

Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra kybernetiky

DIPLOMOVÁ PRÁCE

PLZEŇ, 2024

Bc. Tomáš LEBEDA

Obsah

1	Úvod	4
2	Problematika reprezentace znalostí	5
2.1	Sémantika	5
2.2	Reprezentace znalostí	5
2.3	Popis obrázků	5
3	Návrh a architektura systému	6
3.1	Obecná architektura systému	6
3.2	Reprezentace znalostí	7
3.2.1	Objekty ve scéně	8
3.2.2	Hierarchie objektů	9
3.2.3	Atributy objektů	10
3.2.4	Vazby mezi objekty	11
3.3	Extrakce sémantické informace	13
3.3.1	Podoba sémantických entit	14
3.3.2	Extrakce sémantiky - objekty	14
3.3.3	Extrakce sémantiky - atributy	15
3.3.4	Extrakce sémantiky - vazby mezi objekty	16
3.4	Hodnotící algoritmus	17
3.4.1	Chybějící objekty	18
3.4.2	Chybějící atributy	20
3.4.3	Chybějící vazby mezi objekty	20
3.4.4	Atributy s chybnou hodnotou	21
3.4.5	Výstup hodnotícího algoritmu	23
4	Implementace a testování	25
4.1	Referenční popis obrázku	25
4.1.1	Validace referenčního popisu	29
4.1.2	Další příkazy pro práci s referenčním popisem	30
4.2	Sémantické parsování pomocí gramatik	32
4.2.1	Problémy s existující implementací a SRGS	32
4.2.2	Semantic Parsing Grammar Format (SPGF)	34

4.2.3	Vstupní a výstupní body SPGF gramatiky	37
4.2.4	Tagy v SPGF	38
4.2.5	Parsovací strategie a opakování v SPGF	40
5	Závěr	46

1 Úvod

2 Problematika reprezentace znalostí

2.1 Sémantika

2.2 Reprezentace znalostí

2.3 Popis obrázků

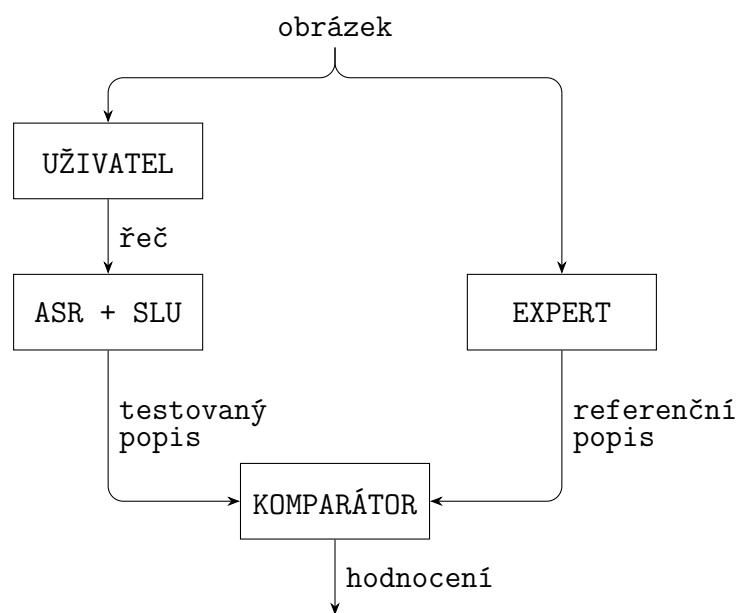
3 Návrh a architektura systému

3.1 Obecná architektura systému

Obecná architektura celého systému vychází z jeho požadované funkčnosti, kterou je porovnání obrázku s jeho popisem v přirozené řeči, a to na sémantické úrovni. Z toho pak plyne, že celý systém se ve své podstatě skládá ze tří základních částí:

1. referenční (vzorový) popis daného obrázku
2. sub-systém pro zpracování přirozené řeči a tvorbu testovaného popisu
3. porovnání referenčního a testovaného popisu

Jednotlivé části spolu vzájemně fungují následujícím způsobem: Uživateli je prezentován obrázek a jeho úkolem je popsat, co na obrázku vidí. Získaný popis v přirozené řeči je převeden na text (ASR). ^{TODO 1} Z tohoto přepisu je extrahována sémantická informace (SLU), ze které je vytvořen testovaný popis. Testovaný popis je porovnaný s referenčním (vzorovým) popisem daného obrázku. Výsledek tohoto porovnání lze pak považovat za finální výstup, ale také je možné jej použít jako vstup pro další zpracování (např. vektor příznaků pro klasifikátor). Schématické znázornění je na Obrázku 1.



Obrázek 1: Schéma obecné architektury systému

¹Jak/kde vysvětlit zkratky?

Pro tuto práci bylo rozhodnuto, že základem bude expertní přístup. Od toho se poté odvíjí konkrétní algoritmy, formáty a postupy navržené a implementované v této práci, které jsou podrobněji popsány v pozdějších kapitolách. Je ale vhodné zmínit, že během návrhu bylo dbáno na to, aby bylo možné pro reálná nasazení některé implementace v případě potřeby zaměnit nebo upravit, aby lépe vyhovovaly specifickým požadavkům pro dané použití. ^{TODO2}

3.2 Reprezentace znalostí

Pro porovnání obrázku s jeho popisem v přirozené řeči bylo nutné zvolit či navrhnout nějakou formu reprezentace znalostí, která by umožnila zachytit sémantiku z obou zdrojů. Jak již bylo výše zmíněno, zvolen byl expertní přístup a to v tomto případě znamená, že referenční popis obrázku je vytvořen lidským expertem, což klade další omezení na formát reprezentace znalostí. ^{TODO3}

Při návrhu bylo tedy potřeba brát v úvahu následující požadavky a najít nějaký formát, který by představoval vhodný kompromis mezi nimi.

- **Čitelnost člověkem:**

Aby byl lidský expert schopen vytvořit, číst a případně upravit referenční popisy, je nutné, aby byl schopen porozumět formě a zápisu uložených dat. Toto omezení tedy upřednostňuje textové formáty a prakticky vyřazuje binární data.

Výjimku by mohl tvořit nějaký binární formát s přidruženým editorem, kde by člověk mohl v grafickém prostředí prohlížet a manipulovat data, ale takový případ je nad rámec této práce.

- **Kompaktnost a struktura dat:**

Dalším důležitým aspektem je struktura a kompaktnost dat. Pomocí počítače je poměrně snadné v krátkém čase zpracovat velké množství jednoduchých datových záznamů, nicméně člověk se bude lépe orientovat v nějakém kompaktnějším popisu, který ačkoli může být složitější ve své struktuře, tak bude pro člověka lépe názorný a uchopitelný.

²zmínit, že referenční popis není třeba pokaždé tvořit znovu, ale lze udělat „offline“ předem?

³zmínit, že expertní přístup umožňuje lepší kontrolu nad obsahem/kvalitou než automat/statistiká?

- **Univerzálnost formátu:**

Podstatnou vlastností pro hledanou reprezentaci znalostí je její schopnost zachytit popis různých obrázků. Navržený formát tedy musí být dostatečně univerzální, aby pomocí něj šlo popsat co nejsírovší spektrum informací, od jednoduchých obrázků zobrazujících například jeden statický objekt, přes složitější obrázky zobrazující více objektů, až po dynamické komplexní scény zobrazující mnoho objektů, činnosti a vazby mezi nimi.

- **Počítačová zpracovatelnost:**

V neposlední řadě je také potřeba dbát na to, aby navržený formát bylo možné co nejsnadněji zpracovat programově, na počítači. Dynamické formáty s volnou strukturou bývají složitější na strojové zpracování, než fixní formáty s přesně definovanou podobou.

S ohledem na tyto body byla navržena reprezentace znalostí založená na sémantických sítích, která definuje 4 základní *aspekty popisu*: objekty, jejich hierarchii, statické atributy a dynamické vazby. Detailnější popis těchto jednotlivých aspektů je v následujících částech, konkrétní technická implementace je pak popsána v sekci 4. TODO⁴

3.2.1 Objekty ve scéně

Cokoli, co lze v obrázku ohraničit rámečkem (angl. bounding-box) TODO⁵ a při separaci od zbytku scény (obrázku) neztratí nebo zásadně nezmění svůj význam, lze považovat za *objekt*.

Jako *objekt* v obrázku lze tedy označit zobrazené fyzické předměty, postavy, zvířata, ale také nehmotné pojmy jako „nebe“, místa, lokace či místnosti (např. „kuchyň“ nebo „louka“) a části jiných objektů (např. „obličej“ jsou součást hlavy nebo celého člověka).

Tato definice objektu byla záměrně navržena velmi obecně, aby byl definovaný formát univerzální a šel použít i pro popis velmi odlišných obrázků s různými účely. Potenciální nevýhodou, která plyne z univerzálnosti formátu, může být v některých situacích problém nejednoznačnosti.

⁴jak a kde se správně používá emph pro zdůraznění pojmu/termínu?

⁵jak s anglikanismy?

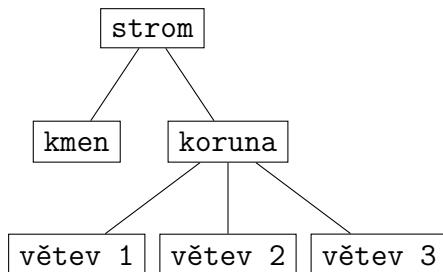
V jednom obrázku lze definovat různé množiny objektů, podle toho, jak moc detailní popis expert vytvoří. Například pokud by byl na obrázku člověk, lze jej popsat jedním objektem jako „člověk“ nebo „osoba“, ale také by šlo definovat ještě mnoho dalších objektů, například pro jednotlivé části těla nebo oblečení.

Kromě různých úrovní detailů lze také na problematiku nejednoznačnosti narazit v situaci, kdy je pro nějaký (dostatečně komplexní) obrázek možné sestavit různé množiny objektů podle toho, pro jaké potřeby je zrovna obrázek a referenční popis používán. Pro aplikaci, kde je podstatné zachycení živých objektů, může být množina objektů v referenčním popise tvořena lidmi či zvířaty. Pro jiné použití pak ale může být podstatné zachytit prostředí a neživé předměty, takže množina objektů by byla tvořena částmi prostředí (např. stromy, kroví, voda, skály), budovami nebo obecnými předměty. TODO6

3.2.2 Hierarchie objektů

Kromě množiny samotných objektů lze v obrázku také definovat jejich hierarchii. To přirozeně plyne z výše zmíněné definice *objektu*, která umožnuje specifikovat část existujícího objektu jako další samostatné objekty.

Příkladem takového popisu může být například situace, kdy je na obrázku strom. Strom je možné rozdělit na korunu a kmen, korunu pak je možné dále dělit na větve. Schématicky lze tento popis znázornit jako stromovou strukturu, viz Obrázek 2.



Obrázek 2: Schématické znázornění hierarchie objektů

Z pohledu daného objektu jsou „vyšší“ (obecnější) objekty označované jako *rodičovské objekty* (nebo jen *rodiče*) a „nižší“ (konkrétnější) objekty pak jako *potomci*.

⁶vyměnit dlouhé popisy za jeden ukázkový příklad?

Možnost definovat hierarchii objektů nabízí mimo jiné také způsob, jak vytvořit skupiny objektů, které k sobě nějakým způsobem patří. Například strom může být součástí lesa, kráva může být součástí stáda nebo postava na hřišti může být součástí fotbalového týmu. Tato příslušnost objektu nějaké skupině je dalším typem sémantické informace, kterou umožňuje navržený formát zachytit bez nutnosti definovat další specializované struktury.

Mohou však nastat situace, kdy je potřeba jeden objekt zařadit do několika různých skupin. Z tohoto důvodu bylo rozhodnuto, že každý objekt může mít libovolné množství rodičů a libovolné množství potomků. ^{TODO7}

3.2.3 Atributy objektů

Vedle pouhého výčtu samotných objektů ve scéně je dále přirozeným požadavkem, aby byla reprezentace znalostí schopna zachytit i jejich vlastnosti. K tomu slouží třetí aspekt popisu - *atribut*.

Atributem je možné popsat jakoukoli informaci o objektu, která není závislá na jiném objektu. Jinými slovy, pokud bychom objekt izolovali od zbytku scény (obrázku), tak všechny vlastnosti, které se tím nezmění nebo nezaniknou, lze popsát pomocí atributů. Typickým příkladem je barva nebo tvar objektu či jeho části.

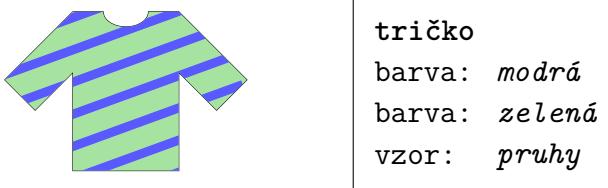
Atribut se skládá ze zvoleného *názvu* a přiřazené *hodnoty*, kdy konkrétní názvy a hodnoty jsou volbou experta, který tvoří referenční popis. Název atributu označuje jakou vlastnost daný atribut popisuje a přiřazená hodnota pak udává, jaké konkrétní hodnoty nabývá. Každý objekt může mít libovolné množství těchto atributů.

Názvy atributů pod jedním objektem nemusí být unikátní, lze specifikovat více stejnojmenných atributů s různými hodnotami. Typickým příkladem může být vícebarevný objekt, kde atribut s názvem „barva“ bude vícekrát, pro různé konkrétní barvy. Příklad jednoho takového popisu je na Obrázku 3, kde název objektu je „tričko“ a definované jsou tři atributy: dvě barvy a vzor. ^{TODO8} ^{TODO9}

⁷ zdůvodňovat? dávat příklad? přirovnávat k OOP dědičnost vs kompozice?

⁸ hezčí tabulka pro popis objektu?

⁹ přidat šipky a popisky do tabulky co je co?



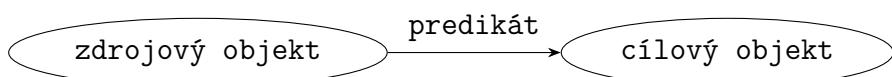
Obrázek 3: Příklad použití atributů pro popis objektu

Pro složitější objekty se opět naskytá problém toho, jaké vlastnosti vybrat a zapsat do referenčního popisu, a které je možné naopak zanedbat. Tato otázka je podobná problematice nejednoznačnosti, zmíněné v části 3.2.1 definující objekty. A i zde platí to, že popis byl záměrně vytvořen tak, aby byl univerzální a volba atributů do referenčního závisí na expertovi a konkrétní aplikaci.

3.2.4 Vazby mezi objekty

Posledním typem informace, kterou by měl být referenční popis obrázku schopen zachytit, jsou vazby a vztahy mezi různými objekty. Může se jednat například o činnosti, které se týkají dvou objektů, nebo popis relativních vlastností, jako je velikost či pozice.

Každý takový záznam je označen jako *triplet* a skládá se ze *zdrojového objektu*, *cílového objektu* a *predikátu*, který popisuje danou vazbu nebo vztah. Tato struktura je inspirována běžným zápisem sémantických sítí a RDF standardem. ^{TODO 10} Obecné schéma jednoho tripletu je na Obrázku 4.



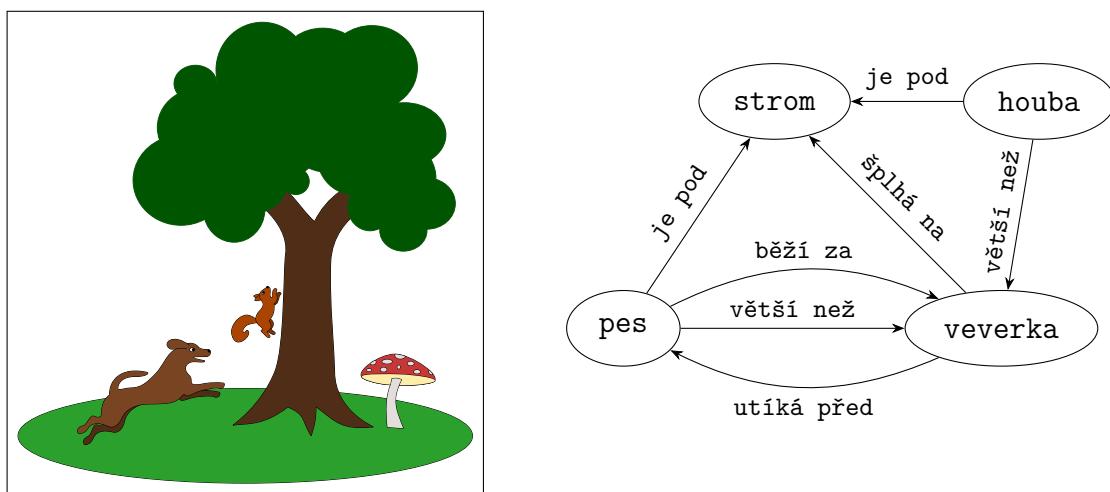
Obrázek 4: Obecné schéma tripletu

Každý objekt může být součástí libovolného počtu tripletů a to jak v pozici zdrojového, tak cílového objektu. Objekt také nemusí být součástí žádných vazeb a dokonce v celém popisu nemusí být žádné vazby definované. Například pokud bychom chtěli pouze testovat paměť uživatele, mohli bychom mu ukázat obrázek, pak jej skrýt a sledovat, jaké objekty si vybaví. V takové úloze nás vazby mezi objekty vůbec nemusejí zajímat a je tedy zbytečné, aby byly součástí referenčního popisu.

¹⁰doplnit reference!

Při vytváření referenčního popisu obrázku se opět naskytá otázka, jaké vazby a vztahy mezi objekty do popisu zavést a které lze ignorovat. Například relativní velikost nebo pozice objektů může být pro některé úlohy klíčová, ale pro jiné zcela nepodstatná. Tato problematika byla opět ponechána na expertovi, který vytváří referenční popis pro danou konkrétní aplikaci, aby rozhodl, které informace je potřeba v referenčním popise zachytit.

Konkrétní příklad několika tripletů je znázorněn na Obrázku 5.



Obrázek 5: Příklad použití tripletů pro popis vztahů mezi objekty

TODO 11

¹¹ příklad s kompletním popisem obrázku (všechno dohromady)?

3.3 Extrakce sémantické informace

Dalším klíčovým bodem bylo najít způsob, jak porovnat uživatelský popis obrázku s jeho referenčním popisem. To je úlohou subsystému pro získání sémantické informace z textu. Cílem je z přirozené řeči extrahovat informace v takové podobě, aby je bylo možné porovnat se strukturovaným referenčním popisem.

Extrakce sémantiky z přirozené řeči se běžně dělá z textového přepisu dané promluvy. ^{TODO12} I v této práci tedy extrakce sémantických informací probíhá z textu. To znamená, že pokud uživatel popíše obrázek mluvenou řečí, tak je potřeba promluvu převést do textu pomocí nějakého systému pro rozpoznání řeči (ASR). ^{TODO13} Problematika rozpoznání řeči a převodu audia do textu je nad rámec této práce a předpokládá se, že bude v praxi řešena nějakou již existující implementací. ^{TODO14} Ve zbytku práce bude tedy pro zjednodušení rovnou uvažovaným vstupem text. ^{TODO15}

Otázka extrakce sémantické informace z popisu přirozenou řečí se tedy zužuje na extrakci sémantiky z textu. Jako způsob řešení byl zvolen přístup založený na sémantickém parsování pomocí bezkontextových gramatik.

Tento přístup funguje tak, že na základě referenčního popisu obrázku bude expertem sestavena gramatika, podle které budou v textu detekované jednotlivé části sémantické informace. Gramatika je v tomto kontextu sada pravidel, která definují, jaké promluvy jsou v textu očekávané a jak mohou tyto promluvy vypadat. Program pak tato pravidla načte a prochází podle nich vstupní text a hledá, zda nějaké části textu „pasují“ na dané pravidlo. Samotný nalezený kus textu, který „pasuje“ na nějaké pravidlo, ale ještě nemá žádný užitečný význam. Proto pravidla obsahují zároveň i informaci o tom, jaká je struktura nalezené části textu a jaký význam mají jednotlivé části této struktury. V českém jazyce by se jednalo o obdobu větného rozboru, kdy se například určuje, která část věty je podmět a která přísudek. Strukturovaná a označovaná část textu je pak označována jako *sémantická entita*. ^{TODO16} Konkrétní syntaxe, použití a funkčnost těchto gramatik bude popsán později spolu s implementací v části 4.2.

¹²ozdrojovat?

¹³existují metody co to dělají rovnou z audia?

¹⁴zmínit SpeechCloud jako existující implementaci?

¹⁵doplnit, že jsem při vývoji používal ruční přepisy + odkud? Nebo až později u implementace?

¹⁶nemá „sémantická entita“ jiný význam a můžu takhle přetížit definici?

Množina všech těchto sémantických entit, které byly ze vstupního textu extrahované, pak tvoří testovaný popis.

3.3.1 Podoba sémantických entit

První otázkou, kterou bylo potřeba vyřešit pro získání sémantické informace z přirozeného popisu, byla její podoba. Jinými slovy, jak by měla extrahovaná sémantika vypadat, aby ji bylo možné porovnat s referenčním popisem obrázku.

Vzhledem k tomu, že výše definovaný referenční popis (viz sekce 3.2) se skládá z objektů, jejich hierarchie, atributů a vazeb, tak se nabízí přímo použít tyto typy informací. Bylo tedy rozhodnuto, že z přepisu přirozené řeči budou extrahované objekty, jejich atributy a vazby mezi nimi.

3.3.2 Extrakce sémantiky - objekty

Základním typem informace, kterou musí být systém schopen z přepisu řeči získat, jsou samotné objekty, které expert definoval v referenčním popisu obrázku.

Například ze vstupní promluvy

„Na obrázku vidím psa s veverkou, strom a nějakou houbu.“

by měl systém vrátit množinu detekovaných objektů \mathcal{O} :

$$\mathcal{O} = \{\text{pes, veverka, strom, houba}\}$$

Ve své nejjednodušší podobě by se mohlo jednat pouze o detekci nějakých klíčových slov, které odpovídají názvům objektů. Skutečná realizace je poněkud složitější, bere v potaz různá synonyma, tvary slov a alternativní vyjádření. Detailněji bude popsána později spolu s konkrétní implementací v kapitole ^{TODO17}.

Detekce samotných objektů by mohla být pro některé aplikace dokonce postačující sama o sobě. Mohlo by se jednat o úlohy, kde je hlavním předmětem pouze zjistit, kolik objektů na obrázku člověk popíše, případně které to jsou. Typickými příklady by mohli být nějaké testy paměti, pozornosti nebo jednoduché klasifikace.

¹⁷doplnit referenci na budoucí kapitolu

3.3.3 Extrakce sémantiky - atributy

Druhým typem sémantické informace, kterou je potřeba, aby byl systém schopen najít a extrahovat z textu, jsou atributy popisující vlastnosti objektů.

Ve srovnání s detekcí samotných objektů se jedná o značně složitější problém, protože pro extrakci atributu je potřeba mít informace o objektu, na který se atribut váže, o názvu daného atributu a také o jeho přiřazené hodnotě.

Například ve větě

„*Na obrázku vidím kluka v modrém tričku.*“

by měl systém detekovat, že objekt „tričko“ má atribut „barva“ s hodnotou „modrá“.

Kromě toho, že se tato informace skládá z více nezávislých částí, tak je možné si všimnout, že se ve zdrojové větě nikde nevyskytuje slovo „*barva*“. Toto je tedy informace, kterou musí být systém schopen nějakým způsobem indukovat z okolních dat a referenčního popisu.

Dále by jiný uživatel mohl popsát stejný obrázek třeba větou

„*Vidím nějakého kluka v tričku, které je modré.*“

Tato promluva obsahuje stejný objekt, atribut i hodnotu, ale vyjádřenou ve zcela jiné podobě. Jak je tedy zřejmé, zde již není možné použít pouze nějakou formu detekce klíčových slov, ale bude potřeba komplikovanějšího přístupu.

Právě dříve zmíněná pravidla, která udávají různé formy hledané informace, umožňují zachytit i takovéto složitější struktury v různých formách. Konkrétní realizace bude opět popsána později, viz [TODO 18](#).

¹⁸doplnit referenci na budoucí kapitolu

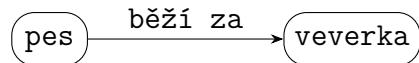
3.3.4 Extrakce sémantiky - vazby mezi objekty

Posledním typem informace, kterou je potřeba získat z textu, jsou vazby mezi objekty.

Například pro větu

„Na obrázku hnědého vidím psa, co běží za oranžovou veverkou.“

je třeba, aby systém dokázal detektovat vazbu mezi objektem psa a veverky:



Jak je zřejmé, pro sestavení vazby je - stejně jako pro atributy - potřeba tří částí informace: zdrojový objekt, cílový objekt a název vazby. Na rozdíl od detekce atributů ale představuje detekce vazeb unikátní problém, protože zdrojový a cílový objekt jsou stejného typu - oba jsou to objekty. To znamená, že v pří konstrukci tripletů musí být použit nějaký mechanismus, který rozpozná, který z objektů má být cílový a který zdrojový. Tento problém je řešen také v rámci definice pravidel v bezkontextové gramatice a bude detailně adresován v sekci [TODO19](#).

Je vhodné zmínit, že ačkoli byly v kapitole 3.2 definované čtyři aspekty popisu, nyní jsou řešené pouze tři různé typy sémantických entit. To je proto, že hierarchie mezi objekty byla při návrhu považována za speciální případ vazby mezi objekty a nebyla pro ni vytvořena samostatná kategorie.

¹⁹doplnit referenci na budoucí kapitolu

3.4 Hodnotící algoritmus

Za předpokladu, že je k dispozici referenční popis vytvořený expertem a testovaný popis získaný automaticky z přirozené řeči, je potřeba tyto dva popisy nějakým způsobem porovnat a určit, do jaké míry odpovídá testovaný popis referenčnímu.

Právě to je předmětem této kapitoly: navrhnut algoritmus, který by zajišťoval porovnání testovaného popisu s referenčním a jehož výstupem by bylo nějaké hodnocení.

Základ hodnotícího algoritmu vychází ze struktury referenčního a testovaného popisu. Oba typy se skládají z množiny objektů \mathcal{O} , množiny atributů \mathcal{A} a množiny vazeb \mathcal{V} . Dolním indexem je značeno, zda se jedná o množinu z referenčního popisu (R), nebo testovaného popisu (T). Oba popisy obrázku lze tedy vyjádřit jako trojice: TODO²⁰

$$\text{referenční popis obrázku} = (\mathcal{O}_R, \mathcal{A}_R, \mathcal{V}_R)$$

$$\text{testovaný popis obrázku} = (\mathcal{O}_T, \mathcal{A}_T, \mathcal{V}_T)$$

Porovnání a hodnocení podobnosti popisů by tak mohlo být převedeno na otázku porovnání podobnosti množin. Otázky teorie množin jsou ale nad rámec zadání a pro tuto práci byl definován hodnotící algoritmus, který je inspirován ztrátovými funkczemi.

Na obecné úrovni by se dalo říci, že hodnotící algoritmus počítá celkové ztráty pro chybějící objekty, chybějící atributy, atributy s chybnou hodnotou, chybějící vazby mezi objekty.

Vstupem hodnotícího algoritmu jsou tedy oba popisy (referenční a testovaný) a k tomu ještě ztrátová tabulka, která určuje, jaký typ chyby způsobí jak velkou ztrátu. Tuto ztrátovou tabulkou také sestavuje expert, společně s referenčním popisem.

Výstupem hodnotícího algoritmu je pak množina označených číselných hodnot, které reprezentují celkové ztráty pro různé druhy chyb.

²⁰definovat symbol pro popis obrázku? použít pro definice jiný symbol než $=$?

3.4.1 Chybějící objekty

Pokud byl v referenčním popise expertem označený nějaký objekt, který chybí v testovaném popise, předpokládá se, že expert považoval daný objekt za důležitý a uživatel jej nezmínil.

Uživatel si třeba nemusel objektu všimnout, nebo jej nepovažoval za dostatečně důležitý. V obou případech se však jedná o nějaký rozpor se vzorovým popisem, který je třeba penalizovat.

Například pro situaci:

$$\begin{aligned}\mathcal{O}_R &= \{\text{pes, veverka, houba, strom}\} \\ \mathcal{O}_T &= \{\text{pes, veverka}\}\end{aligned}$$

je zřejmé, že v testovaném popisu chybí dva objekty: houba a strom.

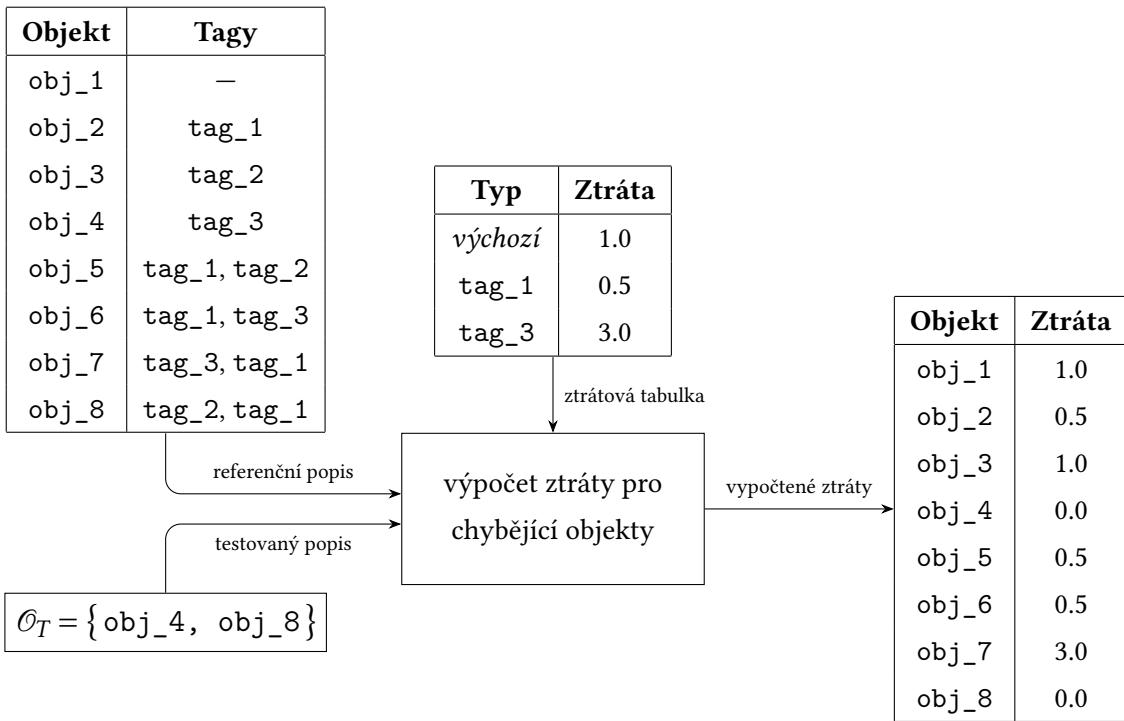
Lze očekávat, že různé objekty budou ve scéně různě důležité a bylo by vhodné, aby byl tento fakt zohledněn při výpočtu ztráty. Proto lze přiřadit objektům *tagy*. Jako *tag* je v tomto kontextu chápána nějaká značka, která říká, že objekt patří do dané skupiny. Například objekt „tričko“ může mít přiřazen tag „*oblečení*“ a objekt „pes“ může mít přiřazen tag „*zvíře*“.

Pro větší variabilitu systému bylo dále rozhodnuto, že každý objekt může mít přiřazený libovolný počet tagů. Název tagů a jejich přiřazení objektům je volbou experta a je součástí tvorby referenčního popisu. Ve ztrátové tabulce pak může expert definovat ztrátové hodnoty pro jednotlivé tagy a tím tak přepsat hodnotu ztráty na daném objektu.

Pokud má objekt přiřazeno více tagů, na jejich pořadí záleží. Při určování ztrátové hodnoty pro daný objekt algoritmus postupně prochází jeden tag po druhém a kontroluje, zda je tento tag uvedený ve ztrátové tabulce. Pokud ano, je ztráta na daném jednom chybějícím objektu rovna této hodnotě a další tagy na tomto objektu již nejsou kontrolované. Pokud žádný z tagů není nalezen ve ztrátové tabulce, je použita výchozí hodnota ztráty pro jakýkoli chybějící objekt.

Pro názornost je schéma jednoho ukázkového případu na Obrázku 6. TODO21

²¹vyměnit obecné názvy za konkrétní?



Obrázek 6: Jednoduchá ukázka ztráty na chybějících objektech

Na příkladu zobrazeném na Obrázku 6 je možné vidět, že objekt obj_4, který byl nalezený v testovaném popisu, nezpůsobil žádnou ztrátu. Ostatní objekty, které v testovaném popise chybí, způsobily ztrátu odpovídající hodnotě ze ztrátové tabulky. Dále je možné si všimnout, že objekty obj_6 a obj_7 mají stejné tagy, ale v jiném pořadí, proto jsou jimi způsobené ztráty rozdílné.

Pro větší kompaktnost výstupu nejsou vypočtené ztráty prezentované pro každý objekt samostatně, ale jsou sečtené do finálního výsledku. Tento proces sčítání ztrátových hodnot je přes všechny objekty, ale také jsou počítané hodnoty přes jednotlivé tagy. Pokud bychom vzali tabulku vypočtených ztrát pro jednotlivé objekty z příkladu na Obrázku 6, šel by tento finální krok znázornit Obrázkem 7.

Zde je vhodné zmínit, že popsaný algoritmus výpočtu ztráty je pouze jednou z možností. Pro konkrétní nasazení by mohla být vhodnější jiná forma počítání ztrátových hodnot, například vypočítat průměr, sumu nebo maximum či minimum. Tento způsob počítání ztráty byl zvolen proto, že poskytuje expertovi další stupeň volnosti, jak vyjádřit důležitost objektů.

Objekt	Tagy	Ztráta
obj_1	—	1.0
obj_2	tag_1	0.5
obj_3	tag_2	1.0
obj_4	tag_3	0.0
obj_5	tag_1, tag_2	0.5
obj_6	tag_1, tag_3	0.5
obj_7	tag_3, tag_1	3.0
obj_8	tag_2, tag_1	0.0

→

Chybějící objekty	Ztráta
všechny	6.5
objekty s tag_1	4.5
objekty s tag_2	1.5
objekty s tag_3	3.5

Obrázek 7: Finálního krok při počítání ztrát na chybějících objektech

3.4.2 Chybějící atributy

Pokud expert popsal na daném objektu nějakou vlastnost a odpovídající atribut chybí v testovaném popise, je zde stejný předpoklad jako u chybějících objektů a také se jedná o chybu, která způsobí navýšení ztrátové hodnoty.

Způsob hodnocení je obdobný jako pro chybějící objekty, pouze s tím rozdílem, že atributům nejsou přiřazovány žádné tagy. Tagy u objektů sloužily k detailnějšímu určení důležitosti objektu a později ke scítání ztrátových hodnot, v případě atributů tuto funkci zastává samotný název atributu.

Například pokud by byl libovolný chybějící atributu penalizován ztrátou 1.0, pak by bylo možné specifikovat, že vynechání atributu „*barva*“ je méně závažné než všechny ostatní a penalizovat pouze ztrátou 0.5. Naopak vynechání atributu „*výraz v obličeji*“ (např. úsměv nebo zamračení) by mohl být považován za důležitý a expert by mohl ve ztrátové tabulce definovat, že jeho vynechání bude penalizováno ztrátou 2.0. TODO22

3.4.3 Chybějící vazby mezi objekty

Posledním typem entity, která může chybět při porovnávání referenčního a testovaného popisu, jsou vazby mezi objekty popsané *triplety*.

²²dát konkrétní příklad včetně tabulek, výpočtů ...?

Zde je hodnotící algoritmus velmi podobný jako při počítání ztráty pro chybějící objekty. Rozdíl zde spočívá v tom, že místo tagů jsou zde použité samotné názvy vazeb, podobně jako u chybějících atributů jsou použité názvy atributů.

Expert ve ztrátové tabulce opět specifikuje, jaká je výchozí hodnota ztráty, pokud v testovaném popise bude chybět libovolná vazba mezi objekty. Následně může také specifikovat, že některé konkrétní typy vazeb (podle jejich názvu) mají větší či menší důležitost a tak jimi způsobená ztráta může být větší nebo menší. Algoritmus pak při výpočtu použije přednostně takovou hodnotu, která je více specifická. TODO23

3.4.4 Atributy s chybnou hodnotou

Pokud v byl v testovém popise nalezen atribut, který odpovídá objektem a názvem nějakému atributu z referenčního popisu, ale neodpovídá svou hodnotou žádnému referenčnímu atributu, pak se předpokládá, že uživatel udělal chybu při popisu a řekl špatnou hodnotu.

Příkladem takové neshody by mohlo být:

$$\begin{aligned}\mathcal{A}_R &= \{ \text{tričko: barva} = \text{modrá} \} \\ \mathcal{A}_T &= \{ \text{tričko: barva} = \text{zelená} \}\end{aligned}$$

pak je možné předpokládat, že uživatel udělal při popisu chybu a řekl špatnou barvu.

Způsob počítání ztrátové hodnoty z těchto chybných hodnot je v principu stejný jako u chybějících objektů a atributů. Expert ve ztrátové tabulce definuje výchozí hodnotu a k tomu případně specifické případy, kdy je tato ztrátová hodnota jiná.

Rozdíl oproti předchozím výpočtům ovšem spočívá v tom, že různé hodnoty si mohou být různě blízko. Například pokud uživatel zamění hnědou a oranžovou barvu veverky, tak se pravděpodobně ve většině případů bude jednat o menší chybu, než kdyby oranžovou zaměnil třeba za zelenou.

Z toho důvodu je ztrátová tabulka navržena tak, aby expert mohl definovat výchozí ztrátu pro všechny chybné atributy, výchozí ztrátu pro konkrétní atribut bez ohledu na hodnoty, ztrátu pro konkrétní atribut a zároveň množinu konkrétních hodnot, jejichž záměna tuto ztrátu způsobí

²³dát konkrétní příklad včetně tabulek, výpočtů ...?

Pro lepší názornost následuje příklad, který předpokládá, že expert definoval ztrátovou tabulkou shodnou s Tabulkou 2. Tu je možné číst následujícím způsobem:

1. Libovolný chybný atribut způsobí ztrátu 1.0, pokud není ve zbytku tabulky definována více specifická alternativa.
2. Pokud je chybná hodnota v atributu „barva“, pak je způsobená ztráta rovna 0.5, pokud se nejedná o záměnu některých konkrétních hodnot:
 - Pokud je zaměněna barva „žlutá“ a „oranžová“, pak je ztráta 0.3.
 - Pokud je zaměněna barva „červená“, „modrá“ nebo „zelená“ (libovolná dvojice), pak je ztráta rovna 0.8.
3. Pokud je chybná hodnota v atributu „tvar“, pak je ztráta 1.5.

Ukázka některých konkrétních atributů s chybnými hodnotami a k nim vypočtených ztrát je zobrazena v Tabulce 1.

Atribut	Ztráta
výchozí	1.0
barva	•
tvar	•

Hodnoty	Ztráta
výchozí	0.5
žlutá, oranžová	0.3
červená, modrá, zelená	0.8

Hodnoty	Ztráta
výchozí	1.5

Tabulka 1: Část ukázkové ztrátové tabulky

Referenční atribut	Testovaný atribut	Ztráta
pes: barva = hnědá	pes: barva = hnědá	0.0
oheň: barva = žlutá	oheň: barva = oranžová	0.3
jablko: barva = červená	jablko: barva = modrá	0.8
tričko: barva = modrá	tričko: barva = zelená	0.8
čepice: barva = modrá	čepice: barva = žlutá	0.5
penál: tvar = válec	penál: tvar = kvádr	1.5
pes: činnost = leží	pes: činnost = leží	1.0

Tabulka 2: Výpočtu ukázkových ztrát chybných hodnot atributů podle Tabulky 1

Přirozeně se naskytá otázka toho, jak počítat ztrátu při konfliktu objektů se stejnými atributy a nebo objekty s více atributy. Pro řešení těchto konfliktů bylo rozhodnuto, že hodnotící algoritmus bude optimistický a bude uvažovat nejmenší možnou chybu, pokud je podle ztrátové tabulky k dispozici více možných interpretací.

3.4.5 Výstup hodnotícího algoritmu

Finálním výstupem hodnotícího algoritmu je množina označených číselných hodnot, získaných akumulací ztrát pro různé druhy chyb. Přirozenou datovou strukturou, která by odpovídala tomuto formátu, by bylo asociativní pole (někdy také označované jako hash-tabulka, nebo slovník).

Výstup hodnotícího algoritmu, jak byl popsán v předchozích částech, obsahuje vždy celkovou ztrátu pro daný typ chyby a poté ještě akumulované ztrátové hodnoty stejného typu, ale rozdělené podle jednotlivých kategorií. todo24

Konkrétně bude výstupní struktura obsahovat:

- celkovou ztrátu způsobenou všemi chybějícími objekty
- celkovou ztrátu způsobenou všemi chybějícími atributy
- celkovou ztrátu způsobenou všemi chybějícími vazbami mezi objekty
- celkovou ztrátu způsobenou všemi atributy s chybnými hodnotami

²⁴příklad?

- ztráty způsobené chybějícími objekty s konkrétním tagem
- ztráty způsobené chybějícími atributy s konkrétním názvem/typem atributu
- ztráty způsobené chybějícími vazbami s konkrétním názvem/typem vazby
- ztráty způsobené atributy s chybnými hodnotami, rozdělené podle názvu/typu atributu

Je vhodné zmínit, že při návrhu hodnotícího algoritmu bylo předpokládáno, že pro různá konkrétní nasazení mohou být rozdílné požadavky na formu výstupu. V rámci implementace popsané později existuje proto „skrytý“ mezi-krok, ve kterém jsou k dispozici všechny jednotlivé ztráty pro konkrétní objekty, atributy a vazby, ze kterých je poté počítána právě popsaná výsledná struktura. Tento mezi-výsledek by bylo možné výhodně využít pro implementaci alternativních způsobů hodnocení, nebo případně rovnou použít jako výstup, pokud by daná aplikace benefitovala z podrobnější analýzy výsledků.

4 Implementace a testování

4.1 Referenční popis obrázku

Jako první byl implementován systém pro práci s referenčním popisem obrázku. Aby bylo možné vytvořenou implementaci rovnou testovat, bylo nutné zvolit nějaký konkrétní obrázek, na kterém by bylo možné jednotlivé části zkoušet. Pro tento účel byla zvolena kresba ^{TODO25}, která je zobrazena na Obrázku 8. ^{TODO26}



Obrázek 8: Kresba použitá při implementaci pro testování

Jak bylo definováno v kapitole 3.2, referenční popis obrázku se skládá z množiny objektů, jejich atributů a vazeb mezi nimi. Nyní bylo potřeba stanovit nějaký datový formát, který by umožnil tuto strukturu dobře zachytit.

²⁵doplnit odkud obrázek je + správě ozdrojovat

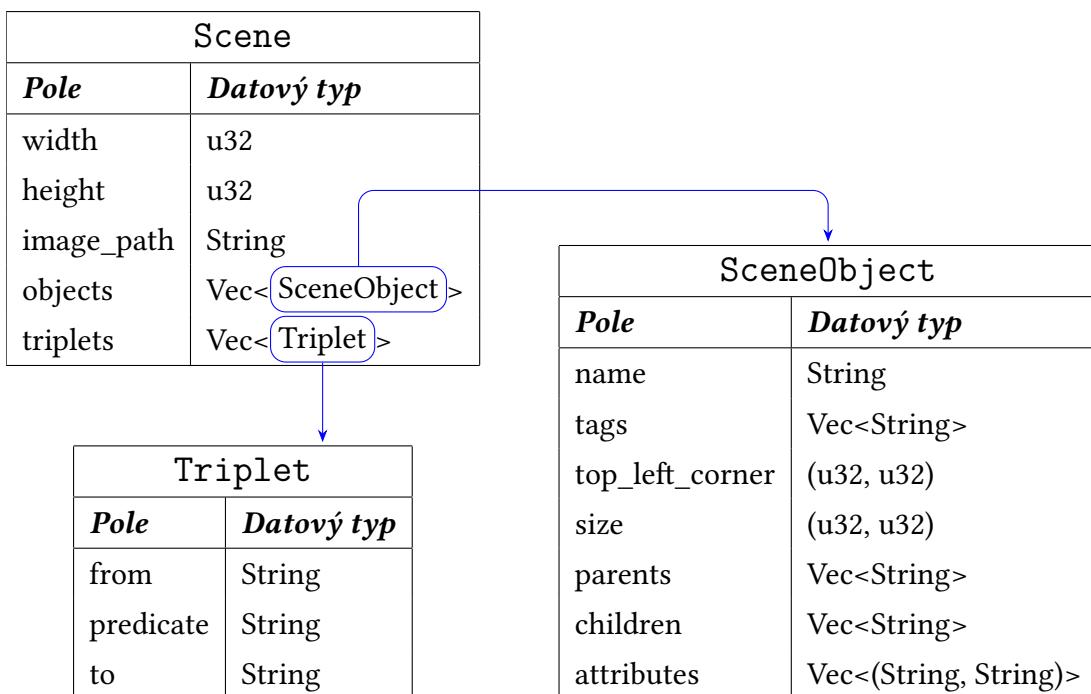
²⁶doplnit nebo odebrat bílý okraj obrázku

Základním prvkem referenčního popisu je struktura „Scene“, která obsahuje informace o obrázku, objekty s atributy a vazby mezi nimi.

Objekty byly definované strukturou „SceneObject“, která popisuje objekt ve scéně a jeho atributy. Každý SceneObject obsahuje informaci o svém názvu, umístění, velikosti a hierarchii vůči ostatním objektům. Dále si každý SceneObject udržuje list atributů a tagů, které mu byly expertem přiděleny.

Další strukturou, která byla definována jako součást referenčního popisu, je „Triplet“. Tato struktura slouží k popisu vazeb mezi objekty, obsahuje názvy dvou objektů a k nim název vazby.

Schéma implementovaných struktur a jejich provázání je na Obrázku 9. Zdrojový kód je pak možné si prohlédnout [TODO27](#) [TODO28](#) [TODO29](#)



Obrázek 9: Schéma implementace referenčního popisu

Ve struktuře Scene je význam jednotlivých polí poměrně jednoduchý. Pole width a height vyjadřují výšku a šířku obrázku v pixelech, image_path udává cestu k souboru,

²⁷doplnit odkaz na zdrojáky

²⁸vyměnit tabulky za pseudo-kód?

²⁹vysvětlit u32 a Vec<>?

který obsahuje samotný obrázek. Pole `objects` a `triplets` pak obsahují list objektů a tripletů.

Struktura `Triplet` je velmi jednoduchá. Její pole `from` a `to` jsou názvy zdrojového, respektive cílového, objektu. Pole `predicate` je název samotné vazby mezi těmito objekty.

Poslední struktura `SceneObject`, která popisuje samotné objekty, je nejsložitější. Pole `name` udává název daného objektu, který slouží zároveň jako identifikátor a musí být tudíž unikátní. Může nastat situace, že v obrázku bude více objektů, které by přirozeně byly označené stejně, například dva stromy. Pro tento případ byla implementována možnost objekty číslovat a tím tak jednoznačně rozlišit i objekty stejného typu. Číslování objektů je součástí jejich názvu a má formát „#*n*“ kde *n* je přirozené číslo označující index objektu. Například „strom #1“ a „strom #2“.

Dvojice polí `top_left_corner` a `size` udávají velikost a pozici objektu zdrojovém obrázku. Jedná se o souřadnice levého horního rohu ohraničujícího rámce (bounding-boxu) a velikost tohoto rámce v pixelech. Obě pole mají jako uvedený datový typ (`u32`, `u32`), který reprezentuje souřadnice (i, j) kde $i, j \in \mathbb{N}^0$. V obrázku je dle konvence jako počátek souřadnic uvažován levý horní roh. TODO30

Pole `tags` reprezentuje list tagů, které expert obrázku přiřadil. Jejich účel byl popsán v kapitole 3.4. Další dvojice polí `parents` a `children` pak reprezentuje list objektů, které jsou rodiče, respektive potomci, daného objektu.

Po formální definice formátů a datových typů bylo potřeba určit, jakým způsobem bude expert referenční popis tvořit. Pro tuto práci bylo rozhodnuto, že tvorba referenčního popisu bude spočívat v napsání strukturovaného textového souboru. Zvolen byl textový formát JSON, který představuje dobrý kompromis mezi čitelností člověkem a strojovou zpracovatelností.

Ukázka části referenčního popisu je na Listing 1. TODO31 TODO32

³⁰zdrojovat kde jsem vzal „konvenci“?

³¹jak se česky skloňuje a překládá Listing?

³²ukázat stejný popis graficky jako schéma + obrázek?

```

1   {
2     "width": 2002,
3     "height": 1123,
4     "image_path": "./data/imgs/summer.png",
5     "objects": [
6       {
7         "name": "tree #1",
8         "tags": ["environment"],
9         "top_left_corner": [1194, 146],
10        "size": [264, 220],
11        "parents": [],
12        "children": ["branch"],
13        "attributes": [
14          ["color", "green"],
15          ["color", "brown"]
16        ]
17      }, {
18        "name": "bird",
19        "tags": ["animal"],
20        "top_left_corner": [1095, 193],
21        "size": [93, 47],
22        "parents": [],
23        "children": [],
24        "attributes": [[["color", "white"]]]
25      }, {
26        "name": "squirrel",
27        "tags": ["animal"],
28        "top_left_corner": [1655, 286],
29        "size": [116, 76],
30        "parents": [],
31        "children": [],
32        "attributes": [
33          ["color", "orange"],
34          ["color", "brown"]
35        ]
36      }, {
37        "name": "bird",
38        "tags": ["animal"],
39        "top_left_corner": [1095, 193],
40        "size": [93, 47],
41        "parents": [],
42        "children": [],
43        "attributes": [[["color", "white"]]]
44      }, {
45        "name": "branch",
46        "tags": ["environment", "background"],
47        "top_left_corner": [1130, 233],
48        "size": [138, 41],
49        "parents": ["tree #1"],
50        "children": [],
51        "attributes": []
52      }
53    ],
54    "triplets": [
55      {"from": "bird", "predicate": "sitting on", "to": "branch"}, ,
56      {"from": "bird", "predicate": "sitting on", "to": "tree #1"}, ,
57      {"from": "squirrel", "predicate": "climbing", "to": "tree #1"}, ,
58      {"from": "squirrel", "predicate": "under", "to": "bird"} ,
59    ]
60  }

```

Listing 1: Ukázka jednoduchého referenčního popisu

4.1.1 Validace referenčního popisu

Jak bylo právě ukázáno, tvorba referenčního popisu je ve své podstatě psaní JSON souboru podle nějaké definované struktury a pravidel. Program při zpracovávání referenčního popisu předpokládá, že data budou validní a odmítne referenční popis, který by obsahoval chyby. Pro komplexnější scénu však může být náročné pro lidského experta neudělat v popise žádnou chybu a udržet celý referenční popis validní. TODO33 TODO34

Z toho důvodu byl implementován v rámci celého programu příkaz `check`, který zajišťuje kontrolu všech dříve definovaných pravidel, které musí referenční popis splňovat:

1. Kontrola názvů objektů

Protože je název objektu použit jako identifikátor, tak musí být unikátní a žádné dva objekty ve scéně nesmí mít shodný název. V případě, že by referenční popis obsahoval duplicitní názvy objektů, program na tyto konflikty upozorní.

2. Kontrola přetékání objektů

Jelikož má program k dispozici informace o velikosti obrázku (v pixelech) a zároveň pro jednotlivé objekty také jejich pozici a velikost, tak lze předem ověřit, zda uvedené ohraničující rámce nezasahují mimo obrázek. Pokud se tak stane, program vypíše varovné hlášení a upozorní na jednotlivé problematické hodnoty.

3. Kontrola referencí na objekty

Objekty mohou v rámci hierarchie odkazovat na jiné objekty. Aby byly tyto reference validní, tak objekt nesmí odkazovat sám na sebe, referovaný objekt musí existovat a hierarchické vazby musí být oboustranné. Tím je myšleno to, že pokud objekt **A** specifikuje objekt **B** jako svého rodiče, pak musí i objekt **B** specifikovat **A** jako svého potomka a naopak.

V případě, že by byly detekované nějaké invalidní odkazy, bude opět vypsáno varovné hlášení o tom, co je v referenčním popise špatně.

4. Kontrola referencí v tripletech

Při specifikaci vazeb mezi objekty jsou také odkazované dva objekty. Aby byl triplet validní, musí být oba referované objekty definované ve scéně. V opačném

³³někam na začátek implementace zmínit, že jsem to dělal jako CLI nástroj

³⁴dávat příklady použití těch příkazů a jejich výpisy?

případě bude vypsáno chybové hlášení s informací o chybějících objekty v tripletech.

4.1.2 Další příkazy pro práci s referenčním popisem

Kromě příkazu na kontrolu referenčního popisu byly implementované ještě následující pomocné příkazy. Tyto příkazy slouží obecně k prohlížení a porovnávání referenčních popisů a byly implementované zejména pro situace, kdy by bylo potřeba analyzovat komplexnější scény, porovnávat je mezi sebou a získat obecný přehled o scéně, aniž by potřeba zkoumat samotné JSON soubory.

1. Příkaz stats:

Tento příkaz vypíše základní vybrané statistiky daného referenčního popisu. Hlavním smyslem tohoto příkazu je poskytnout člověku představu o referenčním popisu k němu příslušejícímu obrázku.

Příkaz vypíše cestu k souboru s referenčním popisem, dále cestu k souboru, který obsahuje popisovaný obrázek a jeho velikost v pixelech. Dále vypíše počet objektů a počet vazeb (tripletů), které byly v tomto referenčním popise definované.

2. Příkaz list:

Příkaz list slouží k vypsání seznamu všech entit daného typu, podobně jako při použití databázových dotazů. Tento příkaz má několik pod-příkazů, které určují, jaký typ informace má být vypsán v podobě seznamu. Může se jednat o objekty, tagy, triplety, atributy, samotné názvy nebo hodnoty atributů, názvy vazeb mezi objekty a další. Seznam všech implementovaných pod-příkazů je možné si prohlédnout ve zdrojovém kódu ^{TODO}³⁵

Cílem tohoto příkazu je poskytnout rychlý a pohodlný způsob, jak získat seznam všech prvků daného typu ve scéně, aniž by bylo potřeba přímo pracovat se složitější strukturou JSON souboru. Výstup je formátován tak, aby jej bylo možné snadno dále strojově zpracovat a integrovat do komplexnějších sekvencí příkazů.

3. Příkaz crumble:

Referenční popis obrazu byl v této práci definován jako množina objektů, jejich atributů a vazeb mezi objekty. Při pohledu na datový typ atributu a tripletu je ale

³⁵ přidat odkaz na zdrojáky

možné si všimnout, že se v podstatě neliší - obojí se skládá ze tří částí, jejichž obsah může být zaměnitelný a rozdíl ve významu je dán pouze zavedenou interpretací. Bylo by tedy možné zobecnit všechny tyto entity a popsat je pouze jedním atomickým datovým typem, který obsahuje dva prvky a vazbu mezi nimi. V této podobě by bylo možné celý referenční popis zobrazit jako tradiční sémantickou síť, kde není rozlišeno, co je vazba mezi objekty a co je pouze popis vlastnosti.

Referenční popis nebyl navržen tímto obecnějším způsobem z toho důvodu, že pro komplexnější scény by počet těchto atomických trojic dosahoval tak vysokých hodnot, že by pro lidského experta bylo velmi obtížné se v datech orientovat. Pro strojové zpracování a případné vizualizace by ale mohlo být prospěšné, aby byly všechny typy informací prezentované v jednoduchém formátu. Právě k tomu slouží příkaz `crumble`, který, jak již název napovídá, daný referenční popis „rozdrobí“ a převede všechny informace do množiny prostých trojic. ^{TODO36}

4. Příkaz render:

Tento příkaz slouží k vytvoření vizuální reprezentace referenčního popisu.

Pro správnou funkci tohoto příkazu je nutné, aby byl na počítači dostupný program Graphviz, ^{TODO37} jehož DOT-engine je použitý pro vytvoření grafické prezentace. Výhodou grafické prezentace je přehlednost pro člověka, nicméně pro velmi komplexní scény s mnoha vazbami ani grafická reprezentace nemusí být optimální. Je vhodné zmínit, že jedním z problémů takto implementované vizualizace je to, že pro scény s mnoha objekty tvoří Graphviz velmi dlouhé a úzké obrázky, protože sázecí mechanismus skládá jednotlivé uzly pod sebe.

Grafická prezentace a metody vizualizace jsou nad rámec zadání této práce, proto bylo rozhodnuto, že pro ilustraci tato implementace dostačuje a případné komplexnější způsoby grafické prezentace jsou ponechány jako možná budoucí rozšíření. ^{TODO38}

³⁶přidat příklad?

³⁷reference na graphviz

³⁸příklad + zlepšit formulace?

4.2 Sémantické parsování pomocí gramatik

Extrakce sémantické informace byla jednou z hlavních řešených problematik. Jak již bylo řečeno dříve, byl zvolen přístup založený na parsování textu pomocí sémantických bezkontextových gramatik.

Při prvních experimentech byla použita již existující implementace tohoto konceptu, ^{TODO39} používající standard SRGS ^{TODO40}

Během těchto prvních pokusů ale bylo zjištěno, že funkčnost, kterou nabízí existující implementace, nebude pro potřeby této práce postačovat. Kvůli tomu byla navržena, implementována a otestována vlastní realizace stejného konceptu analýzy textu pomocí bezkontextových gramatik, která lépe vyhovovala ostatním zde navrženým, realizovaným a popsáným postupům a metodám.

Detailedy této vlastní implementace a rozdíly oproti existujícím variantám jsou popsány v následujících kapitolách.

4.2.1 Problémy s existující implementací a SRGS

Při práci s existující implementací SRGS standardu (SpeechCloud) ^{TODO41} ^{TODO42} vyvstaly následující problémy:

1. Speciální pravidlo \$GARBAGE

První problém, který bylo potřeba vyřešit, byla absence pravidla \$GARBAGE. Jedná se o speciální pravidlo definované ve standardu SRGS, které má tu vlastnost, že dokáže reprezentovat libovolnou promluvu nebo libovolný token.

Lze tak velmi efektivně využít v situacích, kdy je potřeba specifikovat výplňová slova, která dopředu není možné odhadnout a nebo nás při analýze textu nezajímají.

Existující implementace však neměla toto speciální pravidlo implementováno. To představovalo problém, protože možnost specifikovat libovolný token je jeden ze

³⁹referovat na SpeechCloud

⁴⁰referovat SRGS

⁴¹referovat SC správně formálně

⁴²zmínit, že SC neměl danou funkčnost proto, že to prostě nebylo potřeba

základních konceptů, na kterém byl celý navržený koncept sémantické analýzy postaven.

2. Návratová datová struktura

Existující implementace nevracela ze svého API celý derivační strom, ale pouze list tagů, které se v derivačním stromu nacházely. ^{TODO43} To znamená, že byla ztrácena hierarchická informace derivačního stromu spolu s informací o konkrétních pravidlech, které byly během zpracování textu použité.

Tyto informace nebyly zcela nezbytné pro další postup, avšak jejich absence by vyžadovala výrazně složitější definici parsovacích pravidel v bezkontextových gramatikách a s tím samozřejmě spojený složitější systém na zpracování obdržených výsledků.

Dostupnost celých derivačních stromů by tedy byla výhodná ve smyslu zjednodušení dalšího postupu.

3. Řešení nejednoznačných situací

Třetí problém vycházel z toho, že SRGS standard umožňuje existenci některých situací, ^{TODO44} kde není jednoznačně dáno, jak má parsování probíhat, a teoreticky je možné získat z jednoho vstupu více různých derivačních stromů. ^{TODO45}

Existující implementace tyto situace dokázala zpracovat, avšak vrátila pouze jeden výsledek, který byl považovaný podle nějakého kritéria za nejlepší. ^{TODO46}

Pro sémantickou analýzu textu potřebnou pro získání testovaného popisu by ale bylo výhodné, kdyby bylo možné toto kritérium změnit, a nebo ještě lépe, získat všechna možná řešení.

Po důkladném zvážení těchto problémů bylo rozhodnuto, že jako nejlepší způsob řešení bude implementace vlastního parseru.

Během reimplementace vlastního parseru byla jako základní reference využita specifikace SRGS, ^{TODO47} která přesně popisuje chování, funkčnost i syntax gramatik.

⁴³ ověřit?

⁴⁴ referovat přímo do SRGS specifikace kde to tam je?

⁴⁵ příklad

⁴⁶ doplnit podle jakého kritéria (délka textu)?

⁴⁷ přidat odkaz

V průběhu reimplementace byly ovšem za účelem zjednodušení vynechány některé části SRGS standardu, které by pro sémantickou analýzu textu v této práci nebyly nijak užitečné a naopak byly přidány nějaké funkce navíc, které byly pak využity ve zbytku práce.

Výsledkem tak byl nový formát, označený „Semantic Parsing Grammar Format“ (SPGF). K němu byly samozřejmě vytvořené i základní softwarové nástroje, které nabízí

- kontrolu syntaxe a případné hlášení o syntaktických chybách
- TreeSitter modul proobarvení SPGF kódu v textových editorech ^{TODO48}
- parser SPGF syntaxe s validací obsahu gramatiky i jednotlivých pravidel
- parser přirozeného textu, který využívá právě SPGF gramatiky

4.2.2 Semantic Parsing Grammar Format (SPGF)

Základním a nejvyšším prvkem SPGF je *gramatika*. Na rozdíl od SRGS definice, SPGF gramatika nevyžaduje žádnou hlavičku a nepodporuje dodatečné prvky, jako deklaraci jazyka nebo meta-dat. Gramatika má podobu textového souboru, ve kterém je definována množina parsovacích pravidel.

Každé pravidlo se skládá ze svého názvu, který je uvozený symbolem dolaru „\$“. Název pravidla slouží jako identifikátor, musí být tedy unikátní v dané gramatice. Druhou částí pravidla je expanze, která reprezentuje tělo pravidla. Mezi názvem a expanzí pravidla je znak rovnítka „=“. Pravidlo je vždy zakončeno středníkem. Například:

$$\$climbing = \underbrace{(\text{leze} \mid \text{šplhá})}_{\text{název pravidla}} \underbrace{[\text{na} \mid \text{po}]}_{\text{expanze (tělo pravidla)}};$$

Tělo pravidla se skládá z takzvaných *alternativ*, které jsou oddělené znakem „|“. Tato *alternativa* je pak dále definována jako posloupnost *elementů*. Element má tři varianty, může jím být *token*, *reference* nebo *sekvence*. Například:

$$\$dog = \underbrace{\text{pejsek}}_{\text{alternativa 1}} \mid \underbrace{(\text{pes} \mid \text{psík})}_{\text{alternativa 2}} \mid \underbrace{\text{nejlepší}}_{\text{token}} \underbrace{\text{přítel}}_{\text{token}} \underbrace{\text{člověka}}_{\text{token}};$$

⁴⁸referenci na TreeSitter, případně ukázka?

Nejjednodušší variantou *elementu* je *token*. Jedná se o terminální symbol (literál), který udává, jaký řetězec je při parsování právě přípustný. Token je možné chápat jako konkrétní slovo, které je v dané pozici v textu očekávané. Může se skládat z jednoho nebo více znaků, která mohou být písmena, číslice, tečka, pomlčka, podtržítka nebo dvojtečka. Součástí definice tokenu může také být (volitelně) definice opakování a případně libovolný počet tagů. Opakování elementů a tagy budou detailněji popsány později. ^{TODO49}

Druhou variantou elementu je *reference*. Tímto pojmem se rozumí odkaz na jiné pravidlo, a nebo odkaz na nějaké speciální pravidlo s vyhrazeným názvem. Reference začíná symbolem dolara „\$“ po kterém následuje okamžitě (bez mezery) jméno referovaného pravidla definovaného ve stejně gramatice, nebo speciálního pravidla. SPGF umožňuje referovat pravidla rekurzivně a to jak přímo (pravidlo obsahuje referenci na samo sebe), tak v cyklu. ^{TODO50}

Formát SPGF definuje 5 speciálních pravidel s vyhraněnými jmény. Tři z nich jsou převzaté ze SRGS standardu, zbylé dvě byly přidány navíc, aby bylo možné vynutit jistá chování parsovacího algoritmu, která jsou v SRGS automaticky, ale v SPGF nikoli, kvůli odlišné strategii prohledávání (viz později). ^{TODO51}.

1. Speciální pravidlo \$GARBAGE:

Akceptuje libovolný jeden token a posune parsování o akceptovaný token dále.

Speciální pravidlo \$GARBAGE je užitečné, když je potřeba vyjádřit, že na dané pozici v textu může být libovolné slovo, které buďto nedokážeme předvídat, nebo nás nezajímá.

2. Speciální pravidlo \$NULL:

Akceptuje prázdný řetězec. To znamená, že je vždy úspěšně aktivováno a nikdy neposune parsování o žádný znak dále.

Lze využít například pro situace, kde je žádoucí přidat do derivačního stromu nějaké tagy, ^{TODO52} aniž by byl učiněn postup v parsovaném textu.

⁴⁹doplnit referenci na pozdější popis

⁵⁰příklad?

⁵¹doplnit referenci na pozdější kapitolu

⁵²podle wiki: parsing tree = derivační strom?

3. Speciální pravidlo \$VOID:

Vždy selže, bez ohledu na parsovaný text. Nikdy tedy není úspěšné a sekvence obsahující \$VOID také nikdy nebude úspěšná.

Lze využít například pro dočasné blokování některých jiných pravidel, nebo pro zdůraznění, že daná sekvence tokenů se nesmí v textu vyskytovat.

4. Speciální pravidlo \$END:

Akceptuje konec textu. Bude úspěšné právě ve chvíli, kdy je parsování na konci textu a nezbývá již žádný znak ke zpracování.

Umístění speciálního pravidla \$END na konec nějaké expanze tak způsobí, že celá expanze bude úspěšná pouze ve chvíli, kdy dokáže pojmotit text beze zbytku až do konce.

5. Speciální pravidlo \$BEGIN:

Akceptuje začátek textu. Bude úspěšné právě ve chvíli, kdy je parsování na začátku textu a žádný znak ještě nebyl zpracován.

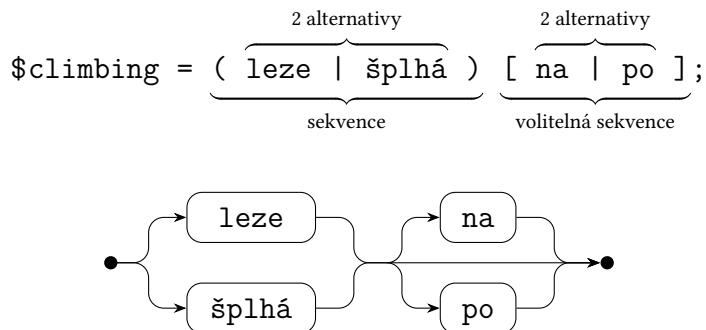
Umístění speciálního pravidla \$BEGIN na začátek expanze tak způsobí, že celá expanze bude úspěšná pouze ve chvíli, kdy dokáže pojmotit text od prvního znaku.

Poslední podobu, kterou *element* může mít, je *sekvence*. Sekvence může být „obyčejná“ nebo „volitelná“.⁵³ V obou případech se jedná o *alternativy* oddělené znakem „|“, list alternativ je ohraničený kulatými, respektive hranatými, závorkami.

Důvodem pro existenci *sekvencí* je to, že umožňují seskupování elementů a přehlednější tvorbu složitějších konstrukcí, podobně jako jsou v matematice používané závorky v aritmetických či algebraických výrazech. Volitelná sekvence (s hranatými závorkami) je pouze syntaktická zkratka pro „obyčejnou“ sekvenci, která má minimální počet opakování roven nule (takže se nemusí vůbec v textu vyskytovat) a maximální počet opakování roven jedné. Jedná se o obdobu operátoru „?“ v regulárních výrazech.

Příklad zápisu pravidla s oběma typy sekvencí a jeho schématická reprezentace je pak na Obrázku 10.

⁵³ přidat referenci na místo, kde je popsáno opakování



Obrázek 10: Pravidlo \$climbing ve formátu SPGF a v podobě konečného automatu

4.2.3 Vstupní a výstupní body SPGF gramatiky

TODO54 Základní rozdíl v parsovací strategii SPGF oproti SRGS spočívá v tom, co je považováno za úspěšný konec parsování. SRGS implementace TODO55 jako úspěšně dokončené parsování považuje takové situace, kde poskytnutý text byl pokryt daným pravidlem, obojí (pravidlo i text) vyčerpané (využité) beze zbytku od začátku do konce. Naproti tomu SPGF parser nekontroluje využití celého textu, pouze celého pravidla. Pokud tedy pravidlo pokryje pouze prvních několik slov a zbytek textu nebude „pasovat“, SRGS bude tuto situaci brát jako za neúspěch, zatímco SPGF to bude považovat za úspěch.

Dalším rozdílem mezi SRGS a SPGF je ten, že SRGS gramatiky poskytují pouze jeden vstupní bod (pravidlo označené klíčovým slovem `root`). Z toho pak plyne, že implementace SRGS mohou pro jeden vstupní text vrátit nejvýše jeden parsovací strom. TODO56

SPGF oproti tomu umožňuje specifikovat více pravidel jako „veřejná“ klíčovým slovem `public` a tím tak gramatika bude mít více vstupních bodů. Parser po načtení SPGF gramatiky vezme všechna pravidla označená jako `public` a pokusí se pomocí nich zpracovat poskytnutý text. Výsledky úspěšných parsování jsou pak vrácené jako výstupy v podobě derivačních stromů, se kterými je možné dále pracovat.

Pro příklad uvažujme vstupní text: „hnědý pes utíká pryč“ a gramatiku na Listing 2. TODO57

```

1 public $sentence = $color $object $action;
2 $color = modrý | zelený | hnědý;

```

⁵⁴lepší název kapitoly

⁵⁵referovat konkrétně na SC?+ ověřit faktickou správnost

⁵⁶ověřit?

⁵⁷zlepšit barvy

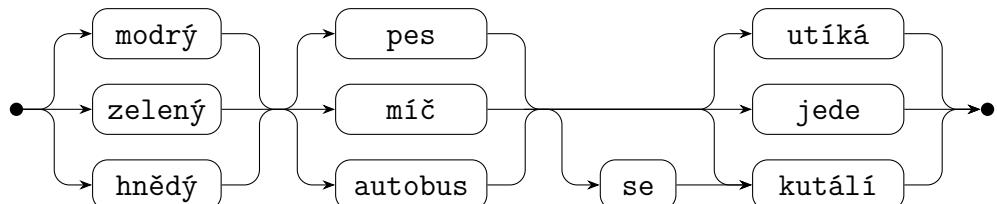
```

3 | $object = pes | míč | autobus;
4 | $action = utíká | jede | [se] kutálí;

```

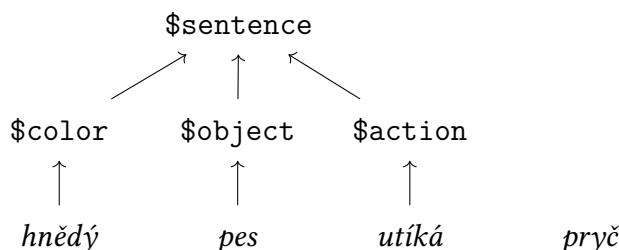
Listing 2: Ukázková SPGF gramatika

Tuto gramatiku z Listing 2 lze zakreslit jako konečný automat na Obrázku 11.



Obrázek 11: Schéma gramatiky jako konečného automatu

Při zpracování textu by mohl vzniknout následující derivační strom, který je na Obrázku 12:



Obrázek 12: Ukázkový derivační strom

Jak je vidět, poslední slovo „*pryč*“ není součástí derivačního stromu, protože parsovací pravidlo *\$sentence* jej nedokáže pokrýt. V případě SRGS by tento derivační strom byl považovaný za neúspěch, zatímco SPGF vrátí zobrazený parsovací strom jako úspěšný výsledek.

4.2.4 Tagy v SPGF

K elementům je možné přiřadit *tagy*. Každý tag je libovolný text ohraničený složenými závorkami „{}“. Za každý element (s výjimkou speciálního pravidla *\$VOID*) je možné připsat libovolný počet těchto tagů, oddělených mezerami.

Tagy v SPGF nijak neovlivňují parsovací proces, slouží pouze jako dodatečná informace, kterou je možné k tokenům nebo jiným elementům přidružit.

V této práci jsou použité k tomu, aby obsahovaly dodatečnou sémantickou informaci. V definice tagů se SPGF liší od SRGS v tom, že tagy nejsou považované za samostatné elementy a není tak možné mít například pravidlo sestávající pouze z tagů nebo začínající tagem. Pro situace, kdy je žádoucí začít pravidlo nějakým tagem, je možné použít speciální pravidlo \$NULL a k němu přiřadit dané tagy.

Příklad gramatiky obsahující tagy je na Listing 3. K této ukázkové gramatice jsou pak na Obrázku 13 zobrazené tři derivační stromy pro promluvy „*pes spí*“, „*kapr plave*“ a „*delfín plave*“. Tagy jsou v derivačních stromech znázorněné zelenou barvou.

V tomto příkladu tagy přidávají informace o počtu nohou jednotlivých zvířat a jejich biologickou třídu. Dále je pomocí tagů označeno, která činnost je klidová a která v pohybu, a která část textu je podmět a která přísudek.

```

1 public $věta = $zvíře {podmět} $činnost {přísudek};
2 $zvíře = (pes {nohy=4} | delfín {nohy=0}) {savec} | kapr {nohy=0} {ryba};
3 $činnost = spí {v klidu} | plave {pohyb};
```

Listing 3: Ukázková SPGF gramatiky s tagy



Obrázek 13: Derivační stromy s tagy

TODO 58

⁵⁸ doplnit (někam) formální definici syntaxe, případně jako přílohu?

4.2.5 Parsovací strategie a opakování v SPGF

Jak bylo zmíněno v sekci 4.2.2, v gramatice je možné ke každému elementu, s výjimkou speciálních pravidel \$END, \$BEGIN, \$VOID a \$NULL, přiřadit definici opakování.

Toto opakování udává, jaký je maximální a minimální počet bezprostředních opakování daného elementu během zpracování textu. Například pokud má daný token specifikováno, že jeho minimální počet opakování je 2 a maximální 3, tak v analyzovaném textu bude odpovídající slovo očekáváno dvakrát nebo třikrát po sobě.

Základní syntaxe je převzata ze standardu SRGS, tedy dvě přirozená čísla oddělená pomlčkou (bez mezer) a ohraničené špičatými závorkami: „ $\langle \text{min}-\text{max} \rangle$ “. Pro pohodlnější zápis bylo definováno několik dalších alternativních způsobů zápisu, viz Tabulka 3. Pokud element žádné opakování nemá specifikované, implicitní hodnota je 1. ^{TODO59}

Syntax	Počet opakování		Poznámka
	min	max	
$\langle m-n \rangle$	m	n	$m, n \in \mathbb{N}^0 \wedge m \geq n$
$\langle m \rangle$	m	∞	v implementaci max = $2^{32} - 1$
$\langle m-m \rangle$	m	m	ekvivalentní s „ $\langle m-n \rangle$ “
$\langle * \rangle$	0	∞	ekvivalentní s „ $\langle 0-\rangle$ “
$\langle + \rangle$	1	∞	ekvivalentní s „ $\langle 1-\rangle$ “
$\langle ? \rangle$	0	1	ekvivalentní s „ $\langle 0-1 \rangle$ “

Tabulka 3: Možné způsoby zápisu opakování v SPGF

S opakováním použitím stejného elementu je úzce spojena i strategie parsování. Jedná se o způsob, jakým jsou řešené neurčité situace, kde je možné postupovat více způsoby. Tyto situace mohou vznikat právě v místech opakování elementů, nebo tam, kde je k dispozici více aplikovatelných alternativ. ^{TODO60}

V existující implementaci SRGS, se kterou byly prováděny první experimenty, byl implementován pouze takzvaný „hladový algoritmus“ ^{TODO61}, běžně označovaný pod ang-

⁵⁹příklad

⁶⁰příklad pro oba způsoby

⁶¹ověřit

lickým názvem *greedy matching*. Pro potřeby sémantické analýzy textu v této práci bylo ovšem zjištěno, že pouze greedy ^{TODO62} algoritmus nebude zcela postačovat.

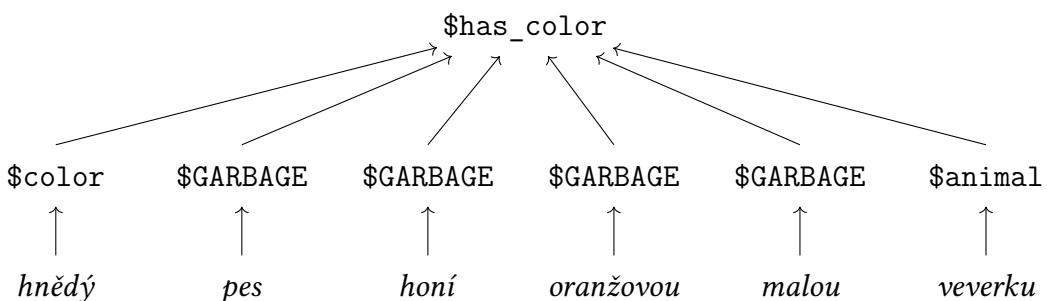
Příkladem takové situace by mohla být analýza textu

hnědý pes honí oranžovou malou veverku

s cílem zjistit vlastnosti objektů (v tomto případě zvířat), zde konkrétně jejich barvy. Pravidlo pro zachycení barvy zvířete by mohlo vypadat takto: ^{TODO63}

```
public $has_color = $color $GARBAGE<*> $animal;
```

kde \$color a \$animal jsou reference na pravidla akceptující konkrétní slova pro barvy a zvířata. Speciální pravidlo \$GARBAGE je zde definováno s operátorem opakování „<*>“, což lze chápat tak, že mezi samotnou barvou a zvířetem může být libovolný počet dalších slov, která nejsou podstatná. Při zpracování textu pak ale dochází k problému, že greedy algoritmus vrátí derivační strom zakreslený na Obrázku 14:



Obrázek 14: Derivační strom znázorňující problém s greedy algoritmem

Jak je na derivačním stromu na Obrázku 14 vidět, následující sémantická analýza by došla pravděpodobně k závěru, že v textu byla informace o hnědé veverce - což je chyba. Zde speciální pravidlo \$GARBAGE s neomezeným počtem opakování v kombinaci s greedy algoritmem způsobilo, že toto speciální pravidlo bylo „příliš agresivní“ a aktivovalo se i v případech, kdy by již bylo možné použít následující element.

Z tohoto důvody byla do systému implementována druhá strategie, která se běžně označuje jako *lazy matching*. ^{TODO64} Zatímco greedy matching se v každém okamžiku snaží

⁶²můžu takhle použít anglické názvy? přijdou mi lepší než české „hladový“ (greedy), „líný“ (lazy) nebo „důsledný“ (thorough)

⁶³vyměnit za poctivý listing s celou gramatikou?

⁶⁴zkusit překládat?

posunout parsování co nejdále, tato lazy strategie se naopak snaží použít element co nejméněkrát ^{TODO65}

Také by se dalo říci, že pokud parsovací algoritmus narazí na element e_1 následovaný elementem e_2 , pak:

- greedy strategie se vždy nejdříve pokusí e_1 použít (opakováně po sobě) a postoupí na e_2 až v moment, kdy již není možné dále opakovat e_1 ,
- lazy strategie se vždy pokusí nejdříve postoupit na element e_2 a vrátí se k e_1 pro opakování použití až v moment, kdy e_2 není možné použít

^{TODO66}

Implementace lazy strategie umožnila zachytit sémantiku, kterou by bylo obtížné získat pomocí greedy algoritmu. Po několika dalších experimentech ovšem bylo zjištěno, že ani tato strategie nebude sama o sobě postačovat pro všechny potřeby sémantické analýzy. Hlavním problémem s lazy strategií bylo to, že v textech, kde bylo více možných vyjádření stejného typu, byla detekována vždy jen ta nejkratší.

⁶⁵je „nejméněkrát“ česky správně?

⁶⁶zmínit že to je obdoba prohledávání do šírky/hloubky?

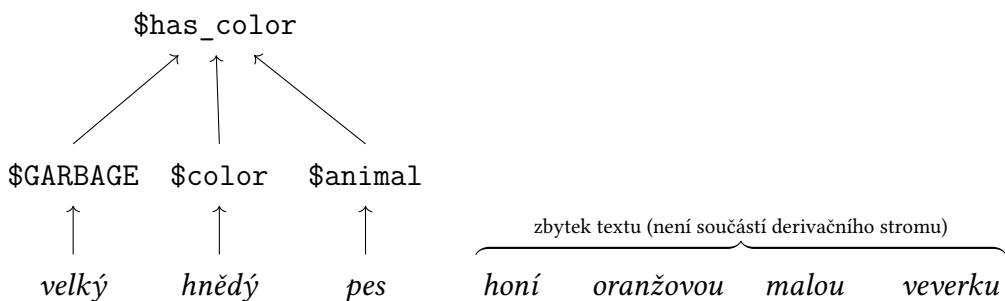
Například předchozí úloha určování barev zvířat se vstupním textem

velký hnědý pes honí oranžovou malou veverku

by mohla být zpracována SPGF pravidlem:

```
public $has_color = $GARBAGE<*> $color $GARBAGE<*> $animal;
```

V tomto případě bylo přidáno na začátek pravidla další `$GARBAGE<*>`, které při použití lazy algoritmu způsobí, že text nemusí začínat přímo barvou, ale je možné hledat barvy a zvířata až později v textu. Derivační strom tohoto příkladu je zakreslen na Obrázku 15.



Obrázek 15: Derivační strom znázorňující problém s lazy algoritmem

Z derivačního stromu na Obrázku 15 je možné usoudit, že následující sémantická analýza by pravděpodobně detekovala, že v textu se nacházela informace o hnědém psovi - to je správně. Nicméně je dále možné pozorovat, že v textu se vyskytovala také informace o oranžové veverce, kterou by systém nedokázal takto zachytit.

Z toho důvodu byl implementována ještě třetí strategie, která byla označena jako „*thorough matching*“. Myšlenka byla taková, že by bylo vhodné, aby algoritmus v místě nejednoznačnosti nemusel rozhodovat o tom, který postup je nejlepší, ale aby místo toho uvažoval všechny možnosti. Tento přístup se tedy od předchozích dvou liší v tom, že může vrátit pro jeden vstup více výstupů.

Základní koncept thorough strategie spočívá v tom, že v místě, kde by bylo možné po stupovat více způsoby, je řešení rozděleno do více paralelních větví (jedna pro každou možnost) a každá větev je dále zpracovávána nezávisle. Tímto způsobem se rekurzivně vytváří strom možných řešení, jehož listy odpovídají finálním derivačním stromům. Po ukončení všech těchto paralelních parsování jsou úspěšné výsledky vráceny jako seznam derivačních stromů.

Na výstupu tedy bude množina všech možných způsobů, jakým bylo možné derivační strom pro dané pravidlo a vstupní text sestavit. To následně zaručí, že během zpracování textu nebyla opomenuta žádná sémantika (za předpokladu, že jsou expertem správně sestavena parsovací pravidla).

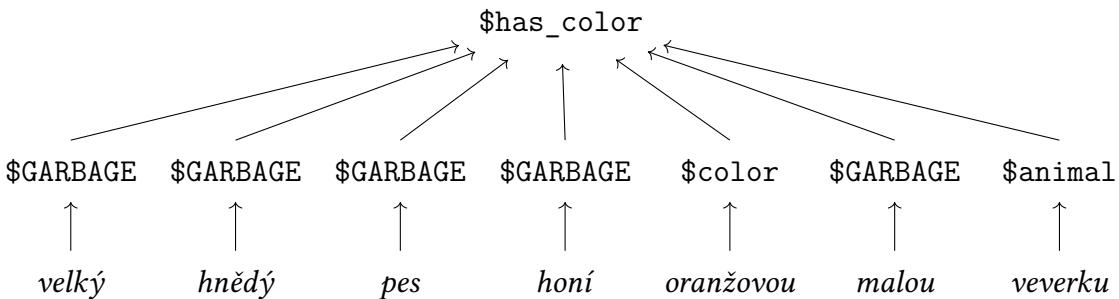
Například pro předchozí úlohu určování barev zvířat se vstupním textem

velký hnědý pes honí oranžovou malou veverku

a SPGF pravidlem:

```
public $has_color = $GARBAGE<*> $color $GARBAGE<*> $animal;
```

by výstupem byly 3 derivační stromy. Dva z nich by byly shodné s výstupem greedy a lazy strategie (viz Obrázky 14 a 15), třetí (nový) derivační strom je na Obrázku 16.



Obrázek 16: Derivační stromy z thorough algoritmu

Celkově by tedy z tohoto výstupu bylo možné získat informace o tom, že v textu byla hnědá veverka, hnědý pes a oranžová veverka. Dvě z těchto informací jsou správně, pouze informace o hnědé veverce je chybná, ta se v textu nevyskytla.

Jedná se o jednu z nevýhod thorough přístupu, že bude v textu hledat i takové informace, které z něj neplynou. ^{TODO67} Tento problém byl vyřešen tak, že získané sémantické informace jsou porovnané s referenčním popisem obrázku a ty, které se v referenčním popise nevyskytují, jsou považované za falešně detekované a jsou z výstupu odstraněné (s výjimkou hodnot atributů, řešení chybných hodnot je popsáno v kapitole 3.4).

Tímto způsobem je pak v testovaném popisu pouze podmnožina referenčního popisu. Jedná se o jednoduchý a efektivní způsob řešení falešně detekovaných sémantických in-

⁶⁷ zmínit, že to je „false positive“ typ chyby? Případně psát typy chyb pak do teorie?

formací. Také ale bohužel představuje omezení v tom smyslu, že neumožňuje detekovat sémantiku, která se v textu skutečně nacházela, ale nebyla zanesena do referenčního popisu - to klade očekávání na kvalitu referenčního popisu od experta. Sestavení nějakého složitějšího algoritmu pro detekci falešně extrahované sémantiky představuje jedno z možných rozšíření do budoucna.

Další nevýhodou thorough strategie je její výpočetní náročnost. Vzhledem k rekurzivnímu charakteru algoritmu a způsobu vytváření nových paralelních větví se naskytá riziko kombinatorické exploze počtu paralelních větví. Během testování se však ukázalo, že pro moderní výpočetní techniku nepředstavují vstupy o velikosti desítek až stovek tokenů žádný problém. Detailnější analýza výpočetní a paměťové náročnosti pro větší vstupy je nad rámec zadání a byla ponechána pro budoucí práce.

Posledním problémem, který bylo potřeba vyřešit v rámci parsovacích strategií, byl způsob, jakým strategie volit a přepínat. V první verzi parseru se jednalo o globální přepínač, kdy bylo možné před samotným parsováním zvolit, jaká strategie má být použita. Bylo ale zjištěno, že pro některé situace by bylo výhodné, aby bylo možné změnit výchozí strategii pro konkrétní pravidla nebo jejich části.

Například při zpracovávání číselných údajů je vhodná greedy strategie. Je totiž potřeba, aby posloupnost sobě jdoucí slov, které reprezentující číselný údaj, byla brána jako jeden celek.

Jiný příklad pak může být již výše zmíněný začátek pravidla \$GARBAGE<*>, který libovolný počet prvních slov považovat za výplň a hledat tak význam až později v textu. Pro tuto situaci je naopak greedy algoritmus zcela nevhodný, protože by celý text byl zpracován hned tímto počátkem. Zde je vhodná lazy strategie, která ve své podstatě bude dávat přednost následujícím elementům v pravidle a tento počátek bude používat až když zbytek pravidla selže.

Z těchto důvodů byla rozšířena syntaxe pro operátor definující opakování. Před samotné hodnoty opakování (tedy hned za otevírací špičatou závorku „<“) je možné napsat jedno z písmen L, G, T následované dvojtečkou a poté zbytek definice opakování. Tato písmena odpovídají jednotlivým strategiím (lazy, greedy, thorough) a umožňují tak vynutit danou strategii pro dané pravidlo nebo jeho část. Ve výsledku tak operátor opakování může vypadat například „<G:1-3>“, „<L:>“, nebo „<T:2->“.

5 Závěr