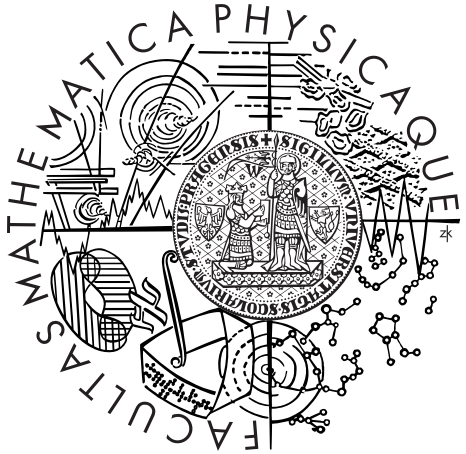
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Robert Brunetto

External sources of axioms in lean theorem proving

Katedra teoretické informatiky a matematické logiky

Vedoucí diplomové práce: RNDr. Martin Suda
Studijní program: Informatika
Studijní obor: Teoretická informatika

Praha 2012

Děkuji prof. RNDr. Petrovi Štěpánkovi DrSc. za to, že mi umožnil na tomto tématu pracovat a za to, že byl vedoucím této práce, dokud to bylo možné. Veliké díky také patří RNDr. Martinovi Sudovi za vše s čím mi pomohl, za konzultace, které mi během práce poskytoval a za to, že se následně po úmrtí profesora Štěpánka stal vedoucím této práce. Dále také děkuji Dr. Geoffovi Sutcliffovi za rychlé odpovědi na mé dotazy.

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V Praze dne 1. 8. 2012 Robert Brunetto

Název práce: External sources of axioms in lean theorem proving

Autor: Robert Brunetto

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí diplomové práce: RNDr. Martin Suda, Katedra teoretické informatiky a matematické logiky

Abstrakt: Automatické dokazovače vět lze upravit tak, aby mohly používat externí zdroje axiomů. To bylo nedávno zkoumáno v kombinaci se saturačním dokazovačem SPASS-XDB, který během saturační smyčky odesílá dotazy o externí axiomy a přijímá odpovědi. Lean dokazovače používají sémantický tableau kalkulus. Proto je potřeba pro ně použít jiný přístup. Tato práce představuje myšlenku, že by bylo možné od sebe oddělit dokazování a komunikaci s externími zdroji axiomů a to tak, že se nejprve vygeneruje schématický důkaz, u kterého se až následně kontroluje zda mu odpovídají nějaká externí data a zda ho lze doplnit na důkaz. Součástí této práce je upravená verze dokazovače LeanCoP tak, aby demonstrovala tuto myšlenku a zároveň se ukazuje, že je tento program často efektivnější než SPASS-XDB - dokonce ho poráží i na všech vzorových příkladech z jeho webové stránky.

Klíčová slova: automatické dokazování vět, externí zdroje axiomů, LeanCoP, LeanCoP-XDB

Title: External sources of axioms in lean theorem proving

Author: Robert Brunetto

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: RNDr. Martin Suda, Department of Theoretical Computer Science and Mathematical Logic

Abstract: Automated theorem provers can be modified in order to use external sources of axioms. This was recently searched in combination with saturation based theorem prover SPASS-XDB. It sends queries to external sources and receives answers during its saturation loop. Lean theorem provers are based on semantic tableau calculus. That's why they need to use different approach. An idea that proving is separated from communication is beeing introduced by this work. Prover generetes so called schematic proof which is later checked if it can be filled with external data so the proof can be completed. This work demonstates this idea on modified version of LeanCoP.

Keywords: automated theorem proving, external sources of axioms, LeanCoP, LeanCoP-XDB

Obsah

Úvod

Automatické dokazování má mnoho uplatnění, nejen dokazování nebo kontrolu důkazů matematických vět, ale například i u verifikace softwaru, verifikace hardwaru a v dalších oborech. Proto vzniklo mnoho programů pro automatické dokazování vět (dále jen dokazovačů). Dokazovač na vstupu dostane obvykle konečnou množinu formulí (axiomů) a k nim jednu domněnku. Cílem dokazovače je dokázat, že z axiomů vyplývá daná domněnka nebo říct, že to není pravda. Výstupem by pak měla být jedna z následujících možných odpovědí:

- Domněnku nelze z axiomů dokázat.
- Domněnku se povedlo z axiomů dokázat. V tomto případě některé dokazovače umí i říct odpovídající důkaz.
- Dokazovač neví a nebo nestihne doběhnout v přiděleném časovém limitu.

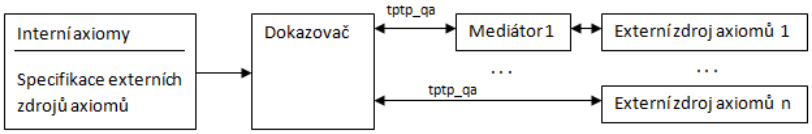
Jazyk, v němž se zapisují axiomy a domněnka, se může mezi dokazovači lišit. Z důvodu snažšího použití různých dokazovačů na stejný problém a z důvodu umožnění porovnání, vznikl standardní jazyk TPTP pro zápis problémů, kterému dnes většina dokazovačů pro logiku prvního řádu rozumí. [16]

V posledních letech vzrůstala potřeba automatického dokazování ve velkých teoriích. Velká teorie je ta, která má hodně axiomů, a ve které je dokazatelné velké množství vět, přestože k důkazu mnoha z nich stačí jen několik málo axiomů. [15] Například ontologie SUMO [8], báze znalostí Cyc [4], matematická knihovna Mizar [12], báze znalostí YAGO [13], WordNet [7] a slovník MeSH [6] jsou ve formátu vhodném pro automatické dokazování nebo do něj byli přeloženy. Některé z těchto teorií obsahují až milióny atomických faktů, převážně pozitivních ground axiomů. Velké teorie také vznikají z dynamických a z výpočetních zdrojů dat, například z online bází znalostí, z bioinformatických databází a nástrojů, z matematických systémů, z nástrojů pro hledání lemmat a z webových služeb.

Takovéto zdroje mohou poskytovat i nekonečné množství axiomů. Velké teorie tvoří pro automatické dokazovače vět úplně jiný typ výzev než malé teorie. [15]

SPASS-XDB [15] vznikl rozšířením známého dokazovače SPASS, aby pracoval právě s velkými teoriemi v predikátové logice prvního řádu. Jazyk TPTP byl rozšířen, aby se v něm kromě běžných formulí (axiomy a domněnka) dala zapsat i specifikace tzv. externího zdroje axiomů. To je program, který běží na lokálním nebo vzdáleném počítači, který na požádání dokazovači poskytuje axiomy. Dokazovač při svém spuštění načte axiomy, domněnku a specifikaci externího zdroje. Ta obsahuje popis toho, jaký typ axiomů je zdroj hypoteticky schopný poskytnout. Dokazovač může během uvažování používat axiomy, které má načtené a může o další požádat externí zdroj axiomů. S ním komunikuje protokolem uvedeným v jeho specifikaci. Za tímto účelem vznikl spolu se SPASS-XDB i protokol tptp_qa, který komunikuje s externími zdroji axiomů. Případně, pokud nějaký zdroj dat nevznikl za účelem být externím zdrojem axiomů, tak pravděpodbně nebude komunikovat prostřednictvím protokolu tptp_qa. Může ale existovat program, který se dokáže chovat jako prostředník a protokol tptp_qa překládat do vhodného formátu. Takovémuto programu budeme říkat mediátor. Těch me-

zi zdrojem a dokazovačem může být i více a nemusí tam být žádný. Situace je znázorněna na následujícím obrázku:



Toto schéma bylo prvně vymyšleno pro SPASS-XDB. Přesto lze za kostičku dokazovač v tomto obrázku dosadit i některý jiný program než SPASS-XDB. Právě o to se pokusíme v této práci.

SPASS-XDB je saturační dokazovač a jeho myšlenka je jednoduchá - saturační smyčka je jen jednoduchým způsobem rozšířená o komunikaci s externími zdroji.

Lean dokazovače, jako například LeanCoP [11] nebo LeanTAP [2], jsou založeny na sémantických tablech. Jejich rozšíření o externí zdroje axiomů vyžaduje jiný postup než u SPASSu-XDB, protože sémantická tabla předepisují striktnější postup dokazování, co se týče pořadí vytváření důkazu. To je rozdíl oproti rezoluci, která u saturačních dokazovačů dovoluje provádět inference mezi libovolnými klauzulemi. Způsob řešení tohoto problému bude pro predikátovou logiku prvního řádu v rámci této práce navrhnut a ukázkově implementován rozšířením LeanCoPu. Tomuto programu budeme dále říkat LeanCoP-XDB.

Domníváme se, že by mohl mít rozdílný přístup velký vliv na efektivitu. Proto oba přístupy následně otestujeme a vzájemně porovnáme. Porovnání provedeme, jak na příkladech získaných z webu SPASSu-XDB, které pracují s existujícími externími zdroji, tak v experimentech s příklady, které byli v rámci této práce za účelem porovnání vytvořeny (včetně dalších externích zdrojů dat a mediátorů k nim).

Hlavními přínosy této práce mají být:

1. Myšlenka, jak lean dokazovače upravit tak, aby mohly efektivně používat externí zdroje axiomů bez toho, aby jimi byli neúměrně zpomalovány.

2. Ukázková implementace - program LeanCoP-XDB.

3. Porovnání SPASSu-XDB a LeanCoPu-XDB na již existujících příkladech s použitím již existujících externích zdrojů a mediátorů.

4. Vymyšlení nových úloh pro dokazovače, které vedou na netriviální použití externích zdrojů axiomů.

5. Výroba externích zdrojů a mediátorů pro tyto nové úlohy.

6. Vzájemné porovnání LeanCoPu-XDB a SPASSu-XDB na nově vytvořených úlohách.

Dále, přestože to nebylo hlavním cílem, bude v této práci:

1. Provedena analýza použitého komunikačního protokolu a budou předloženy návrhy na jeho vylepšení.

2. Navržen nový přístup, jak lean dokazovače rozšířit o aritmetiku.

LeanCoP-XDB, ukázkové příklady, mediátory, externí zdroje a další věci, které vznikly v rámci této práce jsou přiloženy na CD. To mimo jiné dále obsahuje záznamy z provedených experimentů. Kompletní obsah přiložených souborů je v příloze D. Ovládání LeanCoPu-XDB je popsáno v uživatelské dokumentaci která je v příloze A.

0.1 Struktura následujícího textu

Celá práce je strukturována následovně: Kapitola 1 shrnuje všechny potřebné znalosti důležité k pochopení zbývajícího textu a vysvětluje, jak funguje SPASS a LeanCoP. Ve 2 je představena myšlenka schématických důkazů, která umožní lean dokazovače rozšířit o externí zdrote axiomů. Součástí této kapitoly je i analýza toho, jak lze schématický důkaz změnit v opravdový důkaz, co nejefektivněji. Poté budou ve 2. kapitole popsány detaily z implementace LeanCoPu-XDB. Kapitola 3 popisuje externí zdroje axiomů a mediátory, které budou použity v experimentech ve 4. kapitole a to jak ty, které byly převzaté tak i ty, které vznikly v rámci této práce. Ve 4. kapitole budou provedeny experimenty a vzájemného srovnání přístupu z této práce - LeanCoPu-XDB s přístupem SPASSu-XDB. V této kapitole se potvrdí domněnka, že jiný přístup má skutečně zásadní vliv na efektivitu a zároveň tato kapitola odhalí některá nečekaná pozorování o tom, jak LeanCoP-XDB a SPASS-XDB fungují. Dosažené výsledky i zajímavá nově zjištěná pozorování budou poté stručně zrekapitulovány v závěrečném shrnutí 5.

1. Příprava

Přestože existují různé logiky, budeme v celém zbytku textu bez toho, aby to dále bylo zmiňováno, předpokládat, že pracujeme v predikátové logice prvního řádu. Od čtenáře očekáváme, že ji zná v rozsahu libovolného základního kurzu. Potřebné znalosti lze případně najít například v první a třetí kapitole monografie [20]. To bude potřeba především zde v první kapitole, kde si částečně neformálně řekneme, jak funguje SPASS a SPASS-XDB a jak funguje LeanCoP, který pak budeme upravovat na LeanCoP-XDB.

Vstupem dokazovačů jsou formule zapsané v jazyku predikátové logiky. Ty obsahují kvantifikátory, logické spojky, proměnné, predikátové a funkční symboly. Stejným způsobem bývá zapsaná i domněnka, která se má ze zadaných axiomů dokázat.

Obvykle ale dokazovače k dokázání věty nepoužívají všechny možnosti predikátové logiky a zadání si zjednodušší ekvivalentními úpravami. Například implikaci lze nahradit pomocí disjunkce a negace nebo pomocí konjunkce a dvou negací takto: $A \Rightarrow B$ je ekvivalentní s $\neg A \vee B$ a také s $\neg(A \wedge \neg B)$. Právě tento druhý způsob, jak nahradit implikaci, je jednou z možností, jak nerozlišovat domněnku (B) od zbytku axiomů. Mezi ně se může přidat konjunkce tak, že mohou být spojeny do jediné formule (A).

Tedy místo $axiomy \Rightarrow domnenka$ lze dokazovat $\neg(axiomy \wedge \neg domnenka)$. Jedním ze způsobů, jak dokázat $\neg(axiomy \wedge \neg domnenka)$ je předpokládat $axiomy \wedge \neg domnenka$ a dokazovat spor. Tomu se říká dokazování zamítnutím[1]. Tuto taktiku používá jak SPASS a SPASS-XDB, tak i LeanCoP a LeanCoP-XDB. Všechny tyto dokazovače dále pokračují v ekvivalentních úpravách, dokud konjunkci $axiomy \wedge \neg domnenka$ nepřevedou do konjunkce formulí v následujícím tvaru: $Q_1 Q_2 \ldots Q_n c$, kde Q_i jsou kvantifikátory a c je klauzule. To je formule tvaru $L_1 \vee L_2 \vee \ldots \vee L_n$, kde L_i je literál. Literál je atomická formule nebo její negace. Atomická formule je formule tvaru $p(t_1, t_2 \ldots t_r)$, kde p je predikátový symbol a t_i jsou termy. Atomickým formulím budeme v tomto textu také říkat atomy.

Dále ale ještě provedou dokazovače další úpravy. Zbaví se existenčních kvantifikátorů a existenčně kvantifikované proměnné nahradí Skolemovskou konstantou nebo funkcí. Například takto: $\exists x : p(x)$ se nahradí za $p(a)$, kde a je nový konstantní symbol.

Není nezbytně nutné, aby byly tyto úpravy prováděny v pořadí, v jakém zde byli uvedeny, ale vždy si námi uvažované dokazovače upraví zadání do tvaru, kde jsou v konjunkci univerzálně kvantifikované klauzule.

Dále se však přístup SPASSu a LeanCoPu liší. Oba pro dokazování používají jiná inferenční pravidla. SPASS používá rezoluci a LeanCoP connection kalkulus. O inferenčních pravidlech řekneme, že jsou úplná, pokud vše, co platí se v nich dá dokázat. Dále říkáme, že jsou korektní, pokud je vše, co lze dokázat, pravdivé. O dokazovači nebo algoritmu, který provádí dané inference říkáme, že je úplný, resp. korektní, pokud vše, co platí, dokáže. O dokazovači nebo algoritmu říkáme, že je korektní, pokud vše, co dokáže, platí.

[1] V angličtině refutational theorem proving.

1.1 Saturace a SPASS

V této sekci si popíšeme, jak funguje dokazovač SPASS[19]. Komplementárními literály budeme rozumět takové dva literály, jeden s negací a jeden bez negace, které lze až na znaménko ztotožnit unifikací θ. Unifikací dvou atomů rozumíme takovou substituci proměnných, že po jejím provedení jsou atomy totožné. Ztotožňováním dvou literálů až na znaménko je myšleno ztotožnění jejich atomů, tedy odebrání negace.

Pokud máme dvě klauzule $\bigvee_{i\in\{1,\dots,n\}} L_i$ a $\bigvee_{i\in\{1,\dots,m\}} K_i$, které nemají společné proměnné[2] a které obsahují komplementární literály L_u a K_v, tak z nich lze odvodit třetí klauzuli následovně:

$$\frac{\bigvee_{i\in\{1,\dots,n\}} L_i \qquad \bigvee_{i\in\{1,\dots,m\}} K_i}{\theta((\bigvee_{i\in 1,\dots,n,i\neq u} L_i) \vee (\bigvee_{i\in 1,\dots,m,i\neq v} K_i))}$$

V horní části takto zapsaného pravidla jsou vstupní klauzule. Ty mají délky n a m. Ve spodní části je zapsaná výstupní klauzule. Ta má délku $n+m-2$. Vznikla spojením vstupních klauzulí, odebráním komplementárních literálů a následně se na takto vzniklou klauzuli ještě aplikovala substituce proměnných θ, která vznikla při ztotožňování komplementárních literálů.

Tomuto odvozovacímu pravidlu říkáme rezoluce.

Speciálně v případě, že máme dvě klauzule takové, že každá obsahuje jediný literál a oba tyto literály jsou komlementární, tak lze rezolucí odvodit prázdnou klauzuli. Tu považujeme za lživou a říkáme, že byl odvozen spor $\perp$.

Například:

$$\frac{p(t_1, t_2 \dots t_r) \qquad \neg p(t_1, t_2 \dots t_r)}{\perp}$$

Zjednodušeně řečeno SPASS používá následující algoritmus: Převede axiomy na klauzule a uloží si je do množiny Usable a množinu WorkedOff si nastaví na prázdnou. Z množiny Usable pak postupně odebírá klauzule a porovnává je se všemi klauzulemi ve WorkedOff a z nich vytváří nové klauzule rezolucí. Ty ukládá do množiny Usable a použitou klauzuli dá do množiny WorkedOff. Tato smyčka se opakuje, dokud se neodvodí spor nebo dokud není nalezena nejmenší nadmnožina množiny vstupních klauzulí, která je uzavřená na rezoluci. Takovému stavu říkáme saturace. Tento algoritmus je znázorněn v pseudokódu 1.1.

Jelikož SPASS používá úplný algoritmus tak, v případě dosažení saturace, tedy v případě doběhnutí algoritmu a nenalezení důkazu, bylo zjištěno, že existuje model klauzulí, u kterých měl algortimus dokázat spornost. Byl nalezen protipříklad.

Pro jiná zadání smyčka nikdy neskončí. To je způsobeno nerozhodnutelností logiky prvního řádu.

SPASS-XDB tento algoritmus mírně upravuje. V každé aktuální klauzuli hledá literály, pro které by mohl být komplementární literál v externím zdroji axiomů.

[2]Toho lze vždy dosáhnout přejmenováním proměnných

[3]V klauzulích Aktuální a D může být více párů komplementárních literálů. Proto procedura rezuluce v tomto pseudokódu vrací množinu nových klauzulí vzniklých jejich rezolucí pro různé komplementární literály. V případě, že klauzule na vstupu této procedury neobsahují žádné komplementární literály, rezoluce vrací prázdnou množinu.

```
Usable := množina klauzulí vzniklá ekvivalentními úpravami zadání;
WorkedOff := {};
while (Usable ≠ {}) {
    take Aktuální ∈ Usable;
    Usable := Usable - {Aktuální};
    WorkedOff := WorkedOff ∪ {Aktuální}
    foreach (D ∈ WorkedOff) {
        NovéKlauzule := rezoluce³(Aktuální,D)
        if (⊥ ∈ NovéKlauzule) return nalezen důkaz;
        Usable := Usable ∪ NovéKlauzule
    }
}
return nalezena saturace;
```

Obrázek 1.1: Pseudokód SPASSu

V případě nalezení takovýchto literálů je pak odeslán dotaz do externího zdroje, který má získat odpovídající axiomy. Teprve poté algoritmus SPASSu-XDB vstupuje do vnitřní smyčky a odvozuje nové fakty rezolucí. Před každým opakováním vnější smyčky pak ještě kontroluje, jestli už přišla odpověď na některý z dříve odeslaných dotazů a přidá ji do množiny Usable.

Ukázalo se [15], že toto už stačí k vyřešení některých problémů používajících externí zdroje axiomů. Zároveň to ale vede k tomu, že SPASS-XDB komunikuje s externími zdroji i v případech, u kterých by se dalo pouze z tvaru specifikace externích zdrojů říct, že zodpovězení dotazu nemůže nikdy pomoct k dokázání věty.

1.2 Sémantická tabla

Způsob dokazování, který používá LeanCoP, je o něco složitější. Proto si ho nejprve popíšeme neformálně. Konjunkce $axiomy \wedge \neg domnenka$ je převedena na konjunkci klauzulí, které budeme říkat M. Popíšeme si jeden z možných způsobů, jak na činnost LeanCoPu lze nahlížet. Máme dokázat, že z M vyplývá spor. Tedy, že M nemá model. To by znamenalo, že má být M ve všech strukturách nepravdivá.

K dokázání tohoto LeanCoP postupuje sporem. Předpokládá, že by existovala struktura[4] S, ve které by byla M splněná. Pak by ale musela být každá klauzule z M v S splněná. Z toho plyne, že by musel být v každé klauzuli z M alespoň jeden literál, který by byl v S splněný.

Právě toho LeanCoP využije. Dojde ke sporu s existencí takovéto struktury S tak, že v M bude hledat klauzuli takovou, že je každý její literál nesplněný v S. Tomu odpovídá pravidlo start v kalkulu uvedeném níže, které díky nedeterminismu najde vhodnou klauzuli (tj. klauzuli nesplněnou v S) klauzuli.

Před tím, než ale popíšeme, jak se ověří, že klauzule je vhodná, tak úlohu ještě malinko zobecníme. Místo nesplněnosti klauzule v S budeme ověřovat nesplněnost klauzule v S za dodatečných předpokladů, to bude množina literálů, kterým budeme říkat $Path$.

[4]Struktura se skládá z nosné množiny, funkcí a predikátů nad nosnou množinou, které jsou přiřazeny k funkčním a predikátovým symbolům.

Pravidlo start, tedy převádí úlohu dokázat spornost M na úlohu: Dojít ke sporu s existencí struktury S takové, že v M by byla nějaká klauzule nesplněná, kde množinu dodatečných předpokladů $Path$ nastaví na prázdnou.

Nesplněnost klauzule v S, resp. nesplněnost všech jejích literálů v S se dokáže pro každý literál samostatně a vždy sporem. Dokazovač vždy zkusí předpokládat, že by byl literál L_1 pravdivý, ale s tímto předpokladem dojde ke sporu. Zbývající část klauzule označíme C a ke sporu dojdeme následovně:

K literálu L_1 najde v M klauzuli C_2, v níž se vyskytuje komplementární literál L_2 po provedení substituce θ. Předpokládali jsme ale, že je M v S splněná, včetně jejích všech klauzulí. Speciálně tedy klauzule C_2 je splněná v S. Tedy i $\theta(C_2)$ je pravdivá v S. Jelikož $\theta(L_1)$ a $\theta(L_2)$ jsou komplementární literály, tak LeanCoPu k důkazu spornosti předpokladu pravdivosti L_1 v S stačí dojít ke sporu s splněností $(C_2 - L_2)\theta$ v S, kde do dodatečných předpokladů $Path$ přidá L_1 (pravidlo extension níže uvedeného kalkulu).

K dokázání tohoto LeanCoP může používat rekurzivně stejný postup a navíc lze používat dodatečných předpokladů ($Path$) jako byl teď L_1 následovně:

Pokud totiž máme dokázat o nějaké klauzuli, že není splněná v S, tak v případě, že obsahuje komplementární literál L_1 k nějakému literálu z dodatečných předpokladů($Path$), tak lze L_1 odebrat a stačí dokázat splněnost zbylé části klauzule v S po provedení odpovídající substituce (pravidlo reduction).

Prázdné klauzule považujeme za nepravdivé (pravidlo axiom).

Pokud by se LeanCoPu povedlo takto dokázat, že všechny literály jsou v nějaké klauzuli nesplněné v S, tak by M nebyla splněná v S. To by však byl spor s tím, že jsme předpokládali, že máme S, která ve které je M splněná. Pokud se to LeanCoPu dovede tak dokázal, že M je sporná a $\neg M$ pravdivá.

Formálněji by bylo možné popsat dokazování tak, jak ho dělá LeanCoP následujícími pravidly. Těm se říká connection kalkulus.

Pravidlo axiom:	$\dfrac{}{\{\}, M, Path}$
Pravidlo start:	$\dfrac{C_2, M, \{\}}{\epsilon, M, \epsilon}$, kde C_2 je kopie[5] $C_1 \in M$
Pravidlo reduction:	$\dfrac{C, M, Path}{C \cup \{L_1\}, M, Path}$, kde $L_2 \in Path$ a $\theta(L_1) = \theta(\overline{L_2})$
Pravidlo extension:	$\dfrac{C_2 - \{L_2\}, M, Path \cup \{L_1\} \quad C, M, Path}{C \cup \{L_1\}, M, Path}$, kde C_2 je kopie $C_1 \in M, L_2 \in C_2$, $\theta(L_1) = \theta(\overline{L_2})$

K pochopení a odůvodnění toho, proč a jak LeanCoP funguje, by bylo potřeba tato pravidla vysvětlit podrobněji a případně dokázat, že fungují. To lze nalézt v publikaci J. Ottena [10]. Nám ale bude stačit vědět, že existují a že je lze použít k dokazování, jak je shrnuto v následující větě.

Věta 1. *Formule M v klauzálním tvaru je sporná právě tehdy, když existuje*

[5]Kopií rozumíme tutéž klauzuli po přejmenování proměnných. To zajistí, že při opakovaném vybrání stejné klauzule, máme v obou klauzulích jiné proměnné.

odvození „$\varepsilon, M, \varepsilon$“[6] v connection kalkulu takové, že všechny listy jsou prázdné.

Implikaci, že pokud existuje odvození, tak je M sporná, lze nahlédnout z neformálního popisu výše, přesto je důkaz této věty nad rámec této práce. Lze ho nalézt v publikacích[3] a [5][7].

Tento teoretický výsledek lze snadno použít k automatickému dokazování vět. Jednou z imlementací tohoto kalkulu je právě LeanCoP.

1.3 LeanCoP

LeanCoP je dokazovač implementující connection kalkulus napsaný v jazyce prolog. Slovo lean v češtině znamená štíhlý. Ideou tohoto dokazovače je totiž dosáhnout maximální efektivnosti za použitím minimálních prostředků. Jak ukážeme s využitím prologu lze connection kalkulus v prologu implementovat na několik málo řádků.

Při dokazování vět pomocí connection kalkulu máme vždy volnost v tom, zda se rozhodneme použít pravidlo reduction nebo pravidlo extension. Dále máme možnosti volby při výběru klauzule v pravidlu start a v pravidlu extension, a také při volbě literálu v obou pravidlech reduction i extension. K tomu, aby se důkaz povedl, je potřeba si nedeterministicky vybrat to správné. K tomu lze použít nedeterminismu prologu, který prohledáváním správnou volbu najde.

Před tím, než si ukážeme jak, se ale ještě jednou podívívejme detailněji na pravidlo extension. Není ani třeba si říkat jeho význam, jen se podívejme syntakticky na to, co dělá. Na vstupu dostane trojici $(Klauzule, M, Path)$. Z $Klauzule$ vybere literál L_1. Zbytek označí jako C. V M najde klauzuli C_1, ve které se vyskytuje negace L_1 označená jako L_2. Do C_2 se přiřadí kopie klauzule C_1 a z ní se vzápětí odebere L_2.

Tento relativně složitý proces lze v prologu snadno řešit použitím dynamického predikátu[8]. Nejen, že si pomocí něj uložíme M tak, že ji není nutné předávat jako parametr rekurze, ale zároveň se při každém použití tohoto literálu provede automaticky přejmenování proměnných, které potřebujeme. Tento predikát budeme nazývat `lit` a uděláme ho binární. Každou klauzuli si do něj totiž neuložíme jen jednou, ale hned tolikrát, kolik má literálů. Literál do jednoho argumentu, zbývající část klauzule bez daného literálu do druhého argumentu. Mechanismus indexace v prologu nám umožní pro zadaný literál rychle najít klauzule, ve kterých se vyskytuje. Z nich bude rovnou odebrané L_2. Druhý parametr, který LeanCoP používá jako výstupní, bude pak vracet přesně $C_2 - \{L_2\}$, jak potřebuje.

Takto je to implementováno v LeanCoPu od verze 2.0. S drobným rozdílem, že tam má predikát `lit` aritu ještě o něco větší z důvodu dalších vylepšení.

[6] Symboly ε nemají žádný speciální význam. Vyskytují se jen zde v této větě a v pravidlu start. Tím je zajištěno, že je toto pravidlo aplikováno jako první a během celého odvození i naposledy.

[7] V některých publikacích to může být popsáno z opačného pohledu tak, že místo spornosti M dokazujeme pravdivost $\neg M$, kde $\neg M$ je reprezentováno jako disjunkce konjunkcí.

[8] Dynamický predikát se v prologu nazývá predikát, jehož definice není součástí vstupního souboru, ale je vytvořena až za běhu programu.

V případech, že by měl být parametr rekurzivní funkce vždy stejný, může být výhodné tento parametr nepředávat každému zanoření do rekurze. To lze v imperativních programovacích jazycích řešit globální proměnnou. V prologu lze k tomuto účelu použít právě dynamický predikát.

Když máme toto hotové, tak lze implementovat dokazovač založený na connection kalkulu na několika málo řádcích kódu v prologu [9]:

```
1  % Pravidlo start
2  start :- natural_number(N), lit(A,B), prove([A|B],[],N).
3  prove([],_,_).
4  prove([L1|C],Path,PathLim) :-
5          (-L2=L1;-L1=L2) ->
6          (
7                  % pravidlo reduction
8                  member(NegL,Path), unify_with_occurs_check(NegL,L2)
9          ;
10                 length(Path, K), K < PathLim,
11                 % pravidlo extension
12                 lit(L2,C2L2,Grnd1),
13                 prove(C2L2,[L1|Path],PathLim)
14         ),
15         prove(C,Path,PathLim).
16
17 natural_number(N) :- N = 1 ; natural_number(N1), N is N1 + 1.
```

Všimněte si, že pokud nebudeme brát ohled na mírné přeznačení proměnných, tak je tento kód v prologu téměř přímým přepisem pravidel connection kalkulu zmíněných výše.

Přeznačení proměnných:

connection kalukulus	kód v prologu
C_2	$[A\|B]$
$C \cup L_1$	$[L1\|C]$
$C_2 - L_2$	$C2L2$

Jediný rozdíl je v tom, že musíme zabránit nekonečné rekurzi a přimět dokazovač, aby nedeterministicky ozkoušel všechna možná odvození. To uděláme tak, že necháme program procházet všechna možná odvození do dané hloubky a tuto hloubku iterativně prohlubujeme.

Dokazovač z ukázky předpokládá, že dynamický predikát `lit` je již vhodně připraven k použití, a spustí se zavoláním procedury start. Ta si nechá generovat přirozená čísla pro všechny možné hloubky prohledávání a pak provádí start krok z connection kalkulu - zavoláním procedury `lit` najde klauzuli z M, se kterou zkusí začít. Poznamenejme, že toto se ve skutečnosti děje v LeanCoPu o něco chytřeji. Ten k výběru počáteční klauzule používá heuristiku a není implementován voláním procedury `lit`, která nám zde může pro stejnou klauzuli uspět vícekrát. Řádek číslo 3 implementuje pravidlo axiom a pouze uspěje. V ostatních případech se na řádku 4 rozdělí první prvek uspořádané trojice ze vstupu kalkulu na část `L1` a `C`. Poté, co počáteční pravidlo vybralo klauzuli, se kterou začít, je zavolána procedura `prove`, která dále vykonává střídavě pravidla extension a reduction. Oba tyto kroky mají dle kalkulu společné to, že na začátku znegují $L1$ na $L2$, to se děje na řádku 5. Poté se příkaz `;` na řádku 9 nedeterministicky rozhodne zda se bude provádět kód, který je specifický pro pravidlo reduction nebo pro pravidlo extension a na závěr se provede `prove(C, Path, Pathlim)`, který je zase společný pro obě pravidla.

Jelikož je connection kalkulus korektní, tak je i LeanCoP korektní. Dále, protože je connection kalkulus úplný a protože iterativní prohlubování pro libovolnou hloubku prohledá všechna odvození do dané hloubky, tak je i LeanCoP úplný.

1.4 Úvod do externích zdrojů axiomů

Oba tyto přístupy, SPASS, LeanCoP a spousta dalších dokazovačů, předpokládaly, že jsou všechny vstupní axiomy na začátku před řešením problému načteny do paměti. Vývoj se dále ubíral směrem k vylepšení dokazovačů tak, aby byly schopné vyřešit co nejsložitější problémy zadané pomocí několika málo axiomů. V posledních letech ale vzrůstá potřeba neřešit tak složité problémy, ale řešit velké jednoduché problémy, které mají na vstupu milióny nebo i více axiomů. Milión axiomů se ještě do paměti vejde, takže načtení takto velkého zadání by běžné dokazovače ještě mohly zvládnout. V dalším kroku by s nimi pak měly uvažovat. Bez ohledu na to, jaký algoritmus nebo kalkulus by dokazovač používal, tak by pravděpodobně zkoušel k sobě dávat všechny dvojice axiomů, aby zjistil, jestli je nemůže použít k odvození dalších faktů. Tím by se běžné dokazovače rychle zahltily, pokud by nebyly na takovouto situaci připraveny. K zahlcení by dokonce mohlo dojít i pokud se k důkazu mělo použít jen několik málo axiomů a pokud by byl důkaz lehký.

Program SPASS-XDB, který rozšiřuje SPASS, se pokusil tuto nevýhodu překonat. V této práci se pokusíme o to samé rozšířením LeanCoPu a zároveň se pokusíme navrhnout program tak, aby byl v některých ohledech efektivnější.

SPASS-XDB používá ke komunikaci s externími zdroji nebo s mediátory protokol tptp_qa. Ten, jak ještě v této kapitole více přiblížíme, umožňuje dokazovači říkat si o externí axiomy formou dotazů.

V rámci této práce se programem LeanCoP-XDB pokusíme upravit dokazovač LeanCoP tak, aby také umožňoval pracovat s externími zdroji axiomů. K tomu použijeme stejný protokol pro komunikaci s externími zdroji. Díky tomu budeme moci použít stávající externí zdroje a mediátory k nim. Zároveň to ale umožní, aby oba dokazovače, jak LeanCoP-XDB tak SPASSu-XDB, mohly řešit úlohy se stejnými externími zdroji a mediátory vzniklými v rámci této práce, bez nutnosti je pro každý dokazovač zvlášť upravovat. Při použití stejného komunikačního protokolu tedy bude možné oba dva dokazovače porovnat a dát jim stejné vstupy.

Podrobný popis protokolu tptp_qa je v příloze C. Stručně ho ale popíšeme už zde. Dokazovače dostávají na vstupu soubory, ve kterých jsou ve formulích predikátové logiky prvního řádu popsané axiomy a domněnka, o které chceme dokázat, že z nich vyplývá. Symboly predikátové logiky jsou v souboru zadání zakódované dle syntaxe jazyka TPTP.

To, že má dokazovač k dispozici externí zdroje axiomů, je zapsáno také v těchto vstupních souborech. Syntaxe, ve které se zapisují axiomy je pro tento účel mírně upravená, aby umožňovala říct, jak a kam se má s externím zdrojem komunikovat. Tento protokol zatím podporuje jen komunikaci s lokálně běžícími programy přes jejich standardní vstup a výstup. Kromě specifikace procesu, který se má pro komunikaci spustit se také specifikuje jaké axiomy umí externí zdroj poskytnout.

V protokolu tptp_qa to zpravidla jsou ground atomy. Součástí specifikace mimo jiné je:

1. Predikátový symbol.

2. Arita predikátového symbolu.

3. Seznam argumentů, jejichž hodnotu dokazovač musí povinně zadat v dotazu o odpovídající axiomy. Takovéto argumenty budeme ve zbytku textu nazývat univerzálně kvantifikované.

4. Seznam argumentů, jejichž hodnotu dokazovač nemusí v dotazu uvést. V takovémto případě budou odpovědí všechny axiomy, které mohou mít na daném místě libovolnou hodnotu. Těmto argumentům budeme ve zbytku textu říkat existenčně kvantifikované.

Pro zapsání těchto čtyř informací budeme ve zbytku textu používat stejnou syntaxi, jaká je v protokolu tptp_qa. Argumenty pojmenujeme tím, že za ně dosadíme proměnné. Budeme jim říkat odpovídajícím způsobem - existenčně, resp. univerzálně kvantifikované. Vše zapíšeme ve tvaru:
`!` *List1* `:` `?` *List2* `:` *predikat*`(`*argumenty*`)`, kde

- *Predikat* je název predikátového symbolu.
- *List1* je seznam univerzálně kvantifikovaných proměnných.
- *List2* je seznam existenčně kvantifikovaných proměnných.
- *Argumenty* je zápis všech argumentů pomocí názvů proměnných oddělených čárkou.

V případě, že je *List1*, resp. *List2* prázdný, tak `!` *List*1 `:`, resp. `?` *List*2 `:` vynecháváme.

Těmto zápisům a rozdělením argumentů na existenčně a univerzálně kvantifikované budeme říkat kvantifikace externího zdroje. Jeden zdroj může být kvantifikován více způsoby.

Předpokládejme například, že máme binární predikát *kraj_okres*, který říká, který okres je ve kterém kraji. V případě, že tyto axiomy umístíme všechny do externího zdroje a v případě, že je chceme dát všechny dokazovači k dispozici, tak externí zdroj kvantifikujeme následovně:

```
? [Kraj,Okres] : kraj_okres(Kraj, Okres)
```

To umožní dokazovači s externím zdrojem pracovat následovně:

1. Pro zadaný kraj a okres si nechat od zdroje ověřit, zda daný kraj náleží do daného okresu.
2. Pro zadaný kraj zjistit okresy, které se v něm nachází.
3. Pro zadaný okres zjistit kraj(e), ve kterých se nachází.
4. Nezadat nic a nechat si od zdroje vrátit všechny dvojice $(Kraj, Okres)$.

Pokud bychom chtěli dokazovači třeba zakázat, aby pokládal tuto poslední nejobecnější otázku (např. z důvodu, že by ji neuměl externí zdroj zpracovat), tak je třeba zdroj kvantifikovat následovně dvojím způsobem:

```
! [Kraj] : ? [Okres] : kraj_okres(Kraj, Okres)
! [Okres] : ? [Kraj] : kraj_okres(Kraj, Okres)
```

Tyto dva řádky lze číst jako:

1. Pro zadaný kraj lze ve zdroji hledat okresy.

2. Pro zadaný okres lze ve zdroji hledat kraje.

Dále takto specifikovaný zdroj umí i pro zadaný okres a kraj potvrdit zda se okres v daném kraji nachází. O takovémto dotazu budeme říkat, že je specifičtější nebo konkrétnější než předchozí dotazy. Ty budeme zase nazývat obecnější než tento dotaz.

Experimenty ve 4. kapitole ukáží, že SPASS-XDB někdy posílá dotazy, které nikdy nemohou pomoci k dokázání věty. Jindy zase posílá dotazy, které jsou zbytečně obecné.

Pokusíme se LeanCoP-XDB navrhnout tak, aby tyto nevýhody neměl, tzn. aby odesílal jen dotazy, které by hypoteticky mohly vést k dokázání věty a zároveň se pokusíme, aby nebyly příliš obecné. To zamezí tomu, že by v odpovědi přišlo velké množství axiomů. Tím se zároveň přiblížíme k vytyčenému cíli - i velké množství axiomů v externím zdroji dokazovač nezpomalí. Tohoto cíle se budeme snažit dosáhnout i za cenu, že by LeanCoP-XDB odeslal více konkrétnějších dotazů, což považujeme za lepší, než menší množství obecnějších dotazů.

2. LeanCoP-XDB

Hlavním cílem této práce je upravit LeanCoP tak, aby byl také schopný používat externí zdroje axiomů. Takto upravenému LeanCoPu říkáme LeanCoP-XDB a v této kapitole popíšeme, jak funguje. Nejprve se v sekci 2.1 zamyslíme nad možnostmi, jak LeanCoP upravit a představíme myšlenku schématických důkazů, která odděluje uvažování dokazovače od komunikace s externími zdroji axiomů.

Poté se v sekci 2.2 zamyslíme nad tím, co lze dále dělat se schématickým důkazem a jak může dále dokazovač postupovat při jeho přeměně na důkaz. V této sekci jsou také vyjmenovány faktory, které mohou ovlivňovat efektivitu různých postupů. Přestože bylo již dříve prozrazeno, že z důvodu vzájemného porovnání bude nutné použít protokol tptp_qa, tak jeho použití není úplně samozřejmé. Pokud bychom mohli začít od začátku a pokud by nebylo nutné použít už existující externí zdroje a mediátory, tak by bylo možné buď navrhnout nový komunikační protokol, a nebo stávající protokol tptp_qa rozšířit a mít v něm další dodatečné informace, které by byly pro LeanCoP-XDB výhodné. Tyto možnosti, přestože nebyly implementovány a testovány, rozebereme také v této sekci, 2.2.

Otázky, které je potřeba zodpovědět před konkrétním návrhem a rozhodnutí, které bude potřeba učinit, budou vyjmenována v sekci 2.3. Jak uvidíme v sekci 2.4, byla struktura aplikace navržena tak, aby nebyla závislá na zvolených možnostech. Ty při tomto návrhu ovlivní jen způsob implementace jednotlivých komponent. Jaké volby z možností vyjmenovaných v sekci 2.3 byly nakonec vybrány a implementovány popisuje sekce 2.5.

2.1 Schématické důkazy

Jak ale LeanCoP upravit, aby mohl externí zdroje axiomů použít? Jednoduchým a naivním řešením by bylo upravit proceduru `lit` tak, aby hledala axiomy nejen v interní databázi prologu, ale aby se pokoušela získat axiomy přes mediátor z externího zdroje axiomů, pokud by nějaký z nich podle své specifikace mohl takovéto axiomy vracet. Toto řešení by bylo funkční. My ale předpokládáme, že získání axiomů z externího zdroje může trvat mnohonásobně déle, než jejich získání z interní databáze prologu. Proto by bylo vhodné přimět program, aby počítal a využil čas čekání mezi tím, než přijde odpověď. Dále stojí za povšimnutí, že takto bychom dotazy na existenci axiomů do externích zdrojů odesílali i v případě, že by nebylo možné najít důkaz s jakoukoli odpovědí na daný dotaz, která by mohla přijít. Dotazy, o kterých by se dalo dopředu říct, že nejsou k důkazu potřeba, vůbec nemusíme odesílat. To je jedna z věcí, kterou SPASS-XDB neřešil a v čem má být LeanCoP-XDB lepší.

Ale jak na to? Externí axiomy jsou ground atomy. A právě toho, že jsou to atomy, využijeme. Procedura `lit` dostane v prvním (vstupním) parametru literál. Nejprve `lit` vrátí klasicky interní axiomy a dále musí vhodně zareagovat, pokud by axiom mohl být externí.

Jak jsme si již řekli při představení naivního řešení, je jedna ze správných reakcí čekání na odpověď. Ta může navázat nějaké termy do proměnných v literálu. Mezitím ale, než odpověď přijde, chceme využít čas procesoru k něčemu užitečnému. V tuto chvíli budeme předpokládat, že se vhodný axiom podaří najít a

budeme pokračovat v dokazování. To, zda byl tento předpoklad správný, se ověří až později - poté, co přijdou odpovědi na daný dotaz. Není ale nutné provádět ověření ihned. Ve skutečnosti lze u všech míst, kde by bylo potřeba použít externí axiom předpokládat, že tento axiom se doplní později, a zkoušet hledat zbývající část důkazu. Jen je nutné si všechny tyto předpoklady pamatovat a dodatečně je ověřit. Říkejme tomuto seznamu předpokladů *schématický důkaz*. Pokud se nám pomocí odpovědí z externích zdrojů nepodaří vyrobit ze schématické odpovědi důkaz, tak se spustí backtracking a hledá se další nový schématický důkaz.

2.2 Doplňování schématických odpovědí na důkaz

Nyní nám zbývá vyřešit problém, jak doplnit schématickou odpověď na důkaz. Vstupem je seznam atomů, které mohou obsahovat volné proměnné. Pro ně lze poslat dotazy do externích zdrojů axiomů a hledat axiomy, které daným atomům odpovídají. Výstupem je schématická odpověď, která má do proměnných, které byly volné, navázané hodnoty tak, aby jednotlivé atomy odpovídaly datům z externích zdrojů. Vzhledem k tomu, že se volné proměnné v původní schématické odpovědi mohou vyskytovat ve více atomech, je potřeba, aby nalezené odpovídající axiomy z externích zdrojů byly mezi sebou navzájem konzistentní v tom smyslu, že všechny atomy budou mít do dané proměnné navázánu stejnou hodnotu.

Dejme tomu, že máme doplňovat například schématickou odpověď z ukázky 2.1.

```
populace('Sobota', Obec, Pocet)
okres_obec(Okres, Obec)
kraj_okres('Karlovarsky kraj', Okres)
regexp(Obec, 'Z|ov')
$greatereq(Pocet, 5)
$lesseq(Pocet, 10)
```

Obrázek 2.1: Ukázka schématické odpovědi

Takováto schématická odpověď říká, že hledáme v Karlovarském kraji obec, která má v názvu „Z“ nebo „ov“, a ve které bydlí 5 až 10 lidí s příjmením Sobota.

Funkční doplňovač schématických důkazů najde vhodnou substituci proměnných tak, aby každému atomu odpovídal nějaký axiom. Například jako v ukázce 2.2.

Stručně přeloženo: Příjmení Sobota bylo celkem 8x nalezeno v obci Sokolov. Ta se nachází v okresu Sokolov a ten je v Karlovarském kraji. O názvu Sokolov platí, že obsahuje „Z“ nebo „ov“. 8 je v rozsahu 5 až 10.

Problém doplňování schématických důkazů může v určitém smyslu připomínat provádění databázových operací JOIN, kde je cílem najít libovolný záznam, který by byl v výsledné tabulce. Předpokládejme, že bychom měli vše v jedné databázi, která by mohla obsahovat i nekonečné tabulky. Například pro $lesseq bychom měli tabulku, která by obsahovala všechny dvojice čísel takové, že první je menší nebo

```
populace('Sobota', 'Sokolov', 8)
okres_obec('Okres_Sokolov', 'Sokolov')
kraj_okres('Karlovarsky_kraj', 'Okres_Sokolov')
regexp('Sokolov', 'Z|ov')
$greatereq(8, 5)
$lesseq(8, 10)
```

Obrázek 2.2: Ukázka vyplněné schématické odpovědi

rovné druhému nebo například pro tabulku regexp bychom měli zaznamenané všechny řetězce a všechny regulární výrazy, které jim odpovídají. Pak by bylo tento problém, hledání vhodného doplnění schématické odpovědi, možné popsat následujícím SQL dotazem:

```
SELECT okres_obec.okres, okres_obec.obec, populace.pocet
FROM populace
JOIN okres_obec
ON okres_obec.obec = populace.obec
JOIN kraj_okres
ON kraj_okres.okres = okres_obec.okres
JOIN regexp
ON regexp.obec = populace.obec
JOIN greatereq
ON greaterq.a = pocet
JOIN lesseq
ON lesseq.a = pocet
WHERE populace.prijmeni = 'Sobota' AND
kraj_okres.kraj = 'Karlovarsky_kraj' AND
regexp.pattern = 'Z|ov' AND
greatereq.b = 5 AND
lesseq.b = 10
```

Vzájemně propojené prologovské proměnné musejí být při převodu do SQL nahrazeny podmínkou určující, přes který sloupec se mají tabulky spojovat. Instanciované hodnoty v atomech schématického důkazu byly přesunuty do SQL klauzule WHERE. Odpovědí na takovýto SQL dotaz by pak nebylo jen jediné doplnění schématického důkazu, ale seznam všech odpovídajích substitucí.

Náš problém se však od provádění databázových spojení liší v následujících ohledech:

- Jednotlivé tabulky můžou být nekonečné.
- Jednotlivé tabulky se mohou nacházet v různých zdrojích, které jsou i v geograficky rozdílných lokalitách. Nikde není uchovávána struktura databáze, což omezuje použití indexů a dalších chytrých datových struktur pro urychlení prováděného spojení.
- Rozšířená syntaxe TPTP o možnost specifikovat externí zdroj podporuje jak existenční, tak univerzální kvantifikaci. Je tedy možné specifikovat zdroj tak, že v dotazech musí být část atomu už instanciována. To znemožňuje možnost enumerovat všechny axiomy i u zdrojů, které jsou konečné.

V případě, že by byly všechny zdroje existenčně kvantifikované, tak bychom měli úplnou volnost v tom, v jakém pořadí by JOINy byly, jak by byly uzávorkované, v jakém pořadí by se dotaz vyhodnocoval a spojení, která by spolu nesouvisela, by mohla být paralelizována. Pokud ale povolíme univerzální kvantifikaci proměnných pro externí zdroj, tak musíme být opatrnější.

Pro jednoduchost ale nejprve předpokládejme, že jsou všechny zdroje existenčně kvantifikované. Vzhledem k oddělenosti zdrojů a neexistenci pomocných datových struktur je jedinou možností, jak provádět spojení, enumerace. Máme dvě možnosti, jak pomocí enumerování hodnot tabulky spojovat:

1. **Párování každého záznamu s každým -** Tento způsob vyžaduje, abychom si záznamy z tabulek ukládali. Alespoň jednu tabulku je třeba mít načtenou v paměti. Není ale nezbytně nutné, aby tam byla celá před začátkem provádění tohoto spojení. Je ale nutné pro každý záznam z jedné tabulky kontrolovat existenci vhodného záznamu v druhé tabulce a v případě shody dát spojení záznamů do výsledné tabulky. Tuto činnost je možné provádět současně s načítáním vstupních tabulek.

2. **Pro každý záznam z jedné tabulky posílat dotaz do druhého zdroje** - První tabulka může být jak externí zdroj, tak spojení jiných tabulek. Druhá tabulka musí být pro použití této metody externím zdrojem. Není tedy možné, aby druhá tabulka byl výsledek nějakého předchozího spojení. Tato metoda enumeruje všechny záznamy z první tabulky a pro každý z nich pošle dotaz do externího zdroje axiomů, který zastupuje druhou tabulku. Tento dotaz obsahuje už i hodnoty proměnných, přes které provádíme spojení, a tedy v odpovědi přijdou jen ty axiomy, které by byly v daném spojení.

Metoda *párování každého záznamu s každým* je vhodná především pro malé tabulky. Jejich načtení z externích zdrojů zabere krátký čas a množství záznamů v nich je dost malé, aby bylo únosné je zkoušet spárovat každý s každým. To může být někdy výhodnější, než *pro každý záznam z jedné tabulky posílat dotaz do druhého zdroje*, vzhledem k tomu, že by se mohlo ztratit velké množství času čekáním na odpovědi.

Především nedává smysl posílat velké množství dotazů do zdroje, který by mohl vracet jen malé množství axiomů. V takovém případě je lepší si je všechny stáhnout.

Naopak v případě, že očekáváme, že by bylo potřeba stahovat velké množství axiomů, tak je lepší odeslat více dotazů a získat jen axiomy ze spojení, stačí *pro každý záznam z jedné tabulky posílat dotaz do druhého zdroje*.

Možnost mít univerzálně kvantifikované zdroje situaci ještě o trochu komplikuje. U takovýchto zdrojů není možné k JOINu použít *párování každého záznamu s každým*. Vždy je tedy potřeba mít už připravenou tabulku, která se má s tímto zdrojem spojit a *pro každý záznam z dané tabulky posílat dotaz do druhého zdroje*.

Při splnění této jediné podmínky má zbývající volba pořadí a způsobů JOINů vliv jen na efektivitu. Korektnost a úplnost zůstává zachována vždy.

Doposud jsme předpokládali, že každý externí zdroj axiomů má na starosti jedinou tabulku. Tak tomu ale nemusí vždy být. Jediný program může odpovídat na dotazy pro více tabulek. V takovém případě by bylo vhodné, aby se spojení provedlo už na straně zdroje a k nám aby se přenesla už jen výsledná tabulka.

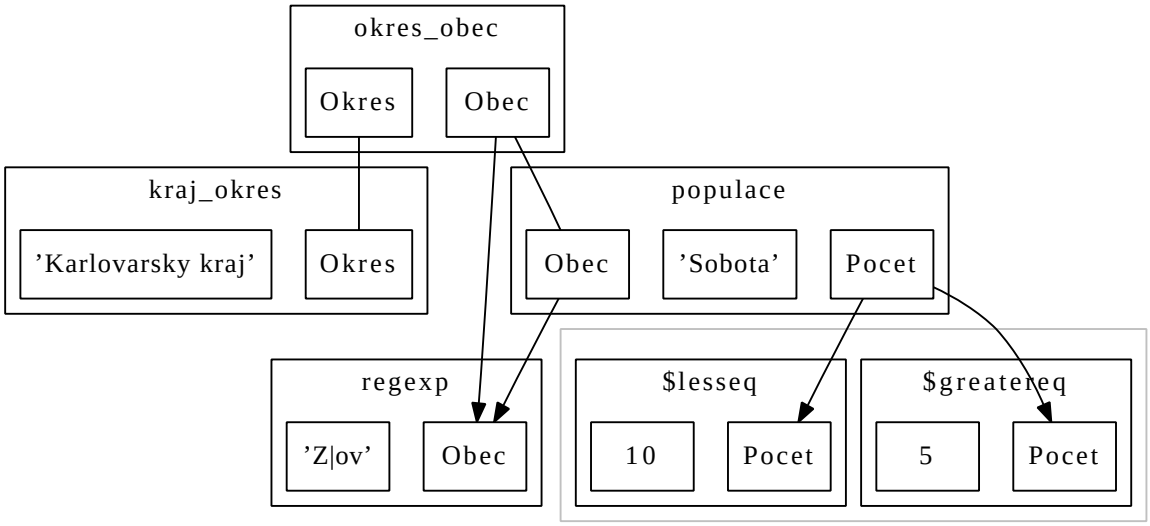
Tuto funkcionalitu podporuje externí zdroj *Mathematica*. Ta pro tento účel komunikuje pomocí mírně upraveného protokolu tptp_qa. Do budoucna se předpokládá, že by toto mohlo být podporováno dalšími zdroji.

Předpokládejme, že máme externí zdroje kvantifikované dle 2.3. Situace v na-

```
! [Prijmeni] : ? [Obec,Pocet] : populace(Prijmeni,Obec,Pocet)
? [Okres,Obec] : okres_obec(Okres, Obec)
? [Kraj, Okres] : kraj_okres(Kraj, Okres)
! [String,Pattern] : regexp(String,Pattern)
! [A,B] : greatereq(A,B)
! [A,B] : lesseq(A,B)
```

Obrázek 2.3: Ukázka kvantifikace externích zdrojů

šem příkladu je znázorněna na obrázku 2.4.



Obrázek 2.4: Příklad schématické odpovědi

Rámečky značí jednotlivé externí zdroje axiomů. Šedivý rámeček spojující $lesseq a $greatereq značí, že tyto tabulky se ve skutečnosti vyskytují v jediném externím zdroji. Hrany značí provázání proměnných. Případy, kdy má některý zdroj některou proměnnou univerzálně kvantifikovanou, jsou značeny orientovanou hranou ve směru toku dat.

Z obrázku je vidět, že dotaz do zdroje populace musí být proveden dříve než dotazy do softwaru Mathematica. Je tomu tak proto, že zdroj populace, bude vracet v axiomech hodnoty proměnné `Obec` a počty jejich obyvatelů s příjmením Sobota. A až poté se tyto počty pomocí zdroje Mathematica zkontrolují, zda jsou v rozsahu 5 až 10.

Dále je potřeba, aby byla proměnná `Obec` instanciována před tím, než přijde do zdroje *regexp*. Tato hodnota může být získána buď ze zdroje *okres_obec* nebo ze zdroje *populace*, a nebo z tabulky vzniklé jejich spojením.

Pořadí dotazů do ostatních zdrojů může být libovolné a tam kde to jde, může být i paralelizováno.

Uveďme si jedno z možných pořadí vyhodnocování jako příklad:

Současně se spustí dotazy do tabulek *okres_obec*, *kraj_okres* a spojují se přes proměnnou `Obec` pomocí párování každého s každým, a dále se obce z výsledku tohoto spojení ověřují ve zdroji *regexp*. Ve druhé paralelní větvi můžou být přijímány odpovědi na dotaz do tabulky populace a dále pak honoty proměnné `Pocet` ověřovány *Mathematicou*. Obě tyto paralelní větve mohou být pak spojovány metodou *každý záznam s každým*. V případě, že budou nalezeny dva záznamy z obou paralelních větví, které se podaří spojit, tak prohledávání končí úspěchem a nalezením vhodného doplnění schématického důkazu. V opačném případě, po ozkoušení všech dvojic z výsledných tabulek z obou paralelních větví, skončí hledání neúspěchem a bylo ověřeno, že tuto schématickou odpověď není možné doplnit na důkaz.

Volba vhodného pořadí vyhodnocování má velký vliv jak na čas strávený strávený doplňováním schématické odpovědi, tak na počet odeslaných dotazů a na počet přijatých axiomů.

Které pořadí vyhodnocování je pro danou schématickou odpověď nejvýhodnější závisí na následujících faktorech:

- Počet jader procesoru, které máme k dispozici. Ten výrazně ovlivňuje, jakou míru paralelizace je vhodné zvolit. S menším počtem jader by mohlo být větší množství paralelních výpočtů, které se budou na jádrech střídat, pomalejší, než odeslání několika málo dotazů do externích zdrojů navíc a počkání na odpověď.

- Doba odezvy zdroje - tedy doba za jak dlouho zdroj po odeslání otázky zareaguje a za jak dlouho začnou přicházet odpovědi.

- Přenosová rychlost - tedy jak rychle budou přicházet odpovědi.

  Ta by se mohla měřit buď v násobcích bajtů za jednotku času nebo třeba v počtu přijatých odpovědí za jednotku času.

- Možnosti paralelizace na straně externích zdrojů axiomů. - Jak se zdroj zachová v případě, že mu bude doplňovač schématických odpovědí posílat neúnosně velké množství otázek? Bude je zdroj axiomů zpracovávat jednu po druhé? A začnou tedy odpovědi na druhou otázku přicházet až po všech odpovědích na první otázku? To je důležitá charakteristika a bylo by vhodné vše naplánovat tak, aby se nemuselo čekat. V případě, že už externí zdroj zvládá odpovídat na více otázek současně, tak jak se to bude dít přesně? Budou se odpovědi na různé dotazy vzájemně brzdit a bude tedy počet přijatých odpovědí za vteřinu od zdroje konstantní? Nebo bude počet přijatých odpovědí od zdroje za vteřinu přímo úměrný počtu aktuálně zpracovávaných dotazů? Pozor, tyto úvahy jsou stále ještě zjednodušené. Protokol tptp_qa neumožňuje mezi sebou míchat odpovědi na otázky. Umožňuje jen sekvenční odpovídání na jednu otázku po druhé. Proto při tomto způsobu paralelizace na straně zdroje by bylo nutné mít více otevřených spojení do zdroje současně. Pak by bylo potřeba si odpovědět ještě na následující otázky:

  - Kolik je možné mít do jednoho zdroje nejvíce otevřených spojení? A v případě, že neomezeně, tak jaká plynou omezení z velkého počtu

současně otevřených spojení?

  - Jak dlouho trvá navázání nového spojení?
  - Jakou zvolit taktiku pro otvírání nových spojení? Otevřít si několik spojení předem a v případě jejich vytíženosti otevírat další? Zvolit si nějaký konstantní počet spojení pro konkrétní schématickou odpověď a pak dotazy dělit do těchto otevřených spojení?
  - Jakou zvolit taktiku pro zavírání nepotřebných spojení? Má se zavírat spojení, pokud pro něj ve frontě není žádný dotaz k odeslání nebo se má spojení nechat otevřené k případnému použití až přijde další dotaz?

- Kolik bude který zdroj při daném uspořádání vracet axiomů jako odpověď na pokládané otázky?

  Tuto otázku, zřejmě s největším vlivem na efektivitu daného uspořádání, není snadné zodpověďět bez znalosti toho, jak externí zdroj axiomů zareaguje. Částečnou představu ale lze získat z velikosti celé tabulky a dále ze znalostí velikostí domén hodnot z jednotlivých sloupců. Díky této informaci se totiž dozvíme očekávaný počet odpovědí na dotaz se zadanou hodnotou nějakého sloupce. Takto by ale bylo možné pokračovat. Mohli bychom předpokládat, že budou ve specifikaci zdroje zadány i hodnoty, ze kterých půjde spočítat i očekávané množství odpovědí, pokud bude v otázce zadána hodnota více sloupců. Ani tím by ale výčet nekončil. Bylo by dále zajímavé, znát i jak moc blízké budou skutečné počty od této teoretické hodnoty - tedy veličina podobná rozptylu.

Jazyk pro specifikaci externích zdrojů axiomů umožňuje specifikování volitelných informací, kterých mohou dokazovače buď využít, nebo je ignorovat. Právě zde by byl prostor pro uvedení většiny z těchto informací.

Údaje o zdroji, které by nebyly zadány v jeho specifikaci by často ani nemusely být zadány vůbec. Většinu z nich si totiž může dokazovač během komunikace se zdrojem změřit a poté při dalších komunikacích s ním využít.

Bohužel ani při znalosti všech těchto údajů stále nelze triviálně říct, které pořadí je nejlepší.

Teoretická analýza toho, jak z těchto údajů určit ideální pořadí vyhodnocování, a ani heruistiky, jak zvolit alespoň nějaké dobré uspořádání, nejsou součástí této práce.

Dále by bylo možné na doplnění jednoho schématického důkazu začít používat jedno pořadí vyhodnocování a kdyby se ukázalo jako nevýhodné, tak ho za běhu změnit a začít používat jiné pořadí.

Pro jednoduchost ale všechny tyto údaje v implementaci LeanCoPu-XDB zanedbáváme a volíme pořadí vyhodnocování jen tak, aby bylo korektní. Efektivitou se v tomto případě nezabýváme.

2.3 Možnosti jak LeanCoP-XDB implementovat

Při tvorbě LeanCoPu-XDB muselo být učiněno několik rozhodnutí. Otázky, které je třeba si před implementací zodpovědět zde budou nejprve vyjmenovány, následně na všechny odpovíme v sekci 2.5.

1. Jak bude synchronizováno jádro LeanCoPu s doplňovačem schématických odpovědí? Bude LeanCoP generovat nové schématické odpovědi mezi tím, co bude doplňovač schématických odpovědí dělat svou práci? V případě, že ano, bude nějak omezena velikost fronty pro vygenerované schématické odpovědi - po kolika schématických odpovědích se má LeanCoP zablokovat?

2. Jaké pořadí doplňování schématických odpovědí bude zvoleno (včetně způsobu paralelizace)?

3. Jak bude naloženo s dalšími přicházejícími schématickými odpověďmi? Bude se alespoň částečně využívat možnosti paralelizace vzájemně mezi různými schématickými odpověďmi. Budou nevyužité externí zdroje během doplňování jedné schématické odpovědi vytěžovány dotazy z následující schématické odpovědi?

4. Kolik bude udržováno spojení do externích zdrojů axiomů?

5. Jak bude řešena potřeba odeslat více otázek do jednoho zdroje před jejich zodpovězením? Budou se před odesláním ukládat ve frontě?

6. Pokud schématická odpověď povede k odeslání již dříve poslané otázky, budou nějak využity odpovědi, které již dříve dorazily?

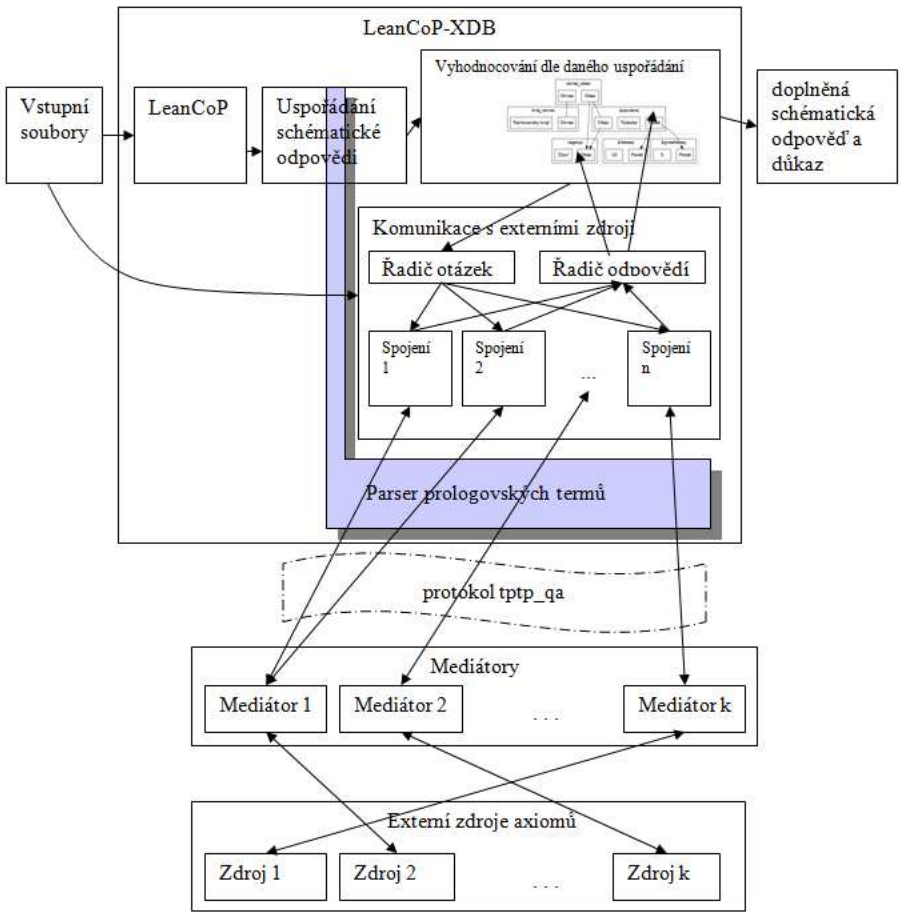
   Budou nějak využity odpovědi na dříve odeslanou specifičtější otázku, pokud se doplňovač schématických odpovědí bude snažit dotázat na obecnější dotaz?

7. Jak budou reprezentované termy, aby s nimi bylo možné provádět JOINy tabulek a aby tato reprezentace umožňovala paralelizaci?

Přestože všechny tyto otázky musely být zodpovězeny a přestože ze všech způsobů řešení, které se nabízely, muselo být do ukázkové implementace vybráno jedno konkrétní, byl v průběhu celého návrhu aplikace brán ohled na případné změny. Tomu odpovídá i celková struktura aplikace, která je na konkrétních rozhodnutích nezávislá. Případné rozhodnutí se pro jinou alternativní možnost by tedy znamenalo jen minimální změnu v programu nebo nejvýše jen přepsání odpovídající komponenty.

2.4 Popis komponent LeanCoPu-XDB

Fungování LeanCoPu-XDB je schématicky znázorněno na obrázku 2.5. Vstupní soubory jsou přečteny dvakrát - jednou LeanCoPem, tedy jádrem dokazovače, který je napsaný v prologu, podruhé komponentou pro komunikaci s externími



Obrázek 2.5: LeanCoP-XDB schématicky

zdroji axiomů. Ta si ze vstupních souborů přečte důležité informace o tom, s jakými se lze spojit zdroji a jak bude probíhat komunikace. LeanCoP hledá důkaz nebo generuje schématické odpovědi. Ty jsou následně přeuspořádány do vhodnějšího pořadí. Toto přeuspořádání může probíhat (a probíhá) částečně v prologu, jehož výstup je pak naparsován aplikací programem v Javě. Ten pak provádí ještě další douspořádávání pořadí schématických odpovědí a vytvoří uspořádání atomů schématické odpovědi, v jakém se bude vyhodnocovat. To je přijato komponentou, která samotné vyhodnocování dle zadaného pořadí vykonává, a v případě úspěchu vrací doplněnou schématickou odpověď a důkaz. K tomu ale potřebuje komunikovat s externími zdroji axiomů. Kontaktuje je pomocí rozhraní, kterému můžeme říkat třeba řadič otázek. Ten ze tvaru otázky pozná, který externí zdroj umí na takovou otázku odpovídat a vybere vhodné spojení, kterým otázku odešle. Zároveň řadič otázek vrátí komponentě pro vyhodnocování daného uspořádání objekt, který je propojen s řadičem odpovědí a kterým bude možné číst odpovědi poté, co přijdou. Dále tento objekt umožňuje provést blokující čtení - zablokovat aktuální vlákno, dokud nebude dostupná odpověď.

To, zda této možnosti bude využito a případně kdy, závisí už na konkrétní implementaci komponenty pro vyhodnocování daného uspořádání.

2.5 Konkrétní implementace LeanCoPu-XDB

Tato sekce popisuje zvolená rozhodnutí přijatá během návrhu. Ta jsou číslována a jednotlivá čísla odpovídají problémům k rozhodnutí z předminulé sekce. Konkrétní volby jsou často kompromisem mezi složitostí celkové ukázkové implementace a ideálním případem.

1. Po zapnutí programu LeanCoP-XDB, který je napsaný v Javě, je spuštěn upravený LeanCoP napsaný v prologu. Ten si sám načte vstupní soubory a je s ním komunikováno přes jeho standardní výstup. Standardní vstup není použit, což zaručuje, že LeanCoP nikdy nezablokuje[1] LeanCoP-XDB z důvodu nečtení vstupu. Z výstupu LeanCoPu jsou čteny schématické odpovědi, které jsou vyhodnocovány v pořadí v jakém přicházejí.

   V implementaci je použita třída InputStream ze standardních knihoven Javy. Ta obsahuje pro čtení výstupu z prologu frontu o velikosti několika desítek kilobytů. Do nich se vejde zhruba několik desítek až stovek schématických odpovědí v závislosti na složitosti řešeného problému a složitosti schématických odpovědí. Důsledkem toho je, že mezitím co LeanCoP-XDB zkouší doplnit jednu schématickou odpověď, tak LeanCoP generuje další schématické odpovědi. Zároveň je tím ale zamezeno, aby jich vygeneroval příliš veliké množství. Po zaplnění této fronty se totiž proces LeanCoPu zablokuje, dokud se ve frontě neuvolní nové místo.

   S možností měnit velikost této fronty a tedy i měnit, po jak dlouhé době je vhodné LeanCoP zablokovat nebylo experimentováno.

2. Vzhledem k neprozkoumanosti situace ohledně toho, které pořadí vyhodnocování je výhodnější, se vždy zvolí první korektní pořadí takové, že se všechny atomy doplňují lineárně jeden po druhém. Toto pořadí je nalezeno hladově: Jako první bude v tomto pořadí první atom, který ke svému vyhodnocení nepotřebuje mít vyplněné hodnoty žádných proměnných. Tento atom je pak odebrán ze seznamu atomů k utřídění. Jako druhý bude vybrán první atom z těch, které zbývají a který nepotřebuje ke svému vyhodnocení znát buď hodnoty žádných proměnných nebo potřebuje znát hodnoty jen již vyplněných proměnných.

3. Vždy je zpracovávána jen jediná schématická odpověď, a až v případě zjištění, že ji nelze použít, se začne zpracovávat nová. Rovněž nebyla testována možnost nechat doplňovat více schématických důkazů paralelně.

4. Do každého externího zdroje je otevřeno nejvýše jedno spojení. Toto spojení je navázáno ve chvíli, kdy řadič otázek má zájem do tohoto zdroje poslat první otázku.

   Optionem `-g` lze změnit nastavení programu tak, aby do každého zdroje otevíral právě jedno spojení a to už před tím než by se případně mělo použít.

   S možností otevírat do jednoho zdroje více spojení nebylo experimentováno.

[1] Zde předpokládáme znalost Unixové komunikace mezi procesy pomocí rour. Ty obsahují malou frontu a mají jednoduchou synchronizaci, která zapisující proces zablokuje, pokud je roura plná a naopak čtecí proces zablokuje, pokud je prázdná.

5. Jelikož se vždy doplňuje nejvýše jedna schématická odpověď, tak počet právě zpracovávaných otázek jedním zdrojem je maximálně počet atomů ve schématické odpovědi, což je množství, které je obvykle v řádu jednotek. Proto pro hromadění otázek na straně LeanCoPu-XDB není použitá žádná fronta a všechny otázky se v okamžiku jejich vzniku rovnou zapisují do streamu, kterým se komunikuje s daným zdrojem nebo s mediátorem do něj. V exterémním případě by se mohlo stát, že by mediátor nemusel stíhat zpracovávat otázky a už by se nevešly do fronty daného streamu a to by mohlo zablokovat hlavní vlákno dokazovače. To se ale v žádném z testů nestalo a vzhledem k malému množství současně zpracovávaných otázek jedním zdrojem to není pravděpodobné. V případě, že by se to přece jen stalo, probíhá čtení odpovědí a jejich ukládání do fronty před řadičem odpovědí ve zvláštním vlákně, takže by tato situace způsobila jen zpomalení programu, nikoliv deadlock.

6. Struktura programu teoreticky umožňuje pamatovat si odeslané otázky a jaké k nim dorazily odpovědi. V době návrhu programu se předpokládalo, že by to ale nepřineslo výrazné zlepšení. Experimenty ve 4. kapitole ale ukáží, že by to za některých okolností přineslo velmi výrazné zrychlení. Proto by tato funkcionalita měla být do programu v budoucnu dodělána.

7. Při běhu celého programu je potřeba provádět operace s termy a především jejich unifikace. Ty mohou mezi sebou propojovat proměnné a všechny tyto operace musí být proveditelné i v opačném pořadí. Termy, které se jednou zunifikují musí být možné opět odunifikovat. K tomu lze použít běžné datové struktury k tomuto účelu určené. Každá prologovská proměnná si pamatuje buď ukazatel na svou hodnotu, a nebo v případě že není instanciovaná, ale přesto je propojená s jinou prologovskou proměnnou, tak je tato situace řešena tak, že si mladší proměnná pamatuje ukazatel na starší. Všechny prováděné operace (unifikace a pod.) se zaznamenávají na zásobník a v případě, že se spouští backtracking, tak se dle informací na tomto zásobníku proměnné a hodnoty spojené unifikací zase rozpojují v opačném pořadí. Tato datová struktura je používána například i při čtení schématických odpovědí z prologu do programu v Javě. Dané unifikace proměnných a jejich hodnot se prováděly přímo při čtení. Před načtením další schématické odpovědi jsou takto načtené údaje backtrackovány a odebrány ze zásobníku provedených unifikací.

   Zajímavější je tato situace u čtení odpovědí, které přicházejí z externích zdrojů a mediátorů. Tam probíhá veškerá komunikace asynchronně. Odpovědi bývají běžně čteny z více zdrojů současně. Pro tento účel program obsahuje dvojí reprezentaci termů. První, chytřejší, dle popisu výše. Druhá slouží k asynchronnímu použití během čtení. Použité datové struktury jsou podobné, jen nelze termy v této reprezentaci mezi sebou dále unifikovat a není tedy potřeba řešit ani odunifikace. Při neexistenci reference na objekt s daným termem je tento term pak zapomenut mechanismem garbage kolekce Javy. Poté, co je term v této reprezentaci přijat z mediátoru a načten, je uložen do fronty, ze které si ho ve vhodnou chvíli převezme prostřednictvím řadiče odpovědí vhodná část doplňovače schématických důkazů dle daného

uspořádání. Ta si ho převede do backtrackovatelné reprezentace, kde už vše pracuje synchronně.

Situace se komplikuje, pokud by se díry nedoplňovaly v lineárním pořadí ale s nějakou mírou paralelizace. Předpokládejme například, že v našem příkladu z obrázku 2.4 doplňovač schématických důkazů enumeruje prvky tabulky populace a rád by je spojil s ostatními tabulkami. Není potřeba to provádět postupně a může se doplňování rozvětvit. Lze poslat dotaz do softwaru Mathematica, zda existuje vhodné spojení tabulek $lesseq a $greatereq současně s dotazy do tabulky *regexp* a do *okres_obec*. Záznamy z této tabulky by se pak spojovaly s tabulkou *kraj_okres*.

Přestože toto vyhodnocovač daného uspořádání schématické odpovědi neumí a bylo by ho třeba pro tento účel přepsat, tak jsou na toto datové struktury pro reprezentaci termů a unifikací připraveny.

Objekt reprezentující term totiž ukazatele na své podtermy nemá uložené přímo v sobě, stejně tak paměť toho, co se má backtrackovat není přímo uvnitř objektů reprezentující termy. Je stranou ve zvláštní třídě, která si pamatuje, které prologovské proměnné je přiřazena která hodnota. K tomu je použita třída HashMap z knihoven Javy. Tedy nalezení argumentu složeného termu trvá konstantní čas. Pokud bychom měli jen jediný objekt, který by si toto zobrazení pamatoval, tak by to bylo ekvivalentní tomu, kdyby si každý složený term pamatoval ukazatele na své podtermy sám. Takto můžeme mít těchto zobrazení více. Pokud povolíme skládání těchto zobrazení, tak bude tedy možné v místě paralelního větvení vytvořit nové objekty tohoto typu a od určité chvíle by se unifikace prováděné v paralelních větvích výpočtu neovlivňovaly.

Bylo by tedy možné enumerovat hodnoty z tabulky populace a s každou z nich provést toto rozvětvení a dále zbylé části schématické odpovědi řešit parerelně. To ale nebylo ozkoušeno vzhledem k tomu, že to obecnost není potřeba. Proto ani nebylo zjištěno, zda by tento způsob vyhodnocování umožňoval vyřešit některé příklady rychleji. Je ale zřejmé, že by se mohlo stát, že to některé příklady vyřeší pomaleji. Neúspěch v jedné větvi totiž může omezit velké množství práce v následující větvi. Ta by totiž mohla v paralelním případě vytížit nějaký zdroj a poslat do něj složité otázky, přestože při postupném vyhodnocování by tyto otázky ani nebyly odeslány.

3. Externí zdroje a mediátory k nim

3.1 Mediátory

Spolu se SPASS-XDB bylo implementováno několik mediátorů pro komunikaci s různorodými zdroji externích axiomů. V příkladech, které jsou součástí této práce, budeme používat následující z nich.

HTTPProxy - Tento mediátor přijímá otázky a protokolem HTTP komunikuje s Q&A serverem. Tento server pak komunikuje s externími zdroji a k tomu sám může používat další mediátory. Přestože adresa Q&A serveru je parametrem a může se nastavit libovolně, tak tento mediátor je tu především za účelem komunikace se zdroji dostupnými na adrese http://www.cs.miami.edu/ tptp/cgi-bin/SystemQATPTP. SPASS-XDB používal *HTTPProxy* napsanou v jazyce c, která používala proprietární knihovny. [17] Z důvodu snažší distribuce a možného použití na více strojích bez závislosti na instalaci daných knihoven byla v rámci této práce *HTTPProxy* přeimplementována v bashi.

XdbUTMediator - Tento mediátor slouží pro komunikaci se zdroji *Xdb universal translator* a *Xdm universal translator*. Bude použit v příkladech níže. Není součástí této práce a byl spolu s dalšími mediátory převzat ze SPASSu-XDB.

Prolog mediator slouží ke komunikaci s prologem. Báze znalostí pak může být zapsaná v prologu, a to buď jen jako seznam jednoduchých faktů, a nebo lze i využít programových vlastností prologu a platnost jednotlivých axiomů se pak odvozuje mechanismem prologu. Tento mediátor již existoval [17] a byl použit se SPASS-XDB. Bohužel není volně dostupný a tak byl s touto prací naimplentován znovu. Je napsán v prologu. Soubor s externími axiomy, a nebo s proceduramy, které je vypočítávají, načte jako běžný program v prologu. Poté naslouchá na standardním vstupu, odkud čte otázky ve formátu tptp_qa, překládá je na dotazy v prologu, které spouští. Nedeterminismus prologu umožňuje vracet více možných odpovědí. Tento mediátor je všechny přeloží na tptp answer dle daného protokolu a pošle je na standardní výstup.

XdbMediator - Tento mediátor komunikuje s databázemi ve formátu SPARQL.

Se SPASS-XDB byly dále implementovány mediátory do SQL, SPARQL a WWW. Ty nejsou v testovacích příkladech použity, ale vzhledem k tomu, že používají také protokol tptp_qa, tak by měli s programem LeanCoP-XDB také fungovat.

Z důvodu testování a porovnání LeanCoPu-XDB a SPASS-XDB byly v rámci této práce ještě vytvořeny následující mediátory:

CSV mediator - Tento mediátor dostane jako parametr cestu k souboru ve formátu CSV (comma separated values) a jednotlivé záznamy vnímá jako axiomy, které vrací v odpovědích na případné dotazy. Je napsán v bashi a vstupní soubor čte s každým dotazem celý znovu. Ke zpracování používá grep, sed a další unixové programy. Důsledkem toho je, že tento zdroj spotřebovává vyšší prostředky než by bylo nutné a není tak rychlý, jak by bylo možné. To není nevýhodou, ale naopak výhodou. Komunikace s externími zdroji bývá často pomalá - obvykle sice z důvodu, že se externí zdroj může nacházet na internetu na druhém konci

světa, ale takto se můžeme i s lokálním zdrojem přiblížit těmto podmínkám. Tím, že je tento zdroj také pomalý.

CSV generator mediator - Tento program je podobný *CSV mediátoru*, s nímž má společné části, ale používá rozdílný vstup. Tento mediátor dostává na vstupu cestu k programu, který umí generovat výstup ve formátu CSV. Dalším vstupem jsou parametry, s jakými se má daný program spustit, a informace o tom, kam mu předat relevantní část dotazu pro vygenerování odpovědi ve formátu CSV.

3.2 Statické externí zdroje

YAGOSUMO se skládá zhruba ze 14 milionů faktů o lidech, místech, klasifikaci entit a dalších věcech z báze znalostí YAGO [13] sjednocené s ontologií sumo. Tento zdroj dat je požadován vzorovými příklady ze SPASSu-XDB. S ním by mělo jít komunikovat přes *HTTPProxy* na Q&A server, který dále používá SQL mediátor k načítání dat z databáze. Zdroj *YAGOSUMO* začal být v průběhu psaní této práce dočasně (možná trvale) nedostupný. Z důvodu zachování funkčnosti příkladů a z důvodu umožnění vzájemného srovnání obou dokazovačů i na příkladech používajících tento zdroj, jsou součástí této práce malé tabulky ve formátu CSV a soubory s axiomy zapsanými v prologu, které tento zdroj nahrazují. S nimi se komunikuje pomocí *CSV mediátoru* a *prolog mediátoru*. U příkladů, kde je napsáno, že používají *YAGOSUMO* byli experimenty prováděny právě za použití těchto náhradních zdrojů. Ty vždy obsahují jen několik málo axiomů potřebných k vyřešení zadané úlohy. Jedinou výjimkou jsou axiomy pro predikátový symbol *capital_city*, ty budeme považovat za samostatný externí zdroj.

capital_city - Tento zdroj obsahuje informace o tom, která země má jaké hlavní město. Přestože má sloužit jako náhrada za část *YAGOSUMO*, tak zde neplatí, že by obsahoval jen axiomy potřebné k vyřešení úloh (navíc úloha *CloudyCapitals*, která bude představena ve 4. požaduje ke svému vyřešení vždy jiné axiomy v závislosti na počasí, které se zjišťuje zdrojem *Wather*). Proto jsou v tomto zdroji hlavní města všech evropských a některých mimoevropských zemí. S tímto zdrojem se komunikuje pomocí *CSV mediátoru*.

LinkedMDB - Linked Movie Database obsahuje informace o filmech a hercích. Tento zdroj nám mimo jiné umožňuje pro zadaný film zjistit, kteří herci v něm hráli a naopak pro zadaného herce zjistit, ve kterých hrál filmech. Komunikujeme s ním pomocí *XdbMediatoru*, který dále komunikuje se serverem data.linkedmdb.org. Tento externí zdroj není součástí této práce a byl převzat.

LatLong and Location - Tento zdroj umožňuje našim příkladům zjistit polohy měst (zeměpisnou šířku a délku). Komunikujeme s ním přes HTTPProxy. Tento externí zdroj není součástí této práce a byl převzat.

Mondial - Z tohoto zdroje si příklady získávají infomrace o světových městech a zemích. Dále tento zdroj umožňuje pro dvě zadané země zjišťovat délku jejich společné hranice. Komunikujeme s ním přes *HTTPProxy*. Tento externí zdroj není součástí této práce a byl převzat.

Dále práce obsahuje několik lokálních zdrojů ve formátu CSV, ke kterým se přistupuje pomocí *CSV Mediatoru*.

Family tree je zdroj, který obsahuje rodokmen anglického krále Jakuba I. Stuarta a jeho potomků. Data byla do tohoto zdroje v rámci této práce ručně

doplněna, jejich obsah i struktura je vyobrazena na obrázku v příloze B.

Zdroje **kraj_okres**, resp. **okres_obec** poskytují informace o tom, který český okres je ve kterém kraji, resp. která (česká) obec s rozšířenou působností je ve kterém okrese. Data do toho zdroje byli naplněny dle definice obcí s rozšířenou působností stanovenou v zákoně [1].

populace - Tento zdroj komunikuje se stránkou www.kdejsme.cz, která obsahuje data, které zpřístupnilo Ministerstvo vnitra ČR [18] o četnosti příjmení v české republice. S tímto zdrojem se komunikuje pomocí mediátoru *CSV generator mediator*.

3.3 Dynamické externí zdroje

Dynamické externí zdroje jsou takové zdroje dat, které poskytují informace, které se v průběhu času mění. Mohou to být informace o počasí, o kurzech měn či akcií. Ve srovnávacích příkladech je použit zdroj **Weather**. Ten poskytuje informace o počasí ve světě.

3.4 Výpočetní externí zdroje a služby

Dalším možným použitím externích zdrojů je výpočet něčeho. To v původním smyslu slova už není jen externí seznam axiomů. Je to spíše služba, která se chová jako zdroj axiomů, jejichž prostřednictvím předává informace dokazovači. V příkladech budou použity následující zdroje tohoto typu:

RegExp - Tento zdroj ověřuje, zda výraz odpovídá zadanému vzoru. Oba výrazy musí být zadány jako vstup. Tento externí zdroj není součástí této práce a byl převzat.

LookDifferent - Tento zdroj ověřuje, zda je zadaný výraz syntakticky rozdílný od druhého. Speciáně u teorií s rovností se liší od operátoru != tím, že porovnává pouze názvy konstant a nezjišťuje skutečnou nerovnost. Tento externí zdroj není součástí této práce a byl převzat.

PrintTTY - Tento zdroj je podporován pouze z důvodu kompatibility s příklady ze SPASS-XDB. Na každý dotaz na existenci axiomu ve tvaru *print(neco)* vrátí axiom stejného tvaru a jako vedlejší efekt *neco* vypíše na standardní výstup. Tento externí zdroj není součástí této práce a byl převzat.

3.5 Matematika a aritmetika

Pro řešení matematických / aritmetických úloh je k dispozici pro ně speciálně určený externí zdroj **Mathematica**. Tento zdroj je realizován softwarem Wolfram Mathematica, který běží na Q&A serveru. S ním LeanCop-XDB a SPASS-XDB komunikují prostřednictvím *HTTTProxy*. Dále se pak na straně Q&A serveru v Miami nachází ještě druhý mediátor [14], který komunikaci v protokolu tptp_qa překládá do syntaxe, které rozumí software Mathematica. Konkrétně používá funkci FindInstance z jazyku Mathematica, která pak hledá vhodnou substituci proměnných tak, aby byly atomy z původního dotazu splněny. Tento externí zdroj je dále výjimečný tím, že jako jediný umí odpovídat na dotazy, které se skládají z více atomů.

V případě, že dokazovač odešle dotaz, který umí *Mathematica* vyřešit, ale výsledek nelze popsat jazykem TPTP, tak Mathematica vrátí jako výsledek nový konstantní symbol v jehož názvu je zakódován výsledek. V případě, že dokazovač s tímto symbolem pracuje a bude ho chtít použít v následujícím dotazu, tak tomu *Mathematica* porozumí a bude s hodnotou správně pracovat.

Tato situace může nastat například u dotazu $\exists X : X * X = 2$. Mathematica odpoví, že $X = \sqrt{2}$. Mediátor na straně serveru zařídí, aby $\sqrt{2}$ byla dokazovači vnímána jako název konstantního symbolu.

4. Experimentální část

4.1 Úvod

V této kapitole budou popsány provedené experimenty srovnávající LeanCoP-XDB se SPASSem-XDB. Všechny testy byly prováděny na počítači s následující konfigurací:

Operační systém	Linux 2.6.32-37-generic, Ubuntu 10.04.3 LTS
RAM	3GB
Procesor	Intel(R) Core(TM)2 Duo CPU T6670 @ 2.20GHz

U všech výstupů z experimentů se budeme řídit následujícím pravidlem. Hodnoty, které zde uvedeme jsou hodnoty z prvního spuštění daného testu, pokud nebude uvedeno jinak. Pokud byl experiment opakován nebo pokud zde bude uveden průměr z více spuštění nebo jiný statistický údaj, tak na to bude speciálně upozorněno.

Při spuštění LeanCoPu-XDB s optionem `-s` se vypíše tabulka se statistikami. Konkrétně budeme měřit následující údaje:

1. CPU čas hlavního vlákna s dokazovačem.

   Do této hodnoty jsou započítány doby, během kterých se provedly následující činnosti:

   - Načtení a zpracování vstupních souborů.
   - Parsování schématických odpovědí.
   - Uspořádání schématických odpovědí.
   - Vyhodnocování dle daného uspořádání.
   - Sestavování otázek.
   - Převody mezi reprezentacemi termů.
   - Spojování odpovědí.

   V tomto čase není započítána doba na zpracování a parsování odpovědí na dotazy. To obstarávají jiná vlákna.

   Tento údaj je v tabulce na řádku `CPU time elapsed`.

2. Skutečný čas, který uběhl.

   Tento čas je měřen jako rozdíl dvou hodnot vrácených funkcí Javy `System.currentTimeMillis()`. Čas startu se měří úplně prvním příkazem programu, ještě před parsováním argumentů příkazové řádky a před vším ostatním. Čas konce se měří až při ukončování programu v okamžik, kdy program vypisuje daný řádek se statistikou.

   Tento údaj je na řádku: `Wall-clock time elapsed:`.

3. Počet jedinečných schématických odpovědí, které LeanCoP vygeneroval a které byly doplňovány.

Někdy se stane, že LeanCoP vygeneruje stejnou schématickou odpověď (až na přejmenování proměnných) vícekrát. To je způsobeno především iterativním prohlubováním, které LeanCoP používá. Schématická odpověď vygenerovaná v jedné hloubce se pak generuje opakovaně i s prohledáváním každé další větší hloubky. Z tohoto důvodu jsou proměnné ve všech schématických odpovědích přejmenovány a cachovány. U každé nové schématické odpovědi se tedy kontroluje, zda je nová a nebo, zda takováto schématická odpověď již byla zkoušena. K této kontrole je použita datová struktura založená na hashování, takže kontrola probíhá v konstantním čase vzhledem k počtu již přijatých odpovědí. Tento údaj je na řádku `leancop core` ve sloupci `axioms`.

4. Počet schématických odpovědí, které LeanCoP vygeneroval, ale nebyly doplňovány z důvodu, že se opakují.

   Tento údaj je na řádku `leancop core` ve sloupci `cache`.

5. Čas, jak dlouho bylo hlavní vlákno dokazovače blokováno procesem LeanCoPu při čekání na novou schématickou odpověď. U takto lehkých úloh, jaké byly testovány, většinou stíhá LeanCoP generovat schématické odpovědi rychleji, než je stačí LeanCoP-XDB ověřovat. K blokování tedy většinou nedochází. V tomto případě je tedy tento čas jen doba čekání na první schématickou odpověď.

   Tento údaj je na řádku `leancop core` ve sloupci `time`.

6. Počet termů, které program musel naparsovat a čas tím strávený.

   Tento údaj je na řádku `term translator`.

7. Počet otázek odeslaných do externích zdrojů axiomů.

   Tento údaj je ve sloupečku `queries`.

8. Počet otázek, ke kterým již začal přicházet balík s odpověďmi.

   Tento údaj je ve sloupečku `started`.

9. Počet otázek, ke kterým již dorazil celý balík odpovědí.

   Tento údaj je ve sloupečku `finished`.

10. Počet axiomů, které přišly v odpovědích.

    Tento údaj je ve sloupečku `axioms`.

11. Čas, po který byl dokazovač blokován čekáním na externí zdroje.

    Tento údaj je ve sloupečku `time`.

Údaje z posledních pěti bodů jsou měřeny jak v součtu pro všechny externí zdroje axiomů dohromady v řádku `TOTAL SOURCES`, tak i pro každý zdroj zvlášť.

Ukázková tabulka se statistikami, která je výstupem LeanCoPu-XDB je ve výpisu 4.1.

Měření statistik pro SPASS-XDB je o trochu komplikovanější. Přestože lze pomocí optionů nastavovat množství výpisů, není možné si nechat vypsat jen statistiky a ani nejsou v tak přehledné tabulce. Proto byl v rámci této práce

Source:	queries	started	finished	axioms	cache	time/ms
populace	1	1	1	86	0	334
kraj_okres	13	13	13	1	1	318
okres_obec	16	16	16	13	0	394
regexp	1	1	1	1	0	3
greatereq	1	1	1	1	0	921
SOURCES TOTAL	32	32	32	102	1	1970
term translator	129	129	129	129	0	111
leancop core	1	1	0	1	0	68
CPU time elapsed:						270
Wall-clock time elapsed:						2324

Obrázek 4.1: Statistika LeanCoPu-XDB na příkladu *Sobota*

Source:	queued	asked	axioms	time
eq	29	17	0	
less1	9	0	0	
lesseq	5	0	0	
greatereq	8	3	0	
populace	1	1	85	
obec_kraj	12	12	3	
obec_okres	151	17	13	
regexp	25	25	8	
SOURCES TOTAL	240	75	109	
Wall-clock time elapsed:				00:27.23

Obrázek 4.2: Statistika SPASSu-XDB na příkladu *Sobota*

napsán program v Haskellu, který parsuje výstup SPASS-XDB, čte všechny zajímavé údaje a dává je také do přehlednější tabulky, která je podobná výstupu z LeanCoPu-XDB. Tento program je přiložen na CD.

U SPASSu-XDB budeme měřit počet odeslaných otázek, počet otázek ve frontě připravených k odeslání a počet přijatých axiomů. Tyto charakteristiky měříme, jak pro každý zdroj zvlášť, tak v součtu pro všechny zdroje dohromady. Parser termů je ve SPASSu-XDB rychlejší než v LeanCoPu-XDB a čtení vstupů u většiny testovaných příkladů je zanedbatelné, trvá nejvýše jednu vteřinu. Dále budeme měřit ještě celkovou dobu běhu. Měřili bychom toho víc, ale další zajímavé údaje se už ve výstupu SPASSu-XDB nenacházejí. Měření něčeho dalšího by tedy nebylo triviální a znamenalo by zřejmě úpravy zdrojových kódů SPASSu-XDB, což není součástí této práce. Ukázka statistik získaných z běhu SPASSu-XDB je vidět na výstupu 4.2.

V následujících sekcích provedeme samotné porovnávání na několika příkladech, které necháme řešit oba dokazovače.

4.2 Příklad *Sobota*

Zadáním této úlohy je najít obec v Karlovarském kraji, která má v názvu „Z“ nebo „ov“ tak, aby v této obci žilo 5 až 10 lidí s příjmením „Sobota“. Zadání je tedy přímo stavěné tak, aby vedlo na jedinou schématickou odpověď a to na tu, která byla uvedena jako ukázka 2.1.

Zdroje jsou kvantifikovány podobně jako v ukázce 2.3. Jediným rozdílem je, že zdroje $lesseq a $greatereq byly kvantifikovány existenčně. Bylo tomu tak z z důvodu, abychom mohli použít stejné specifikace tohoto zdroje i v dalších úlohách. V této úloze to dává jen možnost o něco větší volby v pořadí vyhodnocování, což ničemu nevadí. V ukázkách byla přísnější kvantifikace jen z ilustračních důvodů.

Dokazovače budou mít při řešení této úlohy k dispozici externí zdroje axiomů *kraj_okres*, *okres_obec*, *populace*, *RegExp* a *Mathematica*.

4.2.1 Příklad *Sobota* řešený LeanCoPem-XDB

Atomy jsou ve schématické odpovědi 2.1 uvedeny v pořadí, ve kterém je lze doplnit. LeanCoPem-XDB volí první použitelné uspořádání, a proto si zvolil právě tuto permutaci. Po doplnění vypadá schématická odpověď jako v ukázce 2.2. Z tabulky 4.1 se statistikami LeanCoPu-XDB lze vyčíst trochu více informací o tom, co se dělo.

Z řádku `leancop core` je vidět, že odpověď byla nalezena už při zkoušení první schématické odpovědi. Z uspořádání schématické odpovědi lze vyčíst, že se jako první odesílal dotaz do tabulky populace a z odpovídajícího řádku tabulky vidíme, že tento dotaz byl odeslán pouze jediný. Dle sloupečku `axioms` na tento dotaz dorazilo 86 odpovědí. Příjmení Sobota se tedy vyskytuje v 86 českých obcích.

Další v pořadí je atom *okres_obec* a z odpovídajícího řádku můžeme zjistit, že celkem u 16 z těchto 86 obcí bylo zjišťováno, do kterého okresu patří a ve sloupečku `axioms` je vidět, že to LeanCoP-XDB dokázal zjistit ve 13 případech z 16. To neznamená, že by 3 obce nepatřili do žádného okresu, ale jen to, že některé záznamy v tomto zdroji mohou chybět. To je ale pro účel naší úlohy nepatrný rozdíl, který hledání jen mírně ztěžuje.

Následuje atom *okres_kraj*. Z daného řádku statistiky je vidět, že u všech 13 okresů bylo testováno, zda náleží do Karlovarského kraje. Prvních 12 okresů bylo odjinud a až 13. odpovídal. Dále bylo testováno, zda takto nalezená obec má v názvu „Z“ nebo „ov“ a zda počet výskytů příjmení Sobota v této obci je v rozsahu 5 až 10. Z jedniček na řádcích `regexp` a `greatereq` je vidět, že toto ověření se už povedlo na první pokus a byla nalezena obec Sokolov.

Tento způsob prohledávání zvolený v ukázkové implementaci je vlastně prohledáváním do hloubky. Pokud by hned první obec z Karlovarského kraje neměla v názvu „ov“, tak by se prohledávání vrátilo k předchozímu kroku, který by se zkoušelo vyřešit jiným způsobem. Takto by se prohledávání vynořovalo, až by časem začalo ověřovat 17. z 86 obcí, kde se nachází příjmení Sobota.

4.2.2 Příklad *Sobota* řešený SPASSem-XDB

Při řešení této úlohy došel SPASS-XDB ke stejnému důkazu jako LeanCoP-XDB. Z tabulky 4.2 lze vyčíst více o tom, jak postupoval. SPASS-XDB řadí otázky,

na které by se mohl zeptat, do fronty. Proto ve sloupci `queued` vidíme množství otázek, které bylo vygenerováno, ale až ve sloupci `asked` je znázorněno, kolik otázek bylo odesláno. Rozdílem je pak počet vygenerovaných, ale nikdy neodeslaných otázek. To lze intuitivně interpretovat jako otázky, na které SPASS-XDB napadlo se zeptat, ale neudělal to.

Dále je v tabulce vidět, že má o několik řádků navíc. To je způsobeno tím, že SPASS-XDB použil nebo zvažoval použití jiných zdrojů. Konkrétně přibyly navíc řádky: `eq`, `less` a `lesseq`.

Zdroj `lesseq` je v této tabulce z pochopitelných důvodů. K důkazu je totiž třeba porovnat počet obyvatel dané obce s číslem 10. Tento zdroj není v tabulce LeanCoPu uveden, protože bylo ověřeno náležení do rozsahu 5 až 10 odesláním dotazu o dvou literálech. To je bráno jako odeslání jediného dotazu. Proto je to uvedeno v tabulce se statistikou LeanCoPu-XDB na jediném řádku.

Zbývá tedy vyřešit, proč přibyly řádky pro `eq` a `less`. Odůvodníme si nejprve `less`. To je způsobeno tím, že se specifikací externích zdrojů axiomů je dokazovačům předán ještě i interní axiom $a < b \Leftrightarrow \neg a \geq b$.

Pomocí externích axiomů tedy mohou dokazovače najít a a b takové, že $a < b$. Také je možné najít a a b takové, že $a \geq b$. Dokazovače se mohou v externích zdrojích axiomů snažit hledat takové a a b, aby byly splněny obě podmínky současně. To je jedna z následujících věcí, o kterou by se LeanCoP-XDB pokoušel, pokud by se mu nepovedlo doplnit schématický důkaz, který nalezl jako první.

4.3 Příklad *AbeMammal*

Zadáním tohoto příkladu je pomocí axiomů z externího zdroje *YAGOSUMO* dokázat, že Abraham Lincoln je savec.

$instance('AbrahamLincoln', mammal)$

Toto zadání je zveřejněné na stránce SPASSu-XDB a obsahuje dva predikátové symboly: *instance* a *subclass*. Dále tento problém obsahuje konstantní symboly: $setOrClass$, $mammal$, $primate$, $hominid$, $human$ a $'AbrahamLincoln'$. K dispozici byly v původním zadání následující axiomy:

1. savec, primát, hominoid, člověk jsou instancí třídy $setOrClass$.

   $instance(mammal, setOrClass)$

   $instance(primate, setOrClass)$

   $instance(hominid, setOrClass)$

   $instance(human, setOrClass)$

2. Primáti jsou podtřídou savců. Hominoidi jsou podtřídou primátů. Lidé jsou podtřídou hominoidů.

   $cubclass(primate, mammal)$

   $subclass(hominid, primate)$

   $subclass(human, hominid)$

3. Dále nejsložitější a nejzajímavější logický axiom:

   Pokud je X instancí třídy $setOrClass$, Y instancí třídy $setOrClass$, X podtřídou Y a Z instancí X, pak je Z i instancí Y.

   $\forall X \forall Y \forall Z : ($

   $instance(X, setOrClass) \&$

   $instance(Y, setOrClass) \&$

   $subclass(X, Y) \&$

   $instance(Z, X)$

   $) \Rightarrow instance(Z, Y)$

Dále byl k dispozici zdroj externích axiomů, který byl schopný dodat axiomy typu *instance* a tedy říkat, co je instancí čeho.

K dokázání této věty je důležité, aby dokazovač od tohoto externího zdroje zjistil axiom, že Abraham Lincoln byl člověk. Jelikož pro oba konstantní symboly hominoid i člověk platí, že jsou třídou a jelikož zároveň platí, že člověk je podtřídou třídy hominoid, tak platí, že Abraham Lincoln je hominoid. Opakovaným použitím tohoto pravidla lze domněnku, že Abraham Lincoln je savec, dokázat.

Na tomto příkladu je dobře vidět rozdílný přístup obou dokazovačů. SPASS-XDB při řešení tohoto problému odesílal následující otázky:

1. Je Abraham Lincoln instancí třídy savec?

   Tato otázka je rozumná a pokud by přišla pozitivní odpověď znamenalo by to, že je celá úloha vyřešená.

2. Jaké existují třídy?

   Tato otázka také může pomoci k vyřešení dané úlohy. Je ale trochu riskantní se na takto obecnou otázku ptát. V externím zdroji, který byl pro tuto úlohu vyroben, se vyskytují jen axiomy, které jsou pro vyřešení úlohy potřebné, takže položení této otázky SPASSu-XDB nezpůsobí potíže. Pokud bych ale měl jako externí zdroj rozsáhlejší ontologii, pak by na takovouto otázku mohlo přijít velké množství axiomů.

3. Čeho všeho je instancí Abraham Lincoln?

   Zde by mohl nastat také problém zmíněný již u předchozího bodu. Dále je tato otázka také odvážnější, než by bylo třeba. Už byla odeslána otázka na to, jestli je savec, dále by bylo možné se ptát, zda je instancí tříd, které se vyskytují v zadání problému.

4. Co vše je instancí třídy člověk?

   Pozor, tato otázka by při použití reálné databáze obsahující i jiné osobnosti než Abrahama Lincolna vedla k velkému množství odpovědí.

   Dalším zajímavým pozorováním je, že znalost dalších lidí nikdy dokazovači nemůže pomoct vyřešit tuto úlohu.

5. Co vše je instancí třídy hominoid?

6. Co vše je instancí třídy primát?

7. Co vše je instancí třídy savec?

   Tyto tři otázky mají opět stejný problém - jejich zodpovězení opět nemůže pomoct úlohu vyřešit. Jediné, co by pomohlo, by byl fakt, zda je Abraham Lincoln instancí těchto tříd. Na to se SPASS-XDB zeptal v následujících otázkách.

8. Je Abraham Lincoln primát?

9. Je Abraham Lincoln hominoid?

10. Je Abraham Lincoln člověk?

    Zajímavé je, že tyto otázky vůbec nemusely být odeslány, protože jsou konkrétnější, než ty dříve odeslané a v odpovědích na ně lze nalézt i odpovědi na tyto otázky.

    Přestože už má SPASS-XDB dostatek informací pro sestavení důkazu, tak posílá ještě další otázky:

11. Je třída instancí třídy?

12. Je AbrahamLincoln třída?

    Tyto otázky nedávají moc smysl. Na závěr se SPASS-XDB zeptal ještě na nejobecnější možnou otázku:

13. Co je instancí čeho?

Při tomto testování se SPASS-XDB nezahltil jen díky tomu, že jsme k testování nepoužili skutečnou ontologii, ale jen externí zdroj obsahující přesně vhodné axiomy.

Tento problém byl pravděpodobně při vývoji SPASSu-XDB řešen tím, že protokol tptp_qa mohl obsahovat volitelné parametry, které omezovaly maximální množství axiomů, které může externí zdroj vrátit. Tento limit se mohl pro každou úlohu stanovit různý a to vedlo k tomu, že SPASS-XDB úlohu vyřešil v rozumném čase i s větším externím zdrojem za cenu ztráty úplnosti.

Ale i při testu, který byl navržen tak, aby zadání nebylo pro SPASS-XDB zbytečně komplikované, se projevilo, že LeanCoP-XDB je na této úloze rychlejší.

SPASSu-XDB tímto postupem trvalo vyřešení úlohy 1,17 vteřin.

```
      Source: queued asked axioms time
    instance      13     13     12
SOURCES TOTAL     13     13     12
Wall-clock time elapsed:          00:01.17
```

Z tabulky s přehledem je vidět, že bylo položeno jen 13 otázek a na ně přišlo jen 12 axiomů. To by byl velmi dobrý výsledek - bohužel ho ale bylo dosaženo jen teď - při konfiguraci, kdy externí zdroj neobsahoval žádné nepotřebné axiomy.

Za tohoto předpokladu překonává SPASS-XDB LeanCoP-XDB jak v počtu odeslaných otázek, tak v počtu axiomů, které bylo potřeba zpracovat.

```
      Source: queries started finished axioms cache time/ms
    instance      56      56       56      36    49      88
SOURCES TOTAL     56      56       56      36    49      88
```

```
term translator      82      82      82      82      0      64
   leancop core       1       1       0      21     15     190
CPU time elapsed:                                          290
Wall-clock time elapsed:                                   480
```

LeanCoP-XDB odeslal celkem 56 dotazů a zpracovával 36 odpovědí. Díky lepší implementaci ale dokázal toto (více než dvojnásobné) množství komunikace zpracovat za méně než poloviční čas oproti běhu SPASSu-XDB. Celkem za 0,48 sekund. V případě, že by bylo implementováno použití axiomů z cache, tak by pak tento čas bylo možné ještě víc zkrátit. 49 z 56 dotazů by nemuselo být odesláno. Bylo by odesláno celkem jen 7 dotazů. Čas, po který by byl blokován dokazovač čekáním na zdroj s_instance by mohl klesnout na zhruba sedm padesátišestin z původních 88 ms. Čas na jejich parsování by byl také ušetřen. CPU time strávený prohledáváváním opakujících se větví, by mohl být menší. 190 ms, během kterých počítal LeanCoP-XDB, by bylo zachováno. Stejně tak zhruba 10 ms, po které bylo načítáno zadání, by se nezměnilo. Všechny ostatní časy by se zkrátili zhruba na sedm padesátišestin. Po provedení výpočtu lze zjistit, že s cachováním výsledků by bylo možné čas na této úloze strávený zkrátit až na zhruba 233 ms.

Velmi zajímavý je však způsob jaký LeanCoP-XDB použil. Iterativní prohlubování generovalo možné schématické odpovědi v pořadí od nejjednodušších ke složitějším:

- `instance('Abraham Lincoln', mammal)`

  Nejsnažší schématická odpověd, že by byla domněnka součástí externího zdroje, je vygenerována jako první. To ještě SPASS-XDB také udělal.

- `instance('Abraham Lincoln', primate)`

  Druhé nejjednodušší řešení bylo vygenerováno jako druhé.

- `instance(mammal, setOrClass)`
  `instance('Abraham Lincoln', primate)`

  Tato nečekaná schématická odpověď dokonce oproti předchozí odpovědi obsahuje jen atom navíc a zbytek zůstal stejný.

  To je způsobeno tím, že axiom $instance(mammal, setOrClass)$ je zároveň interním axiomem. Nedeterminismus prologu, ve kterém byl napsán LeanCoP způsobuje, že pokud lze použít k důkazu více různých způsobů použitím různých axiomů, tak se pro takové způsoby vygeneruje více schématických odpovědí. Domněnku o Abrahamu Lincolnovi lze dokázat jak s použitím interního axiomu, tak také pokud by se použil axiom $instance(mammal, setOrClass)$, který by byl nalezen v externím zdroji.

  Zde jsou tedy vidět hned dva způsoby, jak by bylo možné LeanCoP-XDB do budoucna vylepšit. Prvním z nich by bylo zabránit generování schématických odpovědí, které v externí databázi hledají stejný axiom, jako je v interní databázi. Druhým možným vylepšením by bylo okamžité zamítání schématických odpovědí, které jsou specifičtější než některá dřívější odpověď, která už byla vyhodnocována a je v aktuální odpovědi obsažena.

  Dále budou v tomto výpisu uvedeny jen schématické odpovědi, u kterých se toto neděje.

- První takovou je:

  ```
  instance('Abraham Lincoln', hominid)
  ```

- A hned další takováto schématická odpověď je:

  ```
  instance('Abraham Lincoln', human)
  ```

  Pro ní už se jen ověří v databázi, že Abraham Lincoln je skutečně člověk a úloha je vyřešena.

Na tomto postupu je pozoruhodné, že všechny schématické odpovědi se skládají pouze z ground atomů a tedy i otázky, které se budou pokládat, nebudou obsahovat žádné volné proměnné. To způsobuje, že na každou otázku může přijít nejvýše jedna jediná odpověď.

Jak ukážeme v dalších experimentech, množsví dat v databázi může u této úlohy mít velmi velký vliv na efektivitu SPASSu-XDB a naopak by nemělo mít žádný vliv na efektivitu LeanCoPu-XDB.

Napřed ale zkusme ještě tuto úlohu o malinko ztížit. K dokázání byly potřeba axiomy o tom, že savci, primáti, hominoidi a lidé jsou instancí třídy třída. Tyto axiomy byly v původní úloze vloženy do zadání jako interní axiomy. Když už ale máme zdroj, který umí říkat, co je instancí čeho, tak ho můžeme využít a axiomy přesunout do něj.

Tento zdánlivě těžší problém vyřešil LeanCoP-XDB značně rychleji, odeslal méně dotazů a zpracovával méně odpovědí.

```
          Source: queries started finished axioms cache time/ms
        instance        18      18       18     14    10      73
  SOURCES TOTAL         18      18       18     14    10      73

term translator         28      28       28     28     0      27
   leancop core          1       1        0      5     3     103
CPU time elapsed:                                             200
Wall-clock time elapsed:                                      331
```

To je způsobeno právě tím, že nezvažoval dvakrát použití stejných axiomů, jednou interně, jednou externě. Přitom stále ještě zůstává LeanCoPu-XDB potenciál být o trochu rychlejší, kdyby uměl cacheování. To už by ale mohlo v tomto případě přinést zrychlení jen o několik desítek milisekund.

Toto stížení nemělo na SPASS-XDB žádný vliv, nebo ho mírně zpomalilo:

```
      Source: queued asked axioms time
  s__instance     15     15     16
SOURCES TOTAL     15     15     16
Wall-clock time elapsed:          00:01.21
```

Z času, který uběhl je vidět, že je o něco horší. Toto zdánlivé stížení, které SPASS-XDB brzdí (významnost rozdílu byla otestována opakováním experimentu), LeanCoPu-XDB pomáhá.

Ze sledování komunikace lze zjistit, že jako první se SPASS-XDB ptal jako obvykle, zda je Abraham Lincoln savec. Druhá otázka také zůstala stejná: „Jaké existují třídy?“ Hned ale jako třetí se SPASS-XDB ptal na nejobecnější možnou otázku: „Co je instancí čeho?“, která by potenciálně mohla vracet hodně axiomů.

Je možné, že autoři této úlohy, kteří ji se SPASSem-XDB vymysleli, předpokládali, že zdroj instance bude kvantifikován tak, aby musela být jedna nebo druhá proměnná vždy zadána. V dalším experimentu, opět na úloze o Abrahamu Lincolnovi, porovnáme vliv velkého množství dat v externím zdroji na efektivitu dokazovače. Abychom se více přiblížili podmínkám, pro které byla úloha původně zamýšlena, bude zdroj instance bude dále kvantifikován takto:

```
! [Predmet] : ? [Trida] : instance(Predmet,Trida)
! [Trida] : ? [Predmet] : instance(Predmet,Trida)
```

To lze interpretovat následovně: Zdroj umí pro zadaný předmět hledat třídy do kterých patří a pro zadanou třídu hledat její instance. Na LeanCoP-XDB v této úloze nebude mít vliv a SPASSu-XDB to pomůže, protože mu nepřijde veliké množství axiomů jako odpověď na otázku „Co je instancí čeho?“. Na tu se totiž nebude smět zeptat. To mu trochu pomůže v následující úloze.

4.4 *AbeMammal* a k němu nepotřebné axiomy navíc

Domníváme se, že by přidání dalších axiomů do externího nemělo mít vliv na efektifitu LeanCoPu-XDB, jelikož by se do se tyto axiomy do LeanCoPu-XDB ani neměly dostat. Dále víme, že způsob dotazování, který SPASS-XDB používá povede k tomu, že si SPASS-XDB i tyto axiomy stáhne. Lze se tedy domnívat, že by ho to mohlo zpomalit. Zda tomu tak skutečně je ověříme tímto experimentem: Do externího zdroje axiomů přidáme několik axiomů navíc. Budou mít tento tvar:

instance(A, setOrClass), kde místo *A* budou různé konstantní symboly.

Tím říkáme, že kromě savců, primátů, hominoidů a lidí budeme mít v úloze ještě další třídy. Abraham Lincoln a ani nic jiného do těchto tříd nebude patřit. Tedy tyto třídy ani nebude možné k důkazu naší domněnky jakkoli použít a porovnáme, co to udělá s časem výpočtu obou dokazovačů.

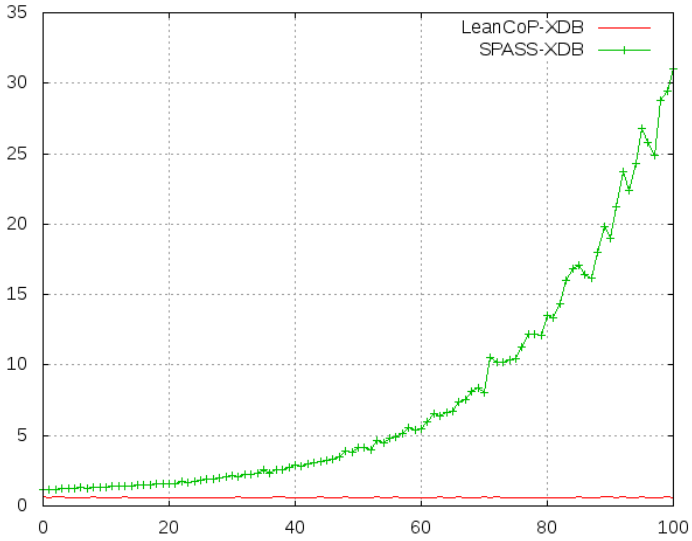
Obrázek 4.3 zachycuje závislost počtu zbytečných axiomů, které nemohou vést k řešení úlohy na čase, který dokazovače potřebují k dokázání dané domněnky.

Na X-ové ose jsou počty axiomů, které byli k původnímu problému přidány. Na Y-ové ose je uveden čas v sekundách, který na úloze dokazovače strávily. Červená křivka je pro LeanCoP-XDB a je zde vidět, že přebytečné axiomy v externím zdroji nemají na dobu běhu dokazovače vliv. Na zelené křivce je SPASS-XDB a na bodech na ní vyznačených můžeme vidět, že množství axiomů, které je v externím zdroji nadbytečné, může výrazně ovlivnit čas běhu programu.

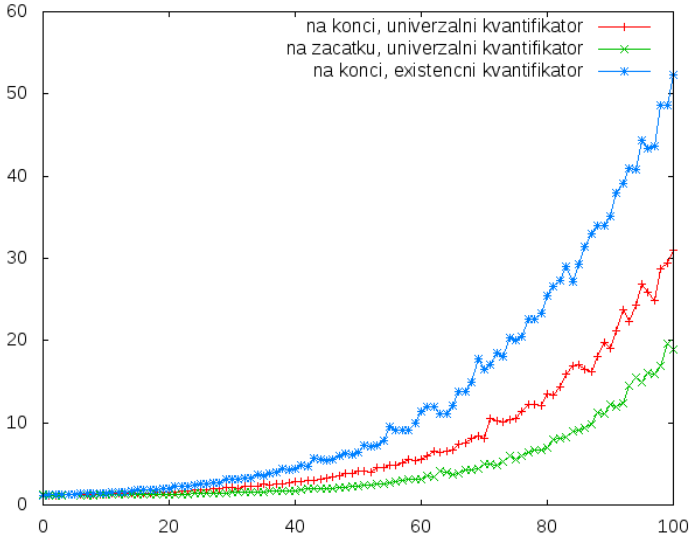
S rostoucím množstvím axiomů v externím zdroji se více projevuje výhodnost opatrného přístupu LeanCoPu-XDB, který posílá jen potřebné dotazy a na rozdíl od SPASSu-XDB se nenechá se ovlivnit zbytečnými daty v externích zdrojích.

Ještě o něco horšího výsledku by SPASS-XDB dosáhl, pokud by byl zdroj instance kvantifikován existenčně. Naopak by SPASSu-XDB pomohlo, pokud by axiomy byli v externím zdroji ve vhodném pořadí, takovém, že při otázkách na které se ptá, přijdou odpovědi, které jsou v důkazu potřeba jako první. I přesto by ale byl SPASS-XDB zpomalován větším množstvím dat v externím zdroji.

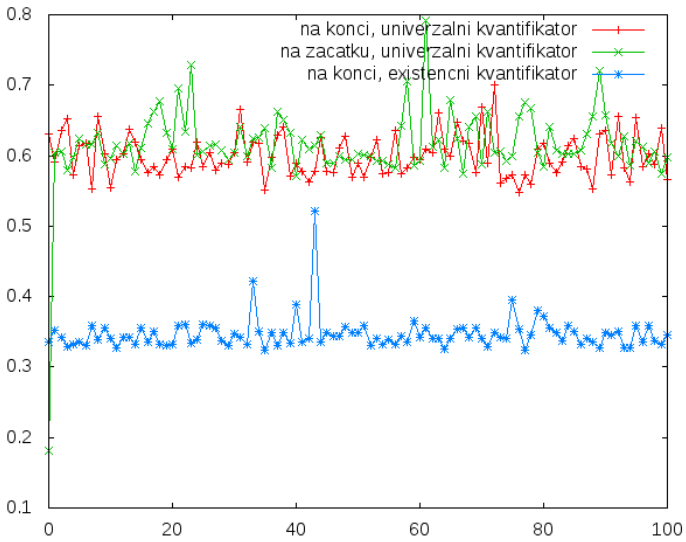
Obě tyto alternativy byly testovány s přidáním 0 až 100 zbytečných axiomů. Výsledky jsou znázorněny na obrázku 4.4. X-ová osa opět značí počet přidaných axiomů navíc. Na Y-ové ose je zaznamenán čas v sekundách. Modrá křivka značí



Obrázek 4.3: AbeLincoln, závislost na nepotřebných axiomech



Obrázek 4.4: AbeLincoln a nepotřebné axiomy, 3 varianty pro SPASS-XDB



Obrázek 4.5: AbeLincoln a nepotřebné axiomy, 3 varianty pro LeanCoP-XDB

výsledky, pokud byl zdroj kvantifikován univerzálně a potřebné axiomy byli až na konci za nepotřebnými. Červená křivka odpovídá konfiguraci z předchozího porovnání - zdroj byl kvantifikován tak, aby musel být jeden parametr zadán a na zelené křivce je totéž v situaci, kdy se SPASSu-XDB pomohlo přesunutím potřebných axiomů na začátek.

Přestože by žádná z těchto variant zadání neměla mít vliv na otázky odesílané LeanCoPem-XDB, tak se domníváme, že množství axiomů v externím zdroji může mít malý vliv na celkovou dobu běhu LeanCoPu-XDB z důvodu, že by externí zdroj axiomů mohl odpovídat pomaleji. Proto bylo pro každou z těchto tří variant zadání bylo spuštěno všech 101 testů pro 0 až 100 přidaných axiomů. Výsledky testů jsou znázorněny v grafu 4.5.

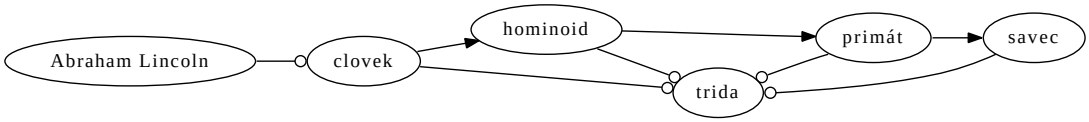
Tento výsledek ukazuje, že takto malá množství axiomů, jako 1 až 100, ve zdroji, se kterým komunikujeme prostřednictvím *prolog mediatoru*, má zanedbatelný vliv na dobu, za jakou mediátor odpoví na otázky. Bylo tedy potvrzeno, že doba běhu LeanCoPu-XDB u tohoto příkladu nezávisí na počtu přidaných axiomů. Po přidání stovky axiomů navíc, libovolná varianta tohoto zadání trvá SPASSu-XDB desítky sekund, než ji vyřeší. LeanCoP-XDB ji ale stále zvládá za zlomek vteřiny.

Dalším zajímavým pozorováním je, že varianta zadání s existenčním zdrojem značená modrou křivkou, která byla pro SPASS-XDB nejhorší, byla pro LeanCoP-XDB nejlepší.

To je pravděpodobně způsobeno tím, že přísněji kvantifikovaný externí zdroj byl kvantifikovaný dvěma způsoby. Ukázková implementace LeanCoPu-XDB v takovémto případě otevírá do zdroje dvě spojení. Řadič otázek musí u každé otázky rozhodovat, do kterého spojení otázku poslat a kontrolovat, které části otázky jsou instanciované. Až teprve poté je ověřeno, že lze otázku odeslat do libovolného ze dvou otevřených spojení. I přesto jsou ale všechny tyto výsledky velmi dobré.

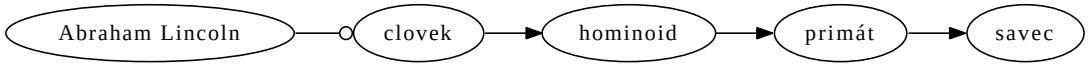
4.5 *Zvětšení příkladu AbeMammal*

Dosud jsme měli v interních axiomem informace o vzájemné inkluzi tříd člověk, hominoid, primát, savec a dokazovali jsme, že Abraham Lincoln je člověk. K tomu bylo zapotřebí najít v externím zdroji axiomů jediný fakt, a to ten, že je Abraham Lincoln člověk. Modelovaná situace je znázorněna na následujícím grafu:



Hrany ve tvaru šipky značí vzájemnou inkluzi. Hrany, které mají na konci kolečko značí relaci instance. Na obrázku jsou znázorněny jen vztahy známé ze zadání a z externího zdroje, nikoliv ty, které lze dále získat odvozováním.

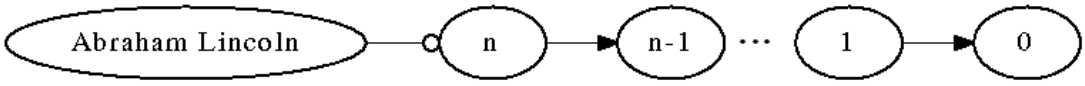
V tomto experimentu upravíme zadání tak, aby bylo nutné v externím zdroji najít více než jen jeden axiom. Za tímto účelem do externího zdroje dáme místo relace instance relaci „být podtřídou“. Pro jednoduchost vymažeme vztahy o tom co je instancí třídy třída:



Aby bylo možné odvodit, že instance jedné třídy jsou i instancemi její nadtřídy

i po tomto zjednodušení, je třeba přidat buď axiom $\forall x : instance(x, trida)$ nebo z původního axiomu s implikací, že instance podtřídy je instance nadtřídy odebrat podmínku, že podtřída i nadtřída jsou instancí třídy. V tomto experimentu volíme druhý způsob.

Přesunutí vztahu „být podtřídou“ do externího zdroje nám umožní zobecnit úlohu *AbeMammal* na tvar:



Informace o tom, že Abraham Lincoln, je instancí třídy n je nyní interním axiomem. Zadáním úlohy je dokázat, že je také i instancí třídy 0. Externí zdroj bude obsahovat informace, že pro každé i z $1..n$ je i podtřídou $i-1$ a právě tyto axiomy z něj budou potřebovat dokazovače získat, aby mohli úlohu dokázat.

Tento experiment byl celkem s oběma dokazovači, i s LeanCoPem-XDB i se SPASSem-XDB, zopakován celkem 71 krát a s oběma pro každé n jednou. I v tomto experimentu LeanCoP-XDB vždy SPASS-XDB porazil. Výsledné časy jsou znázorněny v grafu 4.6.

Je přirozené, že se stěžováním problému roste čas, který dokazovače na úloze strávily. Z výpisů, které lze ze SPASSu-XDB získat, lze vyčíst, že tento dokazovač postupoval takto: Během relativně krátké doby si stáhl z externího zdroje axiomů všechny relevantní axiomy (to, zda stáhne i nepotřebné axiomy, nebylo testováno). Poté SPASS-XDB strávil většinu času odvozováním z těchto axiomů. Kolísání hodnot, které je z obrázku patrné nebylo způsobeno chybou měření, ale chováním SPASSu-XDB. Co ho způsobilo však nebylo zjišťováno.

LeanCoP-XDB zvolil jinou taktiku. Například pro $n = 10$ jako první vygeneroval schématickou odpověď:

```
subclass(10, 0)
```

Poté ji zkusil doplnit použitím externího zdroje a odesláním dotazu, zda externí zdroj obsahuje informaci, že 10 je podtřídou 0.

Jako druhou v pořadí LeanCoP-XDB vygeneroval schématickou odpověď:

```
subclass(V0, 0)
subclass(10, V0)
```

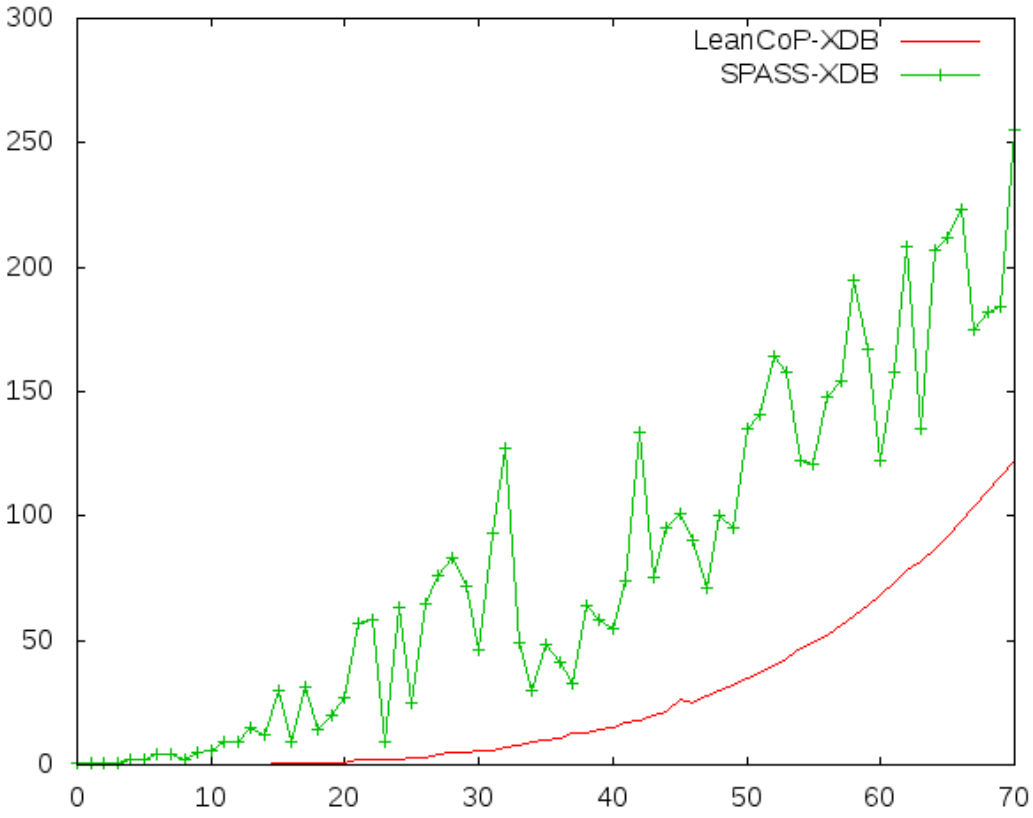
`V0` je proměnná, kterou je potřeba doplnit údaji z externího zdroje. Komponenta LeanCoPu-XDB, která je zodpovědná za doplňování schématických odpovědí, se pak externího zdroje zeptala, jaké zná třídy, které obsahují třídu 0 jako svou podtřídu. Pro každou odpověď, která by dorazila, se pak ptal, zda třída 10 obsahuje tuto podtřídu.

Dále dorazila schématická odpověď:

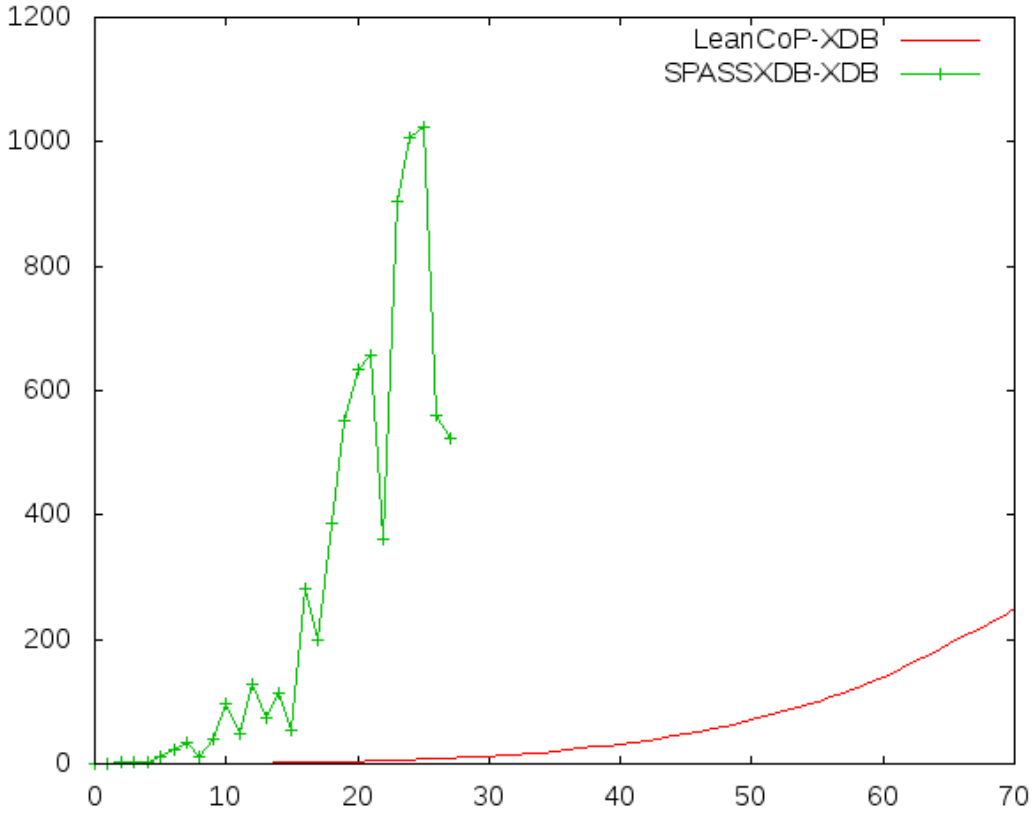
```
subclass(V0, 0)
subclass(V1, V0)
subclass(10, V1)
```

Ta obsahuje již dvě volné proměnné, které je třeba doplnit a znázorňuje možnost, že by třída 10 byla podtřídou třídy `V1`, která by byla podtřídou třídy `V0` a až ta by byla podtřídou třídy 0.

LeanCoP-XDB dále postupuje prodlužováním této schématické odpovědi až nalezne odpověď, že 10 je podtřídou třídy 0 přes dalších 9 mezitříd. Ta pak jde doplnit daty z externího zdroje na důkaz. Tento opatrný způsob vede k tomu, že



Obrázek 4.6: *Zvětšení příkladu AbeMammal*



Obrázek 4.7: *Ještě více ztížená úloha AbeMammal*

z externího zdroje není staženo zbytečně mnoho dat v případě, že by jich tam tolik bylo. Zároveň nalezne nejjednodušší možný důkaz, pokud by jich bylo možných více.

Iterativní prohlubování prohledávání LeanCoPu bohužel vede k tomu, že se schématické odpovědi generují opakovaně. Pokud jsme již vygenerovali všechny schématické odpovědi do nějaké hloubky, tak spolu s vygenerováním všech schématických odpovědí nejvýše o 1 složitějších, se vygenerují všechny dřívější schématické odpovědi znovu. Z tohoto důvodu LeanCoP-XDB řeší tuto úlohu s časovou složitostí $O(n^2)$. Tomu odpovídá i tvar křivky znázorňující čas běhu.

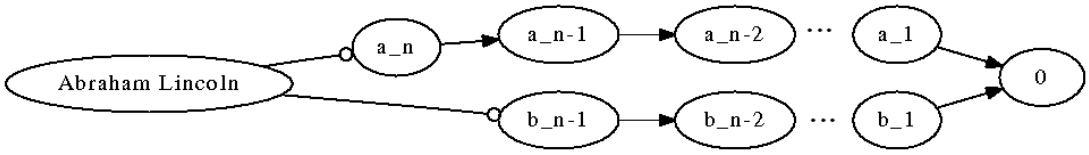
4.6 *Ještě více ztížená úloha AbeMammal*

Další možností, jak zadání úlohy *AbeMammal* ještě více zobecnit, by bylo nepředpokládat posloupnost po sobě jdoucích tříd inkluzí vnořených jedné do druhé, ale uvažovat nějaké obecnější uspořádání. Takto obecná úloha by se ale obtížně testovala, a proto provedeme experiment, který třídy uspořádá trochu složitěji, ale jen jedním způsobem. Konkrétně budeme předpokládat, že třída 0 obsahuje podtřídy a_1 a b_1. Dále pak pro každé $i \in \{2, \ldots, n\}$ je třída a_i podtřídou třídy a_{i-1}. Pro každé $i \in \{2, \ldots, n-1\}$ je třída b_i podtřídou třídy b_{i-2}.

Dále do problému přidáme dva interní axiomy:

1. Abraham Lincoln je instancí třídy a_n.
2. Abraham Lincoln je instancí třídy b_{n-1}.

Situace je znázorněna obrázkem:



Jak je vidět, to že je Abraham Lincoln instancí třídy 0, by mělo jít dokázat dvěma zcela odlišnými způsoby. Co to pro různá n udělá s oběma dokazovači je patrné z grafu 4.7.

Doba běhu SPASSu-XDB se se zvyšujícím se n rostla tak rychle, že byla z časových důvodů změřena pouze pro $n \leq 27$. Tvar křivky znázorňující dobu běhu LeanCoPu-XDB na první pohled vypadá podobně. Konkrétně se ale všechny časy zhruba zdvojnásobily. Hlavní příčinou tohoto zdvojnásobení ale kupodivu není dvojnásobné množství tříd a cest skrze ně, které je potřeba prohledat. To zdvojnásobuje pouze čas potřebný na komunikaci s externími zdroji, a to je jen malý zlomek z celkového času během, kterého LeanCoP-XDB řešil úlohu.

Většinu času totiž strávil LeanCoP-XDB čekáním na jádro dokazovače - LeanCoP. Právě v něm se čas zdvojnásobil. Stalo se tak proto, že kromě změny struktury dat v externím zdroji byl s tímto experimentem přidán jeden interní axiom. Ten zdvojnásobil prostor, který LeanCoP prohledával a zdvojnásobil také množství schématických odpovědí, které bylo potřeba ověřit.

Pro $n = 10$ LeanCoP-XDB vygeneroval dvě schématické odpovědi délky jedna:

```
subclass(a10, 0)
subclass(b9, 0)
```

Dále musel vygenerovat dvě schématické odpovědi délky dva:

```
subclass(V0, 0)
subclass(a10, V0)

subclass(V0, 0)
subclass(b9, V0)
```

Podobně postupoval dále, dokud nenašel schématickou odpověď, kterou bylo možné doplnit na důkaz. Díky tomuto postupu vedl nalezený důkaz vždy po spodní větvi obrázku přes $b_n \ldots b_1$. Podobně i SPASS-XDB v případech, kdy doběhl, našel většinou také tento kratší důkaz. Už tomu tak ale nebylo ve všech případech.

Záznamy ze všech těchto testů lze nalézt ve složkách `statistics/AbeMammal*/`, kde za `*` je potřeba doplnit vhodnou variantu úlohy. Výstupy ze SPASSu-XDB jsou v souborech s příponou `.spa`. Výstupy z LeanCoPu-XDB jsou v souborech `*.lea`.

4.7 Další příklady z webu SPASSu-XDB

V následujících experimentech porovnáme efektivitu LeanCoPu-XDB a SPASSu-XDB na zbylých příkladech, které lze stáhnout z webové stránky SPASSu-XDB. Experimenty však provedeme s přesným zadáním dle dané stránky. Už nebudeme zkoušet zadání upravovat, ani měnit velikost dat uložených v externím zdroji a ani jiné parametry. Příklady budeme nazývat jejich původními anglickými názvy.

4.7.1 Příklad *AlPacino*

Zadáním této úlohy je najít film, ve kterém hraje Al Pacino a který obsahuje ve svém názvu dvakrát slovo „Glen“, a následně ho vypsat do terminálu pomocí zdroje *PrintTTY*. Navíc jsou k řešení této úlohy k dispozici zdroje *LinkedMDB* a *RegExp* ke hledání filmů, ve kterých hraje Al Pacino a k porovnávání zjištěných názvů jako regulární výraz.

LeanCoP-XDB tuto úlohu vyřešil za 14,83 sekund, SPASS-XDB za 30,22 sekund.

4.7.2 Příklad *CloudyCapitals*

Zadáním této úlohy je najít dvě sousední evropské země, nad jejichž hlavními městy je stejná míra oblačnosti. K dispozici jsou externí zdroje axiomů *Mondial*, *capital_city*, *weather*.

Na této úloze je zajímavé, že její důkaz je pokaždé jiný v závislosti na počasí. LeanCoP-XDB tuto úlohu vyřešil za 9,95 sekund a odpověď, kterou nalezl říkala, že `The Hague`, hlavní město země `Netherlands`, má stejnou míru oblačnosti jako `Berlin` v zemi `Germany`. Oblačnost nad oběma místy údajně je `n/a`. Toto sice není odpověď, jakou by uživatel očekával, ale vzhledem k zadání a k datům v externích zdrojích je správná.

SPASS-XDB se při řešení této úlohy v době provádění tohoto měření ukončil chybou. To může být způsobeno tím, že ze zdroje *Weather* přicházejí někdy

termy ve tvaru `n/a` a v dalších tvarech, což může někdy způsobit, že dokazovač nezvládne parsování a ukončí se z důvodu syntaktické chyby na vstupu. Za jiného počasí SPASS-XDB tuto úlohu bezproblémově řešil, jen nebyla přesná doba běhu změřena. Bylo to ale okolo 20 sekund.

4.7.3 *CuriePrizes*

Zadáním této úlohy je najít lidi s příjmením Curie, kteří vyhráli nějakou cenu. K tomu jsou k dispozici axiomy z ontologie *YAGOSUMO*.

LeanCoP-XDB tuto úlohu vyřešil za 0,397 sekund. SPASS-XDB tuto úlohu měl za 4,11 vteřin.

4.7.4 *CapitalLevelMoscow*

Zadáním této úlohy je najít zemi, která je členem OECD[1], jejíž hlavní město má stejnou zeměpisnou šířku jako Moskva. K tomu je k dispozici *YAGOSUMO* pro zjištění, které země jsou členy OECD. Dále je k dispozici externí zdroj axiomů *capital_city*, pro hledání hlavních měst, *Xdb universal translator* pro překlad konstantních symbolů s názvy dle ontologie *YAGOSUMO* na konstantní symboly s anglickými názvy. Zdroj *look_different* dokazovače při této úloze musí použít ke zkontrolování, že nalezené město není Moskva. Pro hledání zeměpisných šířek je k dispozici zdroj *latlong*.

LeanCoP-XDB tuto úlohu řešil 4,494 sekund, SPASS-XDB 5,27 sekund.

4.7.5 *FloodingCopenhagen*

Tato úloha rozšiřuje předchozí úlohu o podmínku, že hledáme město, které by mohlo být zaplaveno. Interní axiomy pak dále popisují, jaká města by mohla být zaplavena - že by mohla být zaplavena města poblíž vodních ploch, že moře je vodní plochou a dále také axiom říkající, že přístavní města se nacházejí poblíž nějakého moře. LeanCoP-XDB tuto úlohu vyřešil 2,873 sekund, SPASS-XDB za 15,88 sekund.

Jedním z důvodů, proč řeší LeanCoP-XDB tuto úlohu je, že se neptá externích zdrojů na zbytečné otázky. Zjišťuje zeměpisnou šířku a další údaje pouze u měst, u kterých se potvrdilo, že jsou přístavním městem.

4.7.6 *ViennaTravel*

LeanCoP-XDB řešil tuto úlohu 1,729 sekund, SPASS-XDB 6,11 sekund.

4.8 Vlastní experimenty

Příklady z webu SPASS-XDB mají několik společných znaků. Všechny mají zdánlivě motivaci, ale o jejich užitečnosti by bylo možné polemizovat. Další společnou vlastností je, že jsou velmi jednoduché. Některé případy nevyžadují netriviální logická odvození a provádějí jen obdobu databázového spojení. Jindy, když už

[1] Organizace pro hospodářskou spolupráci a rozvoj

obsahují interní axiomy tak, aby bylo potřeba logické odvozování, tak tyto axiomy budí dojem, že jsou sepsány jen tak, aby úlohu komplikovaly a buď nepřidávají žádné zajímavé myšlenky nebo není příliš nutné je spojovat s externími axiomy (nezajímavými axiomy je zde myšleno zavádění definic a zesložitění úlohy tím, že se bude něčemu jen jinak říkat). Z tohoto důvodu bude provedeno experimentování ještě na dalších příkladech. Především je cílem otestovat LeanCoP-XDB a případně jeho efektivitu porovnat na úloze, která ke svému vyřešení bude potřebovat netriviální sadu axiomů a které povedou na použití externích zdrojů.

Pro zjednodušení popíšeme jeden příklad a otestujeme i jeho modifikace. Zvolíme následující axiomy:

1. Každý je svým potomkem.

   $\forall X : potomek(X, X)$

2. Dítě někoho z potomků B je také potomkem B.

   $\forall A, B : (potomek(A, B) <= (\exists X, Y : dite(A, X, Y) \& potomek(X, B))$

3. Dva lidé jsou příbuzní, pokud mají společného předka.

   $\forall A, B : pribuzni(A, B) <= (\exists [X] : potomek(A, X) \& potomek(B, X))$

K úloze přidáme jeden externí zdroj axiomů *Family tree* definující ternální predikátový symbol *dite*. Tento predikát popisuje, kdo je čí dítě. Třetím argumentem je *mother* nebo *father*, dle pohlaví rodiče. To se bude hodit v některých z modifikací následující úlohy. Data ve zdroji *Family tree* jsou dle rodokmene anglického krále Jakuba I. Stuarta z obrázku B na straně 69.

Například, že Jiří VI. byl otcem Alžběty II. je zaznamenáno axiomem:

$$dite('ElizabethII', 'GeorgeVI', father)$$

4.8.1 Úloha *Alžběta II. - František Bavorský*

V této úloze máme domněnku, že anglická královna Alžběta II. je příbuzná s Františkem, vévodou bavorským. $pribuzni('ElizabethII', 'Franzdukeof Bavaria')$

Z rodokmenu B je vidět, že oba dva jsou potomky Jakuba I. Stuarta a Anny Dánské. Přesto se ale jedná o vzdálenou příbuznost přes více než 10 generací a několik století.

SPASS-XDB se externího zdroje zeptal na následující dotazy:

1. Kdo jsou rodiče Františka, vévody bavorského?
2. Je František, vévoda bavorský dítě Alžběty II.?
3. Kdo jsou rodiče Alžběty II.?
4. Jaké má Alžběta II. děti?
5. Jaké děti má František, vévoda bavorský?
6. Je Alžběta II. dítě Františka, vévody bavorského?
7. Kdo je čí dítě?

Jako odpověď na tuto poslední a nejobecnější otázku přijde všech 77 záznamů v externím zdroji *Family tree*. Přestože už není možné z externího zdroje získat jakoukoli další informaci, tak SPASS-XDB dále odesílá otázky ve tvaru: „Je člověk A dítětem B?“, kde A i B jsou konkrétní lidé.

Výpočet SPASSu-XDB dále trvá více než 10 minut.

```
      Source: queued asked axioms time
         dite     44     44     81
SOURCES TOTAL     44     44     81
Wall-clock time elapsed:           10:58.91
```

LeanCoP-XDB, jako obvykle, nikdy neposlal takto obecný dotaz. Na druhou stranu odeslal mnohonásobně větší množství dotazů konkrétnějších. Vždy ale postupoval systematicky, dle schématických odpovědí. Ty lze u tohoto příkladu vnímat jako různé nápady, v jakém příbuzenském vztahu by mohla Alžběta II. s Františkem, vévodou bavorským být.

Jsou to následující možné vztahy v tomto pořadí:

1. rodič

   ```
   dite('Franz duke of Bavaria', 'Elizabeth II', V0)
   ```

2. dítě

   ```
   dite('Elizabeth II', 'Franz duke of Bavaria', V0)
   ```

3. sourozenec

   ```
   dite('Elizabeth II', V0, V1)
   dite('Franz duke of Bavaria', V0, V2)
   ```

4. prarodič

   ```
   dite('Franz duke of Bavaria', V0, V1)
   dite(V0, 'Elizabeth II', V2)
   ```

5. teta nebo strýc

   ```
   dite('Elizabeth II', V0, V1)
   dite('Franz duke of Bavaria', V2, V3)
   dite(V2, V0, V4)
   ```

6. vnouče

   ```
   dite('Elizabeth II', V0, V1)
   dite(V0, 'Franz duke of Bavaria', V2)
   ```

7. neteř nebo synovec

   ```
   dite('Elizabeth II', V0, V1)
   dite(V0, V2, V3)
   dite('Franz duke of Bavaria', V2, V4)
   ```

8. sestřenice nebo bratranec

```
dite('Elizabeth II', V0, V1)
dite(V0, V2, V3)
dite('Franz duke of Bavaria', V4, V5)
dite(V4, V2, V6)
```

Díky iterativnímu prohlubování hledá LeanCoP-XDB postupně vztahy od nejjednodušších ke složitějším. Přehled toho, co se dělo, lze vyčíst z následující tabulky:

Source:	queries	started	finished	axioms	cache	time/ms
dite	5442	5442	5442	5283	4802	1871
SOURCES TOTAL	5442	5442	5442	5283	4802	1871
term translator	6107	6107	6107	6107	0	1558
leancop core	1	1	0	168	638	5902
CPU time elapsed:						6340
Wall-clock time elapsed:						10866

Je vidět, že až 168. schématickou odpověď bylo možné doplnit na důkaz. K domu odeslal LeanCoP-XDB mnohonásobně více dotazů do externího zdroje než SPASS-XDB. To ale vůbec nevadí, jelikož tato komunikace trvala jen 1,8 sekundy. Navíc 4802 z 5442 dotazů by nemuselo být odesláno, pokud by bylo implementováno cacheování.

I přes tuto neúspornost v komunikaci s externími zdroji na této úloze LeanCoP-XDB překonává dobu běhu SPASSu-XDB zhruba 60ti násobně. Pokud by navíc zdroj *Family tree* obsahoval dostatečné množství dat, tak by pak i těch 5283 axiomů, které dorazili do LeanCoPu-XDB, bylo méně než počet axiomů, který by přicházel do SPASSu-XDB.

4.8.2 Další příklady s rodokmenem

Dále uvedeme už jen stručněji seznam příkladů, na kterých byl LeanCoP-XDB také testován a k nim dosažené výsledky a pro srovnání uvedeme jakých výsledků dosáhl SPASS-XDB.

Elizabeth obec kraj - Definujme relaci „možná příbuzní“ následovně: A je možná příbuzný s B, pokud je A příbuzný s někým, kdo má stejné příjmení jako B. Zadáním této úlohy je najít český okres, kde žije někdo, kdo je možná příbuzný s Alžbětou II. K řešení této úlohy je potřeba použít externí zdroje axiomů *Family tree*, *okres_obec*, *kraj_okres* a zdroj, který je určen speciálně pro tuto úlohu a pro zadanou přezdívku člověka najde jeho jméno a příjmení. Ten obsahuje jen jediný axiom a to, že František, vévoda Bavorský měl příjmení Prinze von Bayern. To, že tento zdroj obsahuje jen jediný axiom dokazovače ale nevědí.

SPASS-XDB u této úlohy stáhne do interní paměti všechny axiomy, které mu specifikace externího zdroje umožní stáhnout a to ho pak brzdí při dokazování. Při experimentu tuto úlohu nedokázal vyřešit ani během 6 hodin. Následně byl experiment přerušen.

LeanCoP-XDB naopak spotřeboval jen 10,89 sekund času procesoru. Celkem ale běžel 59,142 vteřin. Většinu svého času totiž strávil komunikací s externími zdroji. To ukazuje následující tabulka.

Source:	queries	started	finished	axioms	cache	time/ms
dite	86388	86388	86388	86546	84957	40098
prijmeni	275	275	275	1	224	5777
populace	1	1	1	29	0	1441
kraj_okres	1	1	1	1	0	25
okres_obec	1	1	1	1	0	27
SOURCES TOTAL	86666	86666	86666	86578	85181	47368
term translator	87416	87416	87416	87416	0	15213
leancop core	1	1	0	169	650	6990
CPU time elapsed:						10890
Wall-clock time elapsed:						59142

Z hodnot v této tabulce lze spočítat, že 98% dotazů by nemuselo být odesláno v případě, že by bylo implementováno cacheování.

Dvě ženy - Zadáním této úlohy je najít muže, který měl dvě ženy. Jelikož manželství nejsou součástí rodokmenu ve zdroji *Family tree*, je relace „mít ženu“ definována tak, že v tomto vztahu je muž se ženou s níž má společné dítě. K rozpoznání toho, že jsou dvě ženy různé, se používá externí zdroj *looks_different*.

LeanCoP-XDB řeší tuto úlohu za 1,703 vteřin. SPASSu-XDB to trvá 5,33 sekund. Přestože byl LeanCoP-XDB rychlejší, tak při řešení této úlohy mnohem více komunikoval s externími zdroji, a to dokonce více než desetinásobně.

Alespoň 3 děti - Zadáním této úlohy je najít otce tří dětí. K rozpoznání, že děti jsou různé, se používá zdroj *looks_different*.

Oba dokazovače k řešení používají zhruba stejné množství komunikace s externími zdroji. LeanCoP-XDB řeší tento příklad za 0,345 sekundy. SPASS-XDB až za 3,13 sekundy.

4.9 Aritmetika

Přestože hlavním cílem LeanCoPu-XDB není dokazování v aritmetických teoriích, tak by bylo možné k tomuto účelu LeanCoP-XDB použít. V této sekci bude naznačeno, jak. Pro celočíselnou aritmetiku byla provedena i implementace a experimenty ukáží, jak je tento přístup efektivní.

Při dokazování s aritmetikou se může stát, že by se dokazovači hodilo použít:

- informaci, že je mezi nějakými dvěma čísly nějaká nerovnost.
- informaci, že nějaké číslo vznikne nějakou aritmetickou operací.
- informaci, že se nějaká čísla rovnají.

Ve všech těchto příkladech lze použít stejný přístup, jaký používáme v LeanCoPu-XDB pro dokazování s externími axiomy. Vždy lze nejprve předpokládat, že daná čísla splňují požadovanou nerovnost, nebo že lze aritmetické operace provést, tak aby byly splněny další podmínky, a až následně tyto předpoklady později ověřit.

Všechny tyto aritmetické operátory můžeme umístit do externího zdroje *Mathematica*. Pokaždé, když dokazovač bude potřebovat k důkazu použít daný aritmetický operátor, tak ho přidá do schématické odpovědi. Schématickou odpověď pak může LeanCoP-XDB vyhodnocovat svým obvyklým mechanismem.

Bylo by však neefektivní ve schématické odpovědi zkoušet doplňovat jeden atom po druhém. Pokud by externí zdroj uměl vracet všechny odpovědi, tak by tím dokonce bylo možné ztratit úplnost dokazovače při aktuální konfiguraci. Uvažujme například následující schématickou odpověď:

```
$less(A,B)
$less(5,3)
```

Pokud budeme uvažovat, že by LeanCoP-XDB doplňoval jen jeden atom schématické odpovědi za druhým, tak by postupoval takto: Našel by dvojici čísel A a B takovou, že $A < B$ a pak by se snažil ověřit, zda $5 < 3$, kde by neuspěl. Proto by pokračoval další dvojicí čísel A a B takovou, že $A < B$. Takto by se snažil postup opakovat až do vyčerpání všech takových dvojic A a B. Jelikož je jich ale nekonečno, tak by nikdy s doplňováním této schématické odpovědi neskončil. Pokud by ale bylo zadání dokazatelné například s použitím některé další odpovědi, tak by LeanCoP-XDB nebyl úplný.

Situaci je možné řešit dvěma způsoby:

1. Změnit konfiguraci LeanCoPu-XDB tak, aby bylo možné doplňovat více schématických odpovědí současně.

   Samotná tato změna není příliš náročná. Množství schématických odpovědí, které by ale mohly takto dokazovač zabrzdit však není nijak omezené. proto by bylo potřeba povolit doplňování neomezeného množství schématických odpovědí současně. Zároveň by však bylo vhodné nastavit politiku vyhodnocování a přepínání vláken, tak aby se nadějnějším schématickým odpovědím věnovalo více úsilí.

2. Změnit u aritmetiky a u predikátů, které by mohly vracet potenciálně nekonečné množství odpovědí pořadí vyhodnocování tak, aby zůstala úplnost zachována.

Implementován byl druhý způsob. K vyhodnocování aritmetiky totiž používáme externí zdroj *Mathematica* a ten jako jediný externí zdroj, umožňuje vyhodnocování dotazů na více atomů současně. Je tedy možné mu předat všechny atomy ze schématické odpovědi, na které dokáže odpovídat současně a on najde vhodnou substituci proměnných, tak aby byly všechny atomy pravdivé. Námi uvedený příklad tedy LeanCoP-XDB řeší tak, že celou schématickou odpověď předá softwaru *Mathematica*, který běží na Q&A serveru v Miami a ten vyhodnotí oba atomy současně a odpoví, že nezná vhodnou substituci proměnných.

Tento přístup byl testován a porovnáván na aritmetických problémech z TPTP Problem Library [16]. Pro porovnání se SPASSem-XDB byly vytříděny a odebrány problémy, které SPASS-XDB nedokázal vyřešit do 10 sekund. Jelikož má SPASS-XDB vestavěnou podporu pro aritmetiku, byly zbylé problémy ještě rozděleny na dvě skupiny na problémy, které SPASS-XDB dokáže vyřešit bez použití externího zdroje *Mathematica* a na ty, kterým je *Mathematica* potřeba.

Jelikož není přesně specifikovaný protokol pro pokládání dotazů pro více atomové dotazy a pro tvar jejich odpovědí, tak měl LeanCoP-XDB potíže při parsování odpovědí. Ne vždy totiž termy, které přišly zpět byli čitelné prologem, alespoň ne při běžné definici operátorů. Proto byly problematické úlohy také odebrány a omezíme se především na celočíselnou aritmetiku. S tím, že je tento princip použitelný i pro zlomky nebo pro reálná čísla, jen by bylo potřeba dopsat parser odpovědí, které přicházejí z externího zdroje. Proto jsou podporovány zatím jen částečně.

Takto vybraným problémům, které SPASS-XDB zvládl bez použití zdroje *Mathematica* budeme říkat *easy*. Těm, ke kterým je potřeboval *Mathematicu* budeme říkat *easymath*.

4.9.1 Příklady *easy*

Testy byly provedeny na 33 zadáních ze skupiny *easy*. Ty testují jednoduché aritmetické nerovnosti s celými čísly a s výsledky operací s nimi.

Když byl LeanCoP-XDB testován, dostal na vstupu také specifikaci externích zdroje *Mathematica* spolu s axiomy vyjadřujícími vztahy mezi nerovnostmi a negací:

$$a < b \Leftrightarrow \neg(a \geq b)$$

$$a \leq b \Leftrightarrow \neg(a > b)$$

To LeanCoP-XDB potřebuje proto, že není možné posílat jako dotaz do externího zdroje negativní literál. LeanCoP-XDB by tedy neuměl v externím zdroji zjistit například informaci že $\neg(2 < 1)$. Díky výše zmíněným axiomům si ale dokáže LeanCoP-XDB odvodit, že k dokázání $\neg(2 < 1)$ stačí externím zdrojem ověřit, že $1 < 2$.

Zároveň ale přidání těchto axiomů způsobí, že LeanCop-XDB se bude snažit dokázat spor nalezením a a b takových, že platí buď

$$a < b \wedge a \geq b$$

nebo

$$a \leq \wedge a > b$$

.

Pokud toto budou nejednodušší řešení dané úlohy, tak je LeanCoP-XDB bude ověřovat jako první. Což je očekávané chování, protože chceme hledat nejsnažší řešení jako první. V tomto případě to ale LeanCoP-XDB brzdí.

Zadaných 33 úloh LeanCoP-XDB řešil celkem 48 sekund, během kterých strávil 36 sekund blokován čekáním na *Mathematicu*. Na jedné úloze tedy strávil v průměru 1,45 vteřiny. SPASS-XDB byl díky interním pravidlům pro matematické operace rychlejší. V případě, že neměl na vstupu ani specifikace externích axiomů tak prováděl v podstatě jen vyhodnocování výrazů. To mu zabralo 3,45 sekund. Tedy byl dle očekávání rychlejší než LeanCoP-XDB