezentovány písmeny *a*, *i*, *e*, *o*.

SaP	*obecný kladný soud*	Každé S je P	$\forall x(S(x) \supset P(x))$
SiP	*částečný kladný soud*	Některé S je P	$\exists x(S(x) \wedge P(x))$
SeP	*obecný záporný soud*	Žádné S není P	$\forall x(S(x) \supset \neg P(x))$
SoP	*částečný záporný soud*	Některé S není P	$\exists x(S(x) \wedge \neg P(x))$



Obrázek 2.1: Logický čtverec

Logický čtverec vyjadřuje následující vztahy soudů [5] [3]:

- **kontradiktoričnost** – správný opak, negace daného výroku; dané výroky mají vždy opačnou pravdivostní hodnotu.
  $SaP \equiv \neg SoP$ $SeP \equiv \neg SiP$
  Např. „Všechny labutě jsou bílé“–„Některé labutě nejsou bílé“.

- **subalternost** – lze přejít od a k i (nikoli však naopak), lze přejít od e k o (nikoli však naopak), čili a implikuje i a e implikuje o.
  $SaP \vDash SiP \qquad SeP \vDash SoP$
  Např. „Všechny labutě jsou bílé“–„Některé labutě jsou bílé“

- **kontrárnost** – výroky a a e nemohou být oba pravdivé, ovšem oba mohou být nepravdivé.
  $SaP \vDash \neg SeP \qquad SeP \vDash \neg SaP$
  Např. „Všechny labutě jsou bílé“–„Žádné labutě nejsou bílé“

- **subkontrárnost** – výroky o a i nemohou být oba nepravdivé, ovšem oba mohou být pravdivé.
  $\neg SiP \vDash SoP \qquad \neg SoP \vDash SiP$
  Např. „Některé labutě jsou bílé“–„Některé labutě nejsou bílé“

2.5.1 Kategorický sylogismus

Sylogismus je druh deduktivního argumentu, v němž je závěr odvozen ze dvou premis (vyšší a nižší premisy). „Sylogismus je **platný**, jestliže z pravdivosti premis *vždy* plyne pravdivost třetího soudu“ [10].

Jak již bylo zmíněno, tyto predikáty jsou složeny ze tří *termínů*, které jsou v rámci Aristotelovy logiky pojmenovány:

- *subjekt* S – v závěru stojí na místě subjektu a vyskytuje se ve druhé premise
- *predikát* P – v závěru stojí na místě predikátu a vyskytuje se v první premise
- *střední člen* M – vyskytuje se v obou premisách, avšak nikoli v závěru.

Kategorický sylogismus musí splňovat tyto podmínky. V tomto rozmístění jsou možné právě 4 figury.

Tabulka 2.2: Figury kategorických sylogismů

I. figura	**II. figura**	**III. figura**	**IV. figura**
M P	P M	M P	P M
S M	S M	M S	M S
S P	S P	S P	S P

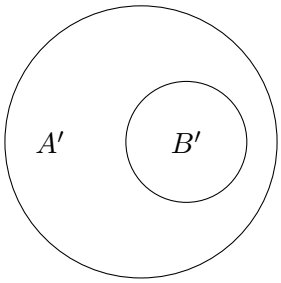
„Kombinací a, e, i, o lze nyní vytvořit 64 tzv. modů, z nichž jen některé jsou platné“[5]. Platnými mody v jednotlivých figurách jsou v tradičním názvosloví: (převzato z) [3]

- I. *barbara, celarent, darii, ferio* – za předpokladu neprázdnosti termínů též *barbari, celaront*
- II. *baroco, camestres, cesare, festino* – za předpokladu neprázdnosti termínů též *camestros, cesaro*

- III. *bocardo, datisi, disamis, ferison* – za předpokladu neprázdnosti termínů též *darapti, felapton* (oba mody byly ve středověku řazeny mezi platné díky přijímanému předpokladu neprázdnosti M)
- IV. *calemes (camenes), dimatis, fresison* – za předpokladu neprázdnosti termínů též *bamalip (bramantip), fesapo, calemos* (první dva mody byly ve středověku řazeny mezi platné, fesapo díky předpokladu neprázdnosti M, *bamalip* rovněž, ačkoli nezbytná je také neprázdnost P)

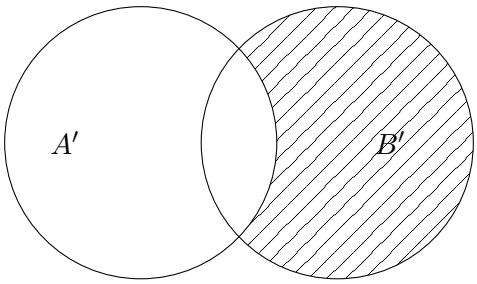
2.5.2 Vennovy diagramy

Průběžné kroky řešení a samotný výsledek budou v implementaci nástroje reprezentovány Vennovými diagramy. Vennovy diagramy vznikly jako řešení pro některé z problémů Eulerových diagramů, na kterých jsou založeny. Eulerův diagram je reprezentací prvků logického výroku, např. „Všechna B jsou A.“ pomocí kružnic, nebo jiných tvarů [1].



Obrázek 2.2: Eulerův diagram

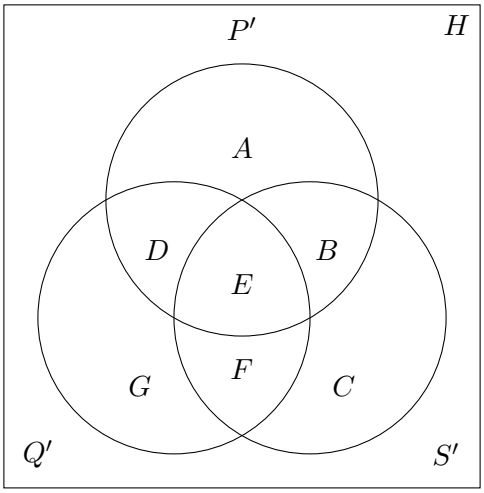
Na tomto jednoduchém diagramu lze vidět jeden z problémů Eulerových diagramů. Nelze totiž vyjádřit případ, kdy prvek spadá do B, ale nespadá do A. Bude obtížné podobný, nebo složitější problém algoritmicky vyřešit. Vennovy diagramy eliminují tento problém vykreslením všech tvarů (většinou kružnic) tak, aby každý protínal všechny již existující průniky [1, s. 5].



Obrázek 2.3: Vennův diagram korespondující k předchozímu Eulerovu diagramu

Oblast B je na obrázku 2.3 vyšrafována, indikující její prázdnost.

Definici množin v jazyce PL1 ilustrujeme následujícím obrázkem, převzato z [5, s. 79]:



Obrázek 2.4: Oblasti Vennova diagramu

A. množinově: $P \setminus (Q \cup S) = (P \setminus Q) \cap (P \setminus S)$
   v jazyce PL1: $P(x) \wedge \neg(Q(x) \vee S(x)) \Leftrightarrow P(x) \wedge \neg Q(x) \wedge \neg S(x)$

B. množinově: $(P \cap S) \setminus Q$
   v jazyce PL1: $P(x) \wedge S(x) \wedge \neg Q(x)$

C. množinově $S \setminus (P \cup Q) = (S \setminus P) \cap (S \setminus Q)$
   v jazyce PL1: $S(x) \wedge \neg(P(x) \vee Q(x)) \Leftrightarrow S(x) \wedge \neg P(x) \wedge \neg Q(x)$

D. množinově: $(P \cap Q) \setminus S$
   v jazyce PL1: $P(x) \wedge Q(x) \wedge \neg S(x)$

E. množinově: $(P \cap Q \cap S)$
   v jazyce PL1: $P(x) \wedge Q(x) \wedge S(x)$

F. množinově: $(Q \cap S) \setminus P$
   v jazyce PL1: $Q(x) \wedge S(x) \wedge \neg P(x)$

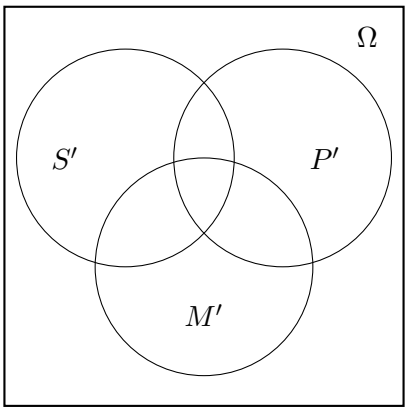
G. množinově: $Q \setminus (P \cup S) = (Q \setminus P) \cap (Q \setminus S)$
   v jazyce PL1: $Q(x) \wedge \neg(P(x) \vee S(x)) \Leftrightarrow Q(x) \wedge \neg P(x) \wedge \neg S(x)$

H. množinově: $U \setminus (P \cup Q \cup S) = (U \setminus P \cap U \setminus Q \cap U \setminus S)$
   v jazyce PL1: $\neg(P(x) \vee Q(x) \vee S(x)) \Leftrightarrow \neg P(x) \wedge \neg Q(x) \wedge \neg S(x)$

2.6 Řešení pomocí Vennových diagramů

Vennovy diagramy jsou tedy grafická zobrazení množin a relací mezi nimi. Jednotlivé predikáty jsou vyjádřeny jako oblasti (množiny) v diagramu a vztahy mezi nimi jsou znázorněny překryvem těchto oblastí.

Nejprve se zaměříme na samotnou grafickou reprezentaci úsudků pomocí Vennových diagramů. Abychom pokryli všechny možné kombinace predikátů (*S, P, M* v případě kategorického sylogismu), zakreslíme kruhy zastupující predikáty tak, aby každý protínal všechny již existující průniky. Universum je ohraničeno obdélníkem a je označeno symbolem Ω.



Poté znázorníme situaci, kdy jsou premisy pravdivé, a to v tomto pořadí:

1. Vyšrafujeme plochy, které odpovídají prázdným třídám objektů.
2. Označíme křížkem plochy, které jsou jistě neprázdné (křížek přitom klademe jen tehdy, když jeho umístění je jednoznačné, tj. neexistuje jiná plocha, kam by mohl být umístěn).
3. V případě, že křížek z předchozího důvodu nelze umístit (jeho umístění není jednoznačné), klademe do diagramu na příslušné plochy otazníky.

Nakonec ověříme, zda vzniklá situace znázorňuje pravdivost závěru [5].

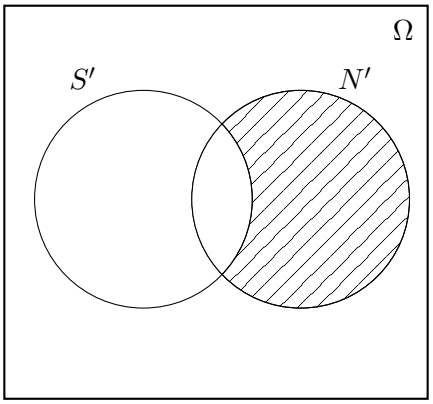
Jednoduchým příkladem může být následující úsudek:

P_1: Všichni netopýři jsou savci.
P_2: Existují netopýři.
$\therefore$ Existují savci.

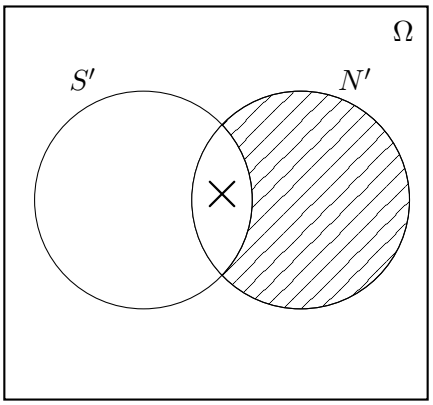
Formalizace:

$$P_1 : \forall x[N(x) \supset S(x)]$$
$$P_1 : \exists x[N(x)]$$
$$\therefore \exists x[S(x)]$$

První premisa říká, že všichni netopýři musí nutně být savci.



Podle druhé premisy existuje alespoň jeden netopýr.



Úsudek je **platný**, protože závěr říká, že v množina S musí být neprázdná.

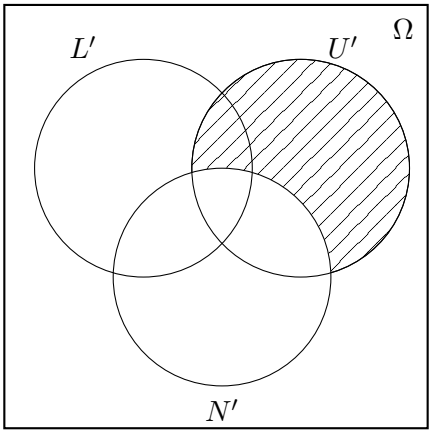
Příklad řešení úsudku

P_1: Všichni umělci jsou nadaní.
P_2: Někteří lidé nejsou nadaní.
$\therefore$ Někteří lidé nejsou umělci.

Formalizace:

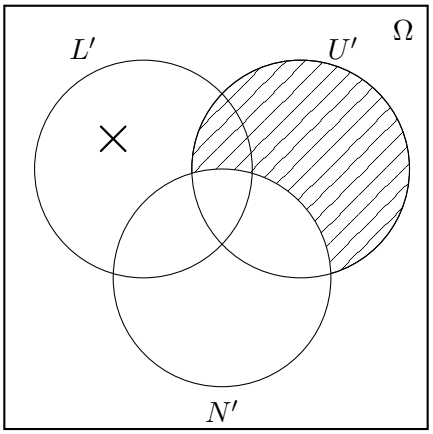
$$P_1 : \forall x[U(x) \supset N(x)]$$
$$P_2 : \exists x[L(x) \wedge \neg N(x)]$$
$$\therefore \exists x[L(x) \wedge \neg U(x)]$$

První premisa říká, že neexistují prvky, které by byly v množině označené U, ale nebyly v množině označené N. V našem diagramu vyšrafujeme oblast odpovídající této prázdné třídě objektů.



Druhá premisa říká, že množina prvků, které jsou v množině L, ale nejsou v množině N je neprázdná. Křížkem označíme neprázdnost této plochy, jelikož je jeho umístění jednoznačné.

Závěr vyžaduje, aby některé prvky v množině L nespadaly do množiny N. Úsudek je platný, protože závěr vyplývá z premis (případ sylogismu, modu baroco).



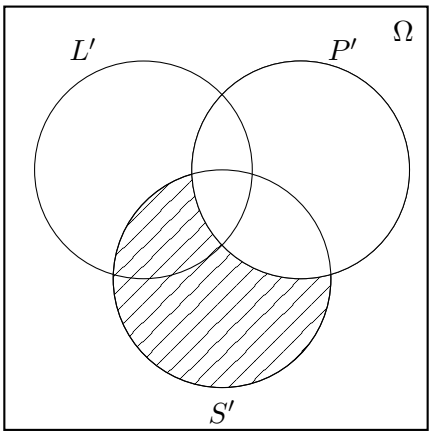
Příklad řešení **neplatného** úsudku

$$\begin{array}{l} P_1\text{: Všechny sovy jsou ptáci.} \\ \underline{P_2\text{: Někteří ptáci umí létat.}} \\ \therefore \text{Některé sovy umí létat.} \end{array}$$

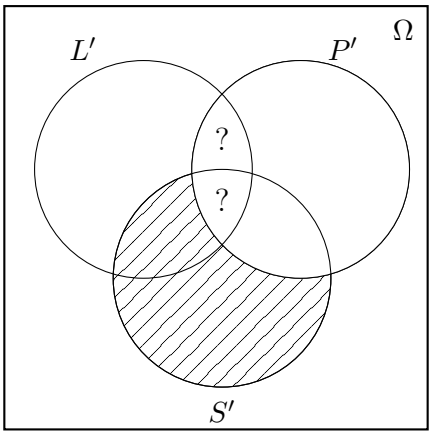
Formalizace:

$$\begin{array}{l} P_1 : \forall x[S(x) \supset P(x)] \\ \underline{P_2 : \exists x[P(x) \wedge L(x)]} \\ \therefore \exists x[S(x) \wedge L(x)] \end{array}$$

První premisa říká, že neexistuje sova, která by nebyla pták. Zaznačíme odpovídající prázdné množiny.



Druhá premisa sice říká, že je průnik množin P a L neprázdný, nevíme však kam umístit křížek. Závěr vyžaduje neprázdnost průniku množin S a L. Úsudek je **neplatný**.



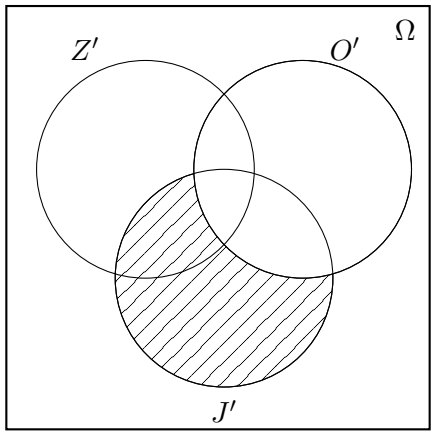
Dalším vhodným příkladem je případ, kdy jsou premisy i závěr kvantifikovány všeobecným kvantifikátorem. Například úsudek:

P_1: Jablka jsou ovoce.
P_2: Ovoce je zdravé.
$\therefore$ Jablka jsou zdravá.

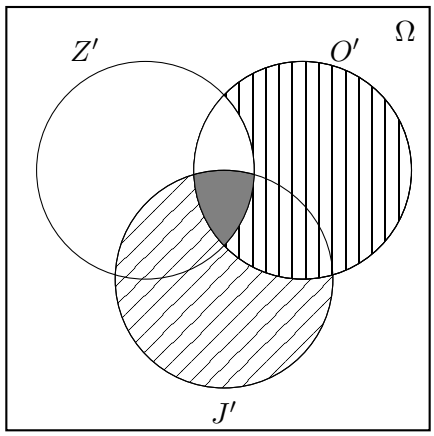
Formalizace:

$$P_1 : \forall x[J(x) \supset O(x)]$$
$$P_1 : \forall x[O(x) \supset Z(x)]$$
$$\therefore \forall x[J(x) \supset Z(x)]$$

První premisa říká, že neexistuje jablko, které by nebylo ovoce.



Podle druhé premisy neexistuje nezdravé ovoce.

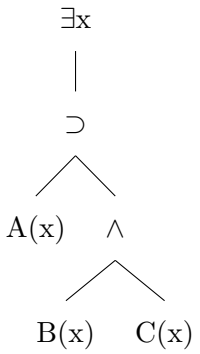


Úsudek je **platný**, protože závěr říká, že všechny prvky množiny J také patří do množiny Z. Tato oblast je v diagramu vyznačena tmavě.

2.7 Syntaktická analýza

Pro správné vyhodnocení problému, je potřeba ze vstupních řetězců udávajících premisy získat vhodnou datovou strukturu, nad kterou dále budou provedeny logické operace. Na této struktuře bude posléze stavěno v implementační části této práce. Způsob, kterým bude vstupní text převáděn na tuto strukturu se nazývá syntaktická analýza.

[7, s. 8] Syntaktická analýza (parsing, syntax analysis) spočívá v sestavování lexikálních jednotek ze zdrojového textu do gramatických frází, které analyzátor používá pro syntézu výstupu. Gramatické fráze vstupního řetězce se obvykle reprezentují **derivačním stromem** (orientovaný strom, který reprezentuje strukturu řetězce podle pravidel bezkontextové gramatiky) obdobným tomu na následující ilustraci pro predikát $\exists x[A(x) \supset (B(x) \wedge C(x))]$:



(2.4)

Při syntaktické analýze a tokenizaci vstupního řetězce se obvykle využívá jednoho ze dvou základních přístupů. Analýze shora dolů, nebo zdola nahoru. Těmto přístupům také odpovídají dva typy gramatik, zkratkou LL a LR gramatiky. Implementačně jsou znatelně jednodušší LL gramatiky. Při vyšších požadavcích na analýzu bývají LR i LL gramatiky generovány automatizovanými nástroji.

Při syntaktické **analýze shora dolů** vycházíme z kořene derivačního stromu, startovacího symbolu gramatiky a snažíme se postupnou expanzí neterminálních symbolů vstupu dospět k terminálním symbolům gramatiky. Tento postup lze popsat jako proces hledání levé derivace vstupního řetězce.

Naivním a značně neefektivním řešením by bylo postupně použít všechna pravidla gramatiky a provádět návraty do bodu, ze kterého lze pokračovat v případě, že se pravidlo nepovede použít. Lepší implementací je sepsání rekurzivních procedur pro jednotlivá pravidla gramatiky a následná analýza neterminálů gramatiky. Tomuto přístupu se také říká **analýza rekurzivním sestupem** [7, s. 37]. Při samotné syntaktické analýze jsou postupně, rekurzivně použita pravidla bezkontextové

gramatiky.

Tento postup se obvykle používá pro bezkontextové gramatiky typu LL(k), kdy syntaktický analyzátor prochází vstup zleva doprava a vytváří levou derivaci. Symbol „k“ označuje počet symbolů, které syntaktický analyzátor musí přečíst pro rozhodnutí pravidla gramatiky, které dále použije. Tento nástroj implementuje LL(1) gramatiku, kdy k výběru následujícího pravidla stačí přečtení jednoho znaku.

2.7.1 Gramatika

LL(k) gramatika nesmí pro implementaci analýzy rekurzivním sestupem obsahovat levou rekurzi, jelikož v případě levé rekurze nelze jednoznačně určit pravidlo, které má syntaktický analyzátor použít. V případě nutnosti volby mezi dvěma vzájemně se nevylučujícími neterminály, syntaktický analyzátor dochází do situace, kdy je nutno zvážit veškeré možné cesty gramatikou a může dojít k zaplnění paměti. Je tedy nutno levou rekurzi odstranit vhodným přepsáním nebo odstraněním pravidel gramatiky.

Následující pravidla jsem převzal z [7, s. 42].

Nechť G je bezkontextová gramatika. Pravidlo A gramatiky G je zleva rekurzivní, pokud je ve tvaru

$$A \to A\alpha_1 \mid A\alpha_2 \mid ... \mid A\alpha_n \mid \beta_1 \mid ... \mid \beta_m \tag{2.5}$$

kde řetězce β_i nezačínají neterminálem A. Takové pravidlo můžeme přepsat zavedením nového neterminálu A’ jako

$$\begin{aligned} A &\to \beta_1 A' \mid \beta_2 A' \mid ... \mid \beta_m A' \\ A' &\to \alpha_1 A' \mid \alpha_2 A' \mid ... \mid \alpha_n A' \mid \epsilon \end{aligned} \tag{2.6}$$

Dále, aby naše gramatika byla LL(1) gramatikou, „*nemůže více než jedna pravá strana pravidla začínat týmž řetězcem terminálních symbolů, t.j. má-li pravidlo tvar*“:

$$A \to \beta\alpha_1 \mid \beta\alpha_2 \mid ... \mid \beta\alpha_n \tag{2.7}$$

můžeme je „vytknout“ zavedením nového neterminálu A’ s pravidly

$$\begin{aligned} A &\to \beta A' \\ A' &\to \alpha_1 \mid \alpha_2 \mid ... \mid \alpha_n \end{aligned} \tag{2.8}$$

Kapitola 3

Návrh a analýza

3.1 Cíl práce

Cílem práce je navrhnout nástroj na dokazování PL1 pomocí Vennových diagramů.[1] Uživatel zadá premisy a závěr a nástroj vyhodnotí pravdivost problému a vykreslí Vennův diagram. Dále, program musí podporovat odkrokování problému a interaktivní mód pro kontrolu výsledku zadaného uživatelem.

3.2 Použité technologie

Nástroj je implementován formou API pro ověření pravdivosti dle zadaného vstupu a prezentační vrstvy ve formě webového aplikace. Prezentační vrstva pouze reprezentuje interpretaci výsledku.

3.2.1 Python

Po zvážení mnoha platforem byla aplikační vrstva nástroje implementována jako API v interpretovaném jazyce Python 3. Ten se nabízí jako vhodná volba pro tvorbu webových aplikací díky své bohaté standardní knihovně a množství knihoven pro tvorbu API. Aplikační vrstva je na straně serveru ve snaze obejít některé z problémů JavaScriptu, zejména slabé typování a zrychlit načítání aplikace.

3.2.1.1 FastAPI

Přenos dat mezi serverem a klientem je zajištěn FastAPI. Jedná se o rámec (framework) pro intuitivní tvorbu API. Mezi jeho výhody patří automatické ověření dat pomocí typových anotací, modularita a rychlost. Umožňuje nám také asynchronní programování – zpracování více požadavků současně, bez zbytečného čekání na dokončení požadavků.

[1] Funkční instance nástroje dočasně běží na `http://130.162.49.62:8080/`

Asynchronní programování není pro tento nástroj zcela nutné, jednotlivé požadavky se zpracují velmi rychle. Mělo by však v případě většího vytížení zlepšit výkon a nepředstavuje v kódu žádné komplikace.

3.2.2 Vue.js

Prezentační vrstva nebo také klientská část nástroje je implementována jako webová aplikace v rámci Vue.js. Jedná se o aktivně vyvíjený JavaScriptový framework pro tvorbu rychlých a škálovatelných webových aplikací.

Vue.js jsem zvolil zejména kvůli nativní podpoře komponent, které nástroj využívá pro zobrazování diagramů. Každý diagram je potom pouze instancí komponenty, což umožňuje považovat jednotlivé diagramy za samostatné sekce bez rizika vzájemné interakce.

Vue.js aplikaci lze zabalit do samostatného balíčku, který na straně serveru nevyžaduje instalaci závislostí, což bylo jedním z požadavků této práce. Pro jeho spuštění je zapotřebí HTTP server.

3.2.3 SASS

Pro deklaraci vzhledu nástroje je využit CSS preprocesor SASS. Disponuje striktnější a přehlednější syntaxí a konvencemi pro tvorbu čistějšího, lépe škálovatelného CSS kódu.

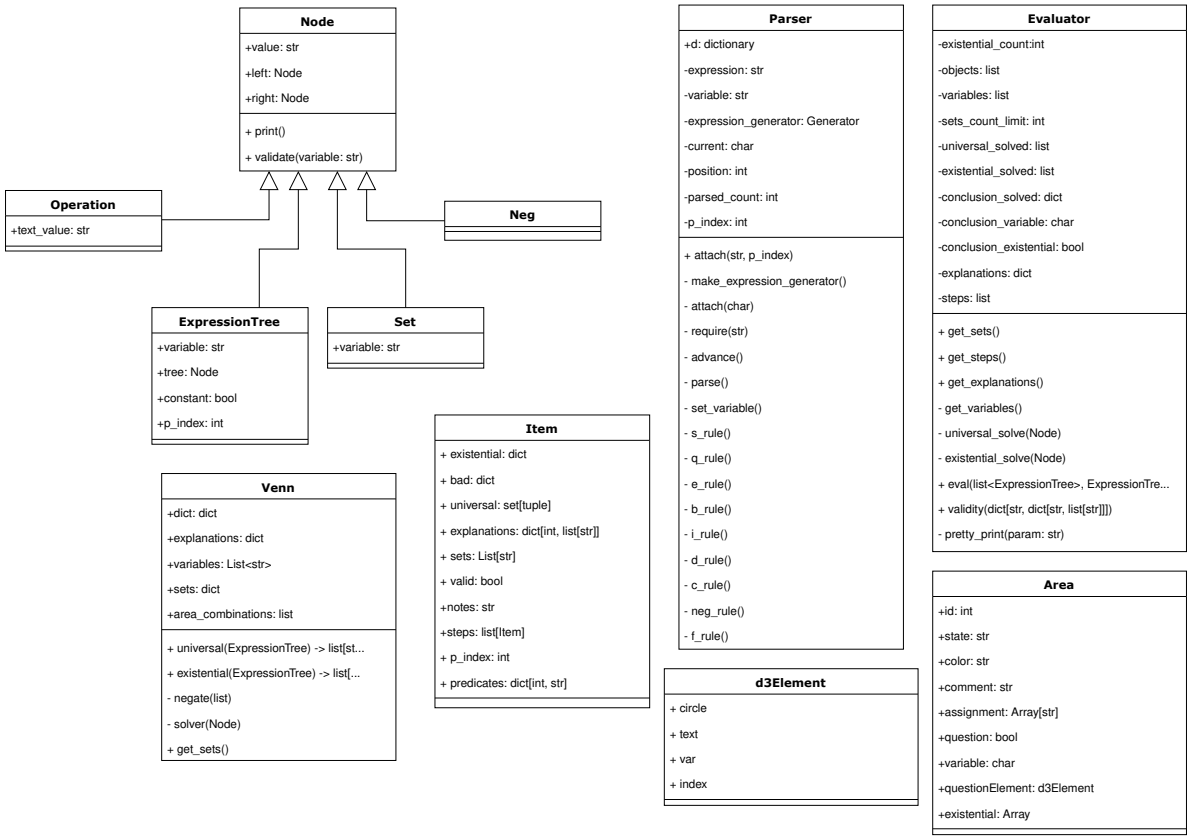
3.2.4 D3.js

Co se samotné vizualizace Vennových diagramů týče, zvolil jsem JavaScriptovou knihovnu D3.js. Oproti mnoha specializovaným variantám této a jiných knihoven, které jsou zaměřeny na Vennovy diagramy, D3.js umožňuje výrazně vyšší úroveň přizpůsobitelnosti. Pro vykreslení diagramů jsou nám k dispozici funkce vektorové grafiky, s jejichž pomocí definujeme samotné diagramy. Z důvodu chybějící funkcionality, zejména pro zaznačení existence prvků, bylo nutno implementovat všechny potřebné diagramy samostatně. Toto prodloužilo dobu vývoje nástroje, ale výsledkem jsou vizuálně kvalitnější výsledky.

Diagramy jsou vykreslovány vektorovou grafikou, konkrétně ve formátu SVG. Ten je pro menší počet detailních prvků vhodnější než HTML5 Canvas, který lze s pomocí rozšíření D3.js také použít. D3.js umožňuje generování SVG prvků pomocí vlastních funkcí.

3.3 Třídní diagram

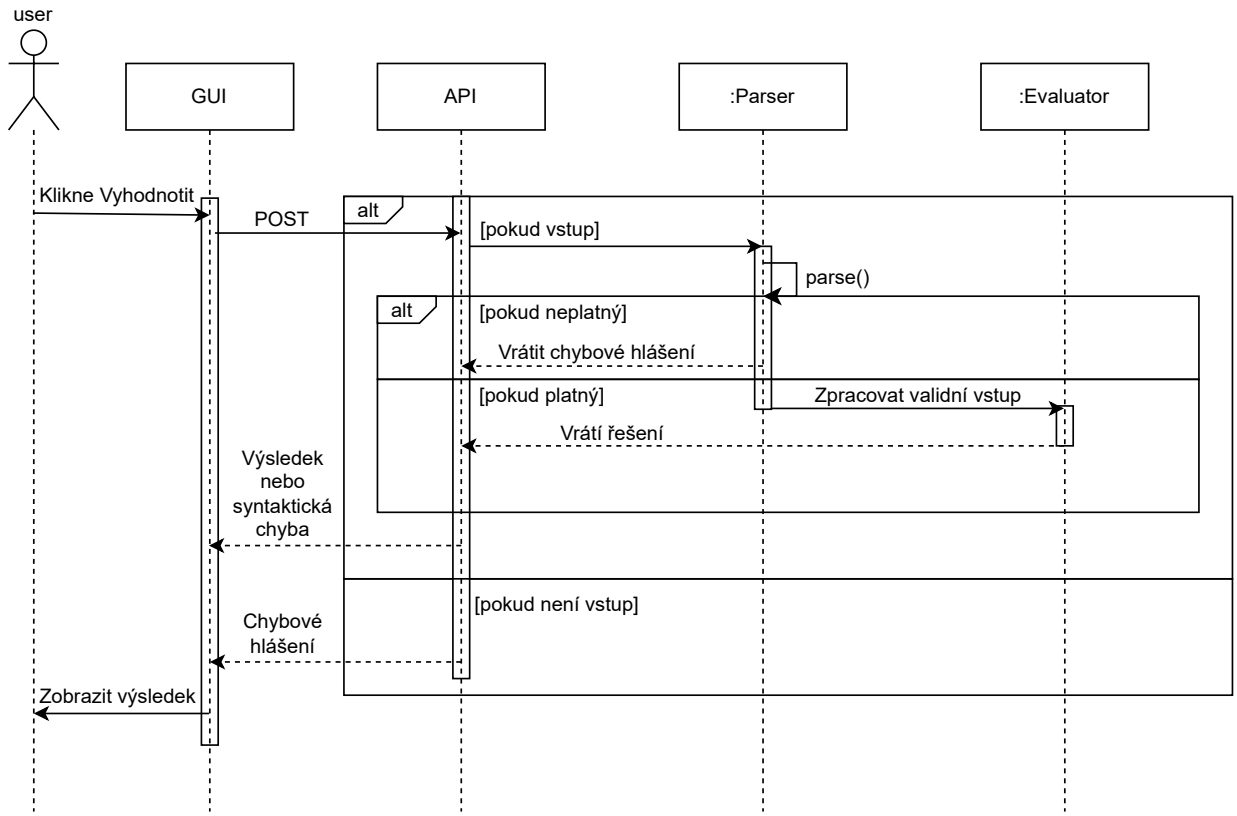
Na obrázku 3.1 lze vidět třídní diagram nástroje. Zahrnuje veškeré třídy v projektu, včetně všech pomocných tříd. Třídy *Area*, *d3Element* a *Venn* jsou třídy prezentační vrstvy. Třída Item reprezentuje datový typ, který je v serializované formě použit pro přenos výsledků mezi API a prezentační vrstvou.



Obrázek 3.1: Třídní diagram

3.4 Sekvenční diagram

Pro demonstraci běhu nástroje může posloužit následující sekvenční diagram. Je jím zachycen proces zobrazení výsledku zadaného úsudku.



Obrázek 3.2: Sekvenční diagram

3.5 Vymezení podmnožiny jazyka

Než se zaměřím na samotné implementační detaily nástroje, je potřeba vymezit podmnožinu predikátové logiky, se kterou bude pracovat. Omezení a specifika zvolena pro implementaci jsou následující:

- Formule musí být uzavřené.
- Ve formuli se musí vyskytovat právě jedna proměnná, nebo konstanta.
- Podporovány jsou pouze unární predikátové symboly.
- Predikátové symboly mohou být nahrazeny celými slovy. Musí začínat velkým písmenem. Validní jsou např. „C“, „Člověk“.

- Logické spojky lze ve vstupním řetězci nahradit alternativními symboly [3.1]
- Funkční symboly nejsou podporovány (pouze symboly konstant).
- Lze používat hranaté a jednoduché závorky.

Symbol	$\neg$	$\wedge$	$\vee$	$\supset$	$\equiv$	$\forall$	$\exists$
Alternativa	!	&	|	>	<>	A	E

Tabulka 3.1: Alternativní symboly

Před zahájením syntaktické analýzy jsou z jednotlivých premis odstraněny bílé znaky a logické operace z tabulky 3.1 jsou převedeny na ty v řádku *Alternativa*. Gramatika, kterou budu v implementaci využívat vypadá po patřičných úpravách zmíněných v kapitole 2.7.1 následovně:

$$
\begin{aligned}
\rightarrow vyraz ::=\ & all\ male\ \text{"["}\ E\ \text{"]"}\ \mid\ exist\ male\ \text{"["}\ E\ \text{"]"}\ \mid\ E \\
E ::=\ & I \mid I\ \text{" <> "}\ E \\
I ::=\ & D \mid D\ \text{" > "}\ I \\
D ::=\ & C \mid C\ \text{"|"}\ D \\
C ::=\ & N \mid N\ \text{"&"}\ C \\
N ::=\ & literal \mid \text{"!"}\ literal \\
literal ::=\ & \text{"("}\ E\ \text{")"} \mid slovo\ \text{"("}\ male\ \text{")"} \mid slovo\ \text{"("}\ konst\ \text{")"} \\
velke ::=\ & \text{"}A\text{"} \mid \text{"}B\text{"} \mid ... \mid \text{"}Y\text{"} \mid \text{"}Z\text{"} \\
male ::=\ & \text{"}h\text{"} \mid \text{"}i\text{"} \mid ... \mid \text{"}y\text{"} \mid \text{"}z\text{"} \\
konst ::=\ & \text{"}a\text{"} \mid \text{"}b\text{"} \mid ... \mid \text{"}f\text{"} \mid \text{"}g\text{"} \\
libovolne ::=\ & male \mid konst \\
all ::=\ & \text{"}\forall\text{"} \mid \text{"}A\text{"} \\
exist ::=\ & \text{"}\exists\text{"} \mid \text{"}E\text{"} \\
slovo ::=\ & velke, \{libovolne\}
\end{aligned}
\tag{3.1}
$$

Neterminál „slovo“ je definován jako velké písmeno následováno libovolným počtem malých písmen. Pro konstanty jsou rezervována malá písmena od „a“ do „g“, pro proměnné malá písmena od „h“ do „z“.

Jednou z výhod rekurzivního sestupu je přesné modelování gramatiky, což umožňuje poměrně jednoduše určit pořadí (prioritu), ve kterém mají být jednotlivé symboly vyhodnoceny pomocí rekurzivního volání příslušných funkcí.

Dále je provedena sémantická analýza, která zajistí mj. výskyt právě jedné proměnné, nebo právě jedné konstanty v každé ze vstupních formulí a odpovídající kvantifikaci.

3.6 Syntaktická analýza

Jak již bylo zmíněno v teoretické části práce, je žádoucí převést vstup na vhodnou datovou strukturu. V našem případě se jedná o derivační strom. V nástroji je implementován LL(1) syntaktický analyzátor (viz kapitola 2.7.1), který je modelován podle gramatiky [3.1]. Při jeho implementaci jsem využil článku [2], který mi poskytnul základní inspiraci pro tvorbu syntaktického analyzátoru. Každé pravidlo gramatiky odpovídá jedné metodě třídy *Parser*. Instance analyzátoru vyžaduje připojení řetězce, který bude analyzován, a vrací stromovou strukturu. Jednotlivé uzly derivačního stromu jsou reprezentovány instancemi dědičné struktury *Node*, viz třídní diagram 3.1. Třídy *Operation, Neg, Set* a samotný strom odpovídají jedné z těchto tříd.

Analyzátor postupně prochází vstupní řetězec a volá metodu počátečního neterminálu. Tím se skrze zbylé metody dostane k poslednímu pravidlu gramatiky, v případě naší gramatiky se jedná o pravidlo *literal*. Metoda match() zjišťuje, zda aktuální znak odpovídá znaku, který očekáváme v rámci dané metody. Pokud znak odpovídá, je vrácena nová instance patřičné třídy. V opačném případě metoda vrátí hodnotu None a proces se opakuje pro metodu, která aktuální metodu původně volala.

Pro případ, kdy vyžadujeme, aby určitý znak následoval jiný (například levá závorka „(“ musí mít odpovídající pravou „)“, ekvivalence se značí dvěma znaky „<>“, a je potřeba, aby se bezprostředně následovaly), je implementována metoda require(). Tato metoda se chová identicky jako metoda match(), pouze v případě špatného znaku vyvolá výjimku.

```
def __d_rule(self) -> Operation | None:
    left = self.__c_rule()
    if not left:
        return None
    if self.__match('|'):
        right = self.__d_rule()
        if right:
            return Operation(left, right, 'or')
    return left
```

Listing 3.1: Příklad metody pro neterminál gramatiky

Po rozparsování získáme stromovou strukturu, kterou však musíme zkontrolovat. Například v situaci, kdy by byla na vstupu premisa s neshodujícími se proměnnými, např. $\forall x[A(y)]$, musíme zajistit vyvolání chyby. Na konci tohoto procesu budeme mít pole syntaktických stromů validních premis, které můžeme dále zpracovat.

3.7 Proces řešení

Tento nástroj se zabývá vizualizací Vennových diagramů o maximálním rozsahu čtyř množin, které odpovídají predikátovým symbolům. Samotné ověření platnosti (bez vizualizace) není nutné nějakým způsobem omezovat, jelikož jediným limitujícím faktorem pro jeho ověření je výpočetní složitost této operace. Nástroj je však přesto na úrovni API omezen na 4 množiny z důvodu potenciálního zneužití jeho přetížením.

Algoritmus řešení byl již zmíněn v teoretické části. Postup je následující:

1. Všechny premisy kvantifikované všeobecným kvantifikátorem jsou provedeny jako první. Cílem tohoto kroku je získat „vyšrafované“ oblasti diagramu, tedy prázdné třídy objektů.

   - Stromová struktura, která reprezentuje premisu, je procházena od svých listů, které byly převedeny na množiny prvků odpovídajících oblastem výsledného diagramu.
   - Vnitřní uzly stromu jsou operacemi nad těmito množinami. Jsou přepsány na ekvivalentní operace v množinové notaci (kapitola 2.4) a operace jsou provedeny.
   - Algoritmus končí v kořenovém uzlu stromu a vrací výslednou množinu, která je platná pro danou předmětovou proměnnou.

2. Poté jsou vyřešeny premisy kvantifikované existenčním kvantifikátorem. Tímto krokem získáme množiny ploch, které jsou jistě neprázdné. Je použit identický algoritmus.

3. Následně jsou obdobně vyřešeny také premisy s konstantami a závěr.

Po tomto kroku jsme schopni vykreslit Vennův diagram, včetně všech kroků, které k jeho reprezentaci vedly. Datová struktura je posléze v prezentační vrstvě přímo přepsána na odpovídající diagram. Je však stále potřeba rozhodnout, zdali je daný úsudek platný.

3.7.1 Platnost úsudku

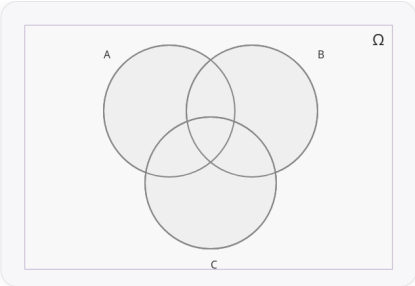
Platnost úsudku je ověřována metodou *validity()*. Metoda nejdříve ze závěru extrahuje předmětovou proměnnou. Pokud nebyla zadána žádná premisa, je proces ukončen patřičným chybovým hlášením. Poté je porovnána množina vyšrafovaných oblastí diagramů s množinou závěru. V případě, že je vyškrtána oblast, která by dle závěru neměla, je úsudek neplatný. Pokud závěr vyžaduje existenci prvků, jsou nejprve ověřeny konstanty. Poté jsou otestovány ostatní prvky, odpovídá-li jejich předmětová proměnná. Pokud se v oblastech daným závěrem nevyskytují prvky, je úsudek neplatný.

V případě, že metoda zpracovala všechny premisy a nenašla žádné chyby, je úsudek platný. Výsledkem je pole vysvětlující každý krok, které je také posláno do prezentační vrstvy a zobrazuje se nad kroky výsledku včetně závěru.

3.8 Vizualizace a interaktivita Vennových diagramů

V této kapitole a implementaci jsem čerpal z [11] [8] [6].

Nyní se zaměřím na vykreslování Vennových diagramů pomocí knihovny D3.js. Pro vykreslení kružnice jsou zapotřebí pouze souřadnice jejího středu a poloměr.



Obrázek 3.3: Základní Vennův diagram

Lze tedy jednoduše vykreslit kružnice. Tento stav však není zdaleka dostačující, jelikož se každý průnik těchto kružnic musí chovat jako samostatný geometrický útvar. V opačném případě by ho nebylo možno vyšrafovat. Pro výpočet průsečíků jsou zapotřebí opět pouze souřadnice středů kružnic a jejich poloměry. Pro výpočet souřadnic průsečíků kružnic je vhodné využít vzorce, který lze nalézt například na stránkách [11].

```
const getPointsOfIntersection = (firstX, secondX, firstY, secondY, flip) => {
    let d = Math.sqrt((firstX - secondX) ** 2 + (firstY - secondY) ** 2);
    let sx = firstX + ((d/2)/d)*(secondX - firstX);
    let sy = firstY + ((d/2)/d)*(secondY - firstY);
    let v = Math.sqrt((vennRadius ** 2) - (d/2) ** 2);
    if (flip)
      return {
        "x": (sx + (v/d)*(firstY - secondY)),
        "y": (sy - (v/d)*(firstX - secondX))
      }
    return {
      "x": (sx - (v/d)*(firstY - secondY)),
      "y": (sy + (v/d)*(firstX - secondX))
    }
}
```

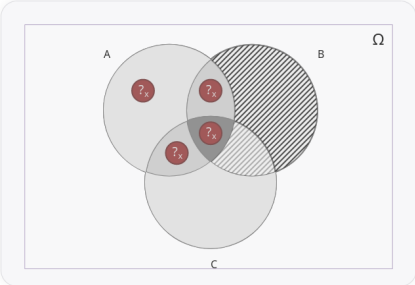
Listing 3.2: Metoda pro výpočet průsečíků dvou kružnic

Knihovna D3.js pracuje s formátem SVG, který umožňuje definovat kruhy a elipsy pomocí Bézierových křivek nebo kruhových oblouků (angl. arc). S ohledem na to, že se poloměr kružnic nemění a známe body počátku a konce kruhového oblouku, je využití SVG funkce Arc intuitivnější.

Funkce Arc má několik parametrů, pro naše účely jsou však dostačující parametry „sweep-flag“ (určující směr vykreslování, a to do kladných, nebo záporných úhlů) a „large-arc-flag“. Ten nám umožní vykreslit všechny potřebné kruhové oblouky. Následuje Arc příkaz z [6]:

$$A\ rx\ ry\ \ xAxisRotation\ \ largeArcFlag\ \ sweepFlag\ x\ y$$

Po výpočtu průniků kružnic lze tedy vykreslit dané průsečíky. Obdobně jsou znovu vykresleny oblasti kružnic, nyní bez jejich průsečíků se zbylými kružnicemi. Dále jsou těmto oblastem, které se nyní nachází v DOM prohlížeče, pomocí D3.js nastaveny posluchače (angl. EventListener), které při kliknutí vyšrafují danou oblast, nebo do ní vloží křížek.



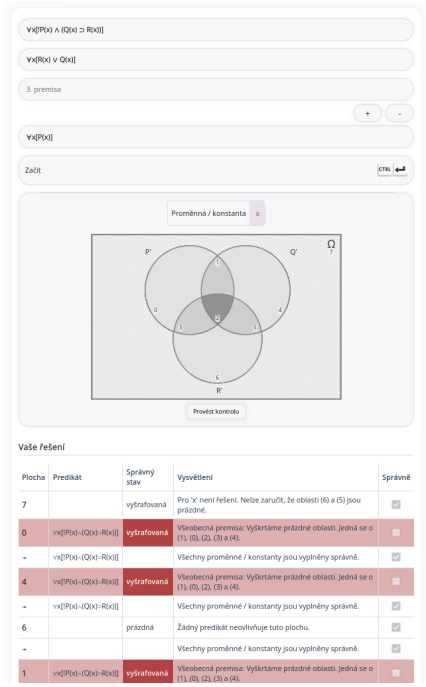
Obrázek 3.4: Diagram v případě, že není možno umístit křížek

Diagramy jsou nyní vykresleny a jejich oblasti jsou interaktivní. API prezentační vrstvě v objektu *Item* zprostředkovává pole objektů stejného typu korespondující ke krokům řešení úsudku. Ty nástroj poté individuálně vykresluje. Tento přístup umožňuje využít stejné Vue.js komponenty pro výsledek i kroky.

3.9 Interaktivní ověření platnosti

Programu lze využít pro procvičování řešení úsudků v interaktivním módu. Oproti tomu standardnímu, který zobrazuje vyřešený diagram, interaktivní mód zobrazuje prázdný Vennův diagram. Uživateli je poté dána možnost ručně diagram upravit. V diagramu lze levým kliknutím myši vyškrtat plochu, pravým lze přidat křížek s proměnnou danou vstupním polem nad diagramem.

Po dokončení úprav diagramu může uživatel stisknutím tlačítka „Provést kontrolu“ získat tabulku se všemi plochami diagramu a informací o tom, zda byly plochy zadány správně. Součástí výstupu je také správný diagram pro daný vstup.



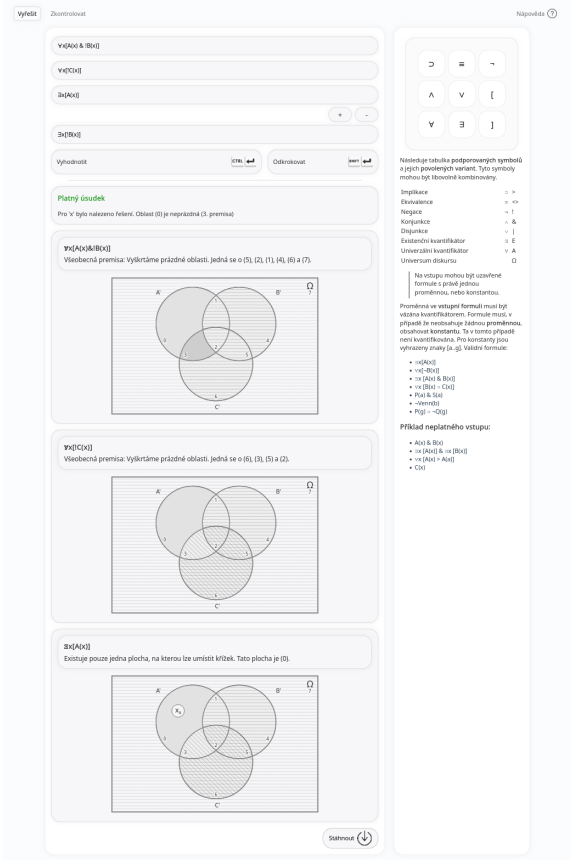
Obrázek 3.5: Interaktivní řešení

3.10 Uživatelské rozhraní

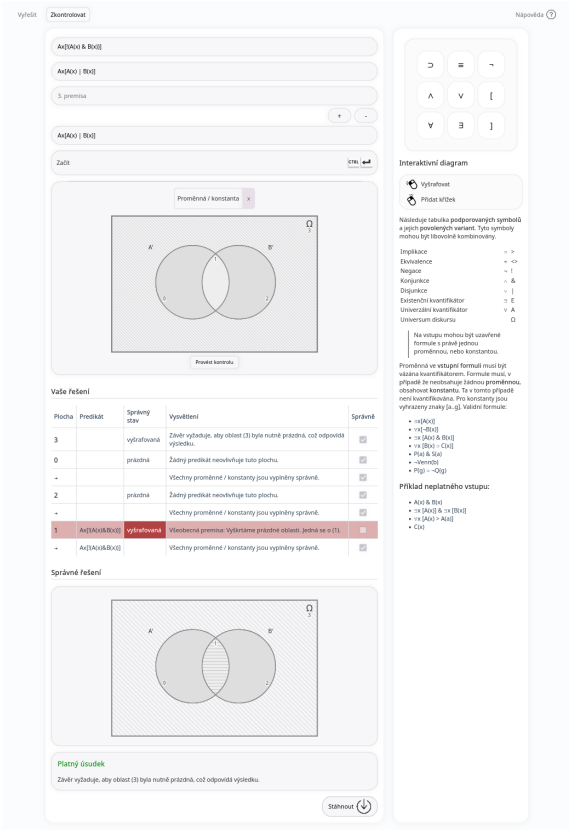
Uživatelské rozhraní je rozděleno do dvou karet reprezentující jednotlivé módy řešení. Ve vrchní části se nachází vstupní pole pro zadání premis a závěru společně s tlačítky pro řešení a zobrazení kroků, které nástroj provedl.

Na pravé straně nástroje se nachází virtuální klávesnice pro zadání logických znaků společně s nápovědou pro správné použití nástroje a pochopení jeho gramatiky. Pro jednodušší použití nástroj podporuje také ovládání pomocí klávesových zkratek (Ctrl + Enter, Shift + Enter pro zobrazení řešení resp. kroků). Následují samotné diagramy, resp. tabulka se správným řešením a tlačítko pro vytištění/stažení výsledku.

Následuje příklad možného výstupu při zpětném řešení jednoduchého příkladu PL1.



(a) Zobrazení kroků řešení



(b) Uživatelské rozhraní při interaktivním řešení

Obrázek 3.6: Uživatelské rozhraní

Kapitola 4

Závěr

Cílem této bakalářské práce bylo implementovat nástroj pro ověřování platnosti úsudků pomocí Vennových diagramů. V první části práce byla popsána nezbytná teorie týkající se výrokové a predikátové logiky 1. řádu. Dále byla představena metoda pro ověřování platnosti úsudků pomocí Vennových diagramů a syntaktická analýza vstupních řetězců.

Druhá část práce se zabývá návrhem a implementací nástroje. Byla definována podmnožina predikátové logiky, se kterou nástroj pracuje, a gramatika odpovídající tomuto jazyku.

Nástroj pro ověřování úsudků se povedlo vytvořit. Pro jeho implementaci byl využit framework Vue.js, vizualizační knihovna D3.js a API v jazyce Python. Nástroj na základě vstupního úsudku generuje informaci o jeho platnosti, diagram zobrazující řešení a jednotlivé kroky, které k výsledku vedly. Dále poskytuje interaktivní mód pro ověření řešení vloženého uživatelem pomocí tabulky.

Literatura

[1] A. W. F. Edwards. *Cogwheels of the Mind: The Story of Venn Diagrams*. Baltimore, U.S.: The Johns Hopkins University Press, 2004.

[2] Eoin Davey. *Writing a simple recursive descent parser in Python*. 2018. URL: `https://vey.ie/2018/10/04/RecursiveDescent.html` (cit. 16. 04. 2023).

[3] doc. PhDr. Jiří Raclavský, Ph.D. *Úvod do logiky: klasická predikátová logika*. Brno: Masarykova univerzita, 2015. ISBN: 978-80-210-7965-6.

[4] doc. PhDr. Jiří Raclavský, Ph.D. *Úvod do logiky: klasická výroková logika*. Brno: Masarykova univerzita, 2015.

[5] doc. RNDr. Marie Duží, CSc. *Logika pro informatiky*. Ostrava: Ediční středisko VŠB-TUO, 2012.

[6] MDN Web Docs. *Paths*. 2023. URL: `https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial/Paths` (cit. 13. 04. 2023).

[7] Dr. Ing. Miroslav Beneš. *Překladače*. Ostrava. URL: `https://www.cs.vsb.cz/benes/vyuka/pjp/skr-mb.pdf`.

[8] Calder M. Myers. *How I made an interactive Venn diagram with d3*. 2019. URL: `https://medium.com/@cmmyers/how-i-made-an-interactive-venn-diagram-with-d3-fa723c55a148` (cit. 16. 04. 2023).

[9] Petr Vopěnka. *Úvod do klasické teorie množin*. Plzeň: Vydavatelství Západočeské univerzity v Plzni, 2011.

[10] RNDr. Antonín Sochor, DrSc. *Logika pro všechny ochotné myslet*. Praha: Univerzita Karlova v Praze, 2012. ISBN: 978-80-246-1959-0.

[11] WikiKnihy. *Numerický výpočet průniku dvou kružnic*. 2020. URL: `https://cs.wikibooks.org/wiki/Geometrie/Numerick%C3%BD_v%C3%BDpo%C4%8Det_pr%C5%AFniku_dvou_kru%C5%BEnic` (cit. 13. 04. 2023).

Příloha A

Uživatelská příručka

V této příloze se nachází návod pro použití nástroje.

A.1 Vstup

Nástroj pracuje s omezenou podmnožinou predikátové logiky a jeho vstupem musí být dobře utvořené formule jejího jazyka. Logické operace, které nástroj přijímá jsou následující:

Symbol	$\neg$	$\wedge$	$\vee$	$\supset$	$\equiv$	$\forall$	$\exists$
Alternativa	!	&	|	>	<>	A	E

Tabulka A.1: Alternativní symboly

Lze tedy zaměňovat logické znaky za alternativní symboly pro snažší zadávání vstupních řetězců.

Je nutno dbát na zadání správných proměnných. Pro konstanty jsou použita malá písmena od „a“ do „g“, pro proměnné malá písmena od „h“ do „z“. Dále je potřeba správně uzavřít závorky, zejména u již zmíněných proměnných, nebo při využití závorek pro změnu priority operací.

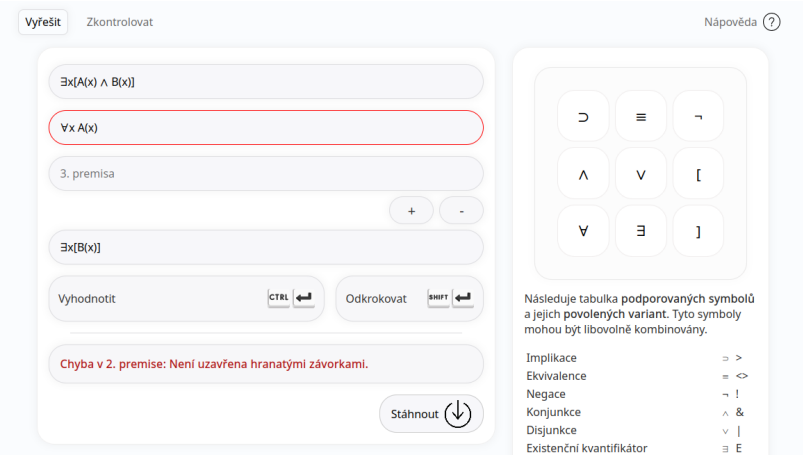
Příklady platného vstupu:

- $\forall x[A(x) \supset B(x)]$
- $\exists x[Vozidlo(x) \wedge \neg Automobil(x)]$
- $Ex[L(x)\ \&\ !U(x)]$
- $Student(a)$
- $\forall y[!P(y)\ \&\ (\ Q(y) > R(y))]$

Tato pravidla jsou také popsána na pravé liště v nástroji.

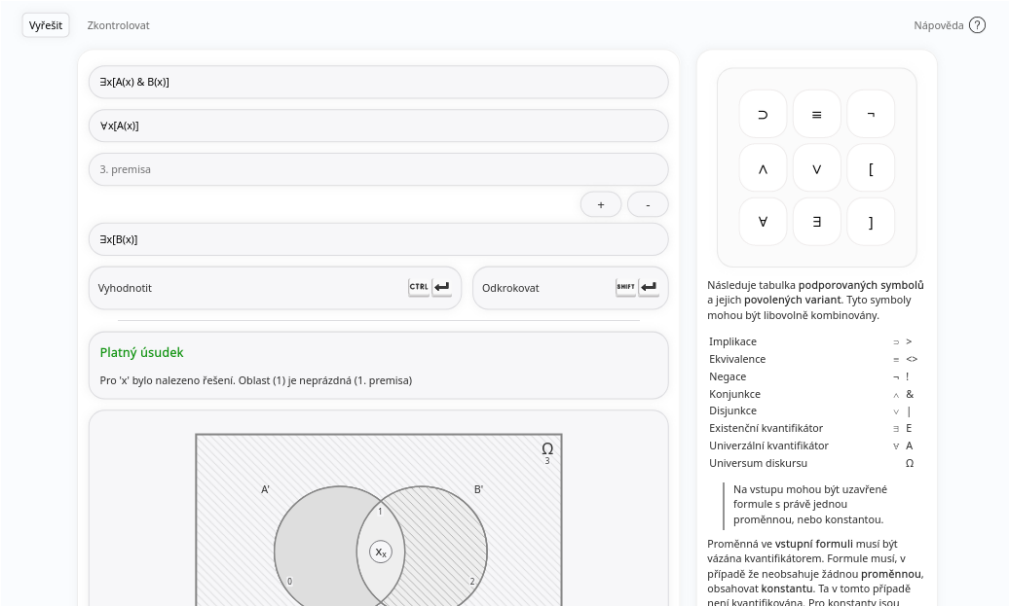
A.2 Vyhodnocení

Nástroj po načtení nabídne uživateli zadat premisy do číslovaných vstupních polí. Při jejich zadávání průběžně kontroluje syntaxi vstupu. V případě nalezení chyby ji zobrazí a vyznačí špatně zadanou premisu červenou barvou.



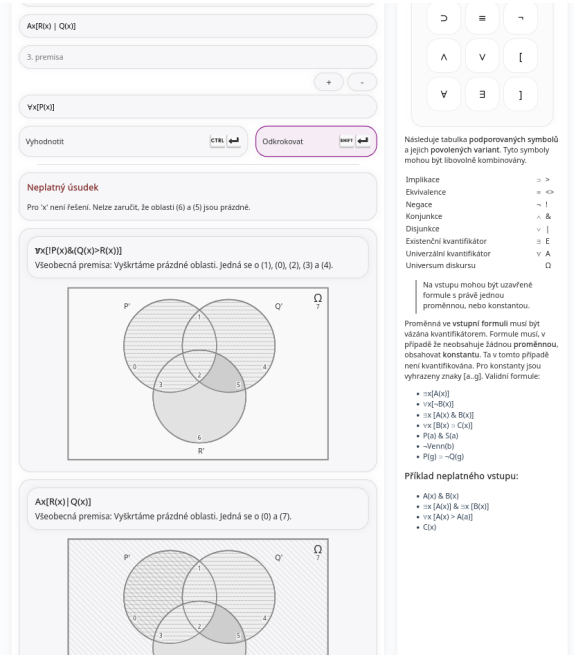
Obrázek A.1: Špatně zadaná premisa

Poté, co uživatel zadá veškeré premisy, může stisknout tlačítko „Vyhodnotit“ a nástroj vrátí vyřešený úsudek.



Obrázek A.2: Zadání premis

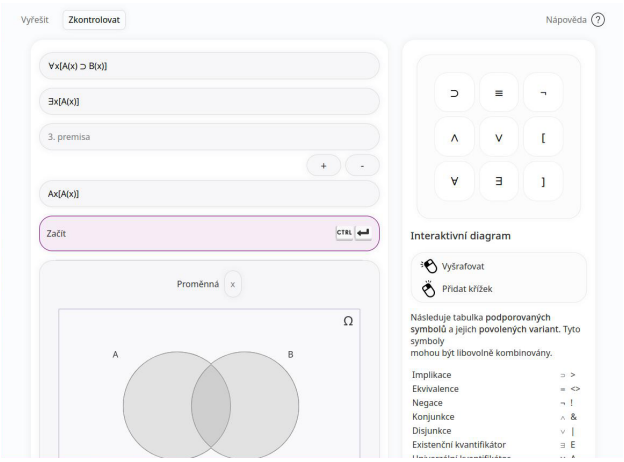
Pro zobrazení kroků, které vedly k výsledku lze stisknout tlačítko „Odkrokovat“.



Obrázek A.3: Kroky vedoucí k řešení

A.3 Interaktivní mód

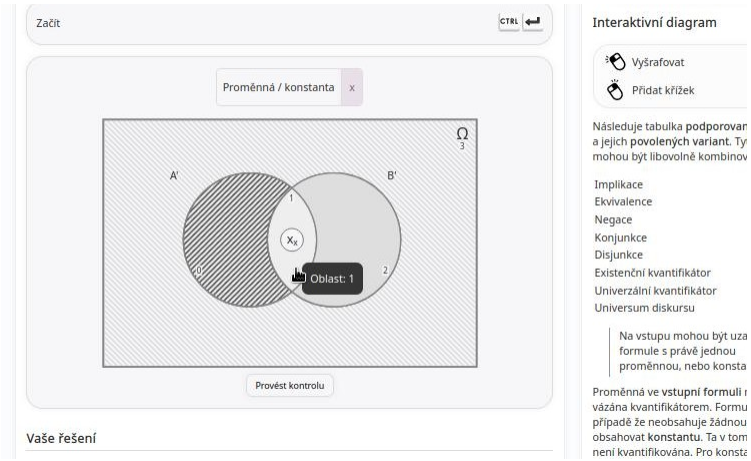
Pro kontrolu řešení v interaktivním módu je nejprve potřeba zadat úsudek.



Obrázek A.4: Interaktivní mód

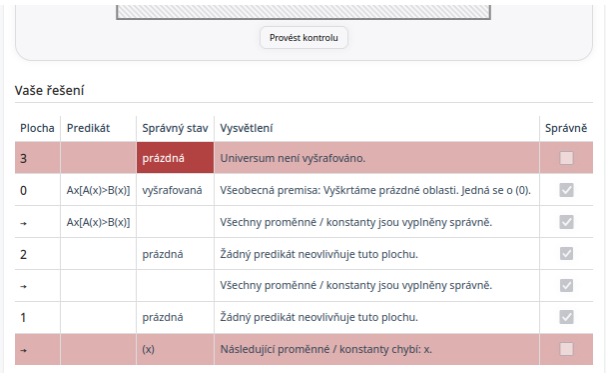
Po stisknutí tlačítka „Začít“ se vykreslí diagram, do kterého uživatel ručně zadá řešení. Levým tlačítkem myši se šrafují oblasti diagramu, pravým se do oblastí přidávají křížky. Pro zvolení proměnné, pro kterou křížek uživatel vkládá slouží vstupní pole označené „Proměnná“. V jedné oblasti může být křížek pro mnoho proměnných.

Po zadání řešení uživatel pokračuje stisknutím tlačítka „Provést kontrolu“, které zkontroluje vstup a vypíše tabulku všech oblastí diagramu vč. informace o správnosti řešení.

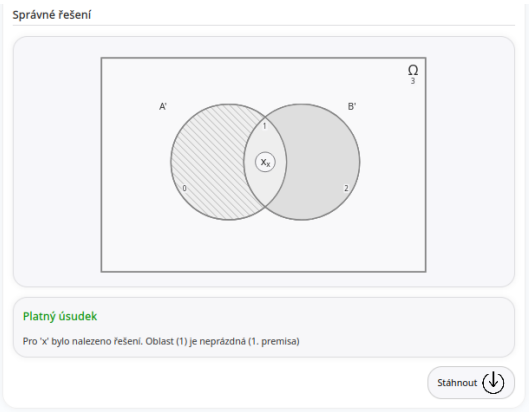


Obrázek A.5: Zadání řešení

Nástroj dále vykreslí správné řešení pro možnost porovnání s původním řešením zadaným uživatelem.



(a) Kontrola pomocí tabulky



(b) Uživatelské rozhraní při interaktivním řešení

Obrázek A.6: Srovnání výsledku

Příloha B

Návod pro spuštění

Pro spuštění instance nástroje je nejdříve potřeba stáhnout patřičné závislosti. Tento návod byl napsán pro Ubuntu 22.04. Funkční instance dočasně běží na `http://130.162.49.62:8080/`.

```
sudo apt install npm python3 python3-pip
```

Listing B.1: Stažení NPM a Python + pip

```
pip install "fastapi[all]"
pip install "uvicorn[standard]"
cd web/
npm install
```

Listing B.2: Instalace závislostí

Po stažení závislostí je potřeba nastavit IP adresy a porty pro komunikaci. Nejprve je nutno v souboru **/api/api.py**, v sekci *allow_origins=[]* nastavit adresu webové aplikace. Poté v souboru **/main/src/main.js** proměnnou *axios.defaults.baseURL*.

Následně lze spustit API příkazem:

```
python3 -m uvicorn api.api:app --reload --port=8000
```

B.1 Kompilace uživatelského rozhraní

Uživatelské rozhraní lze nyní spustit pomocí **npm run serve**. Pro jeho nasazení je však vhodné zabalit veškeré závislosti a posléze použít libovolný HTTP server pro jeho distribuci.

```
npm run build
cd dist
npm serve .
```

Listing B.3: Spuštení uživatelského rozhraní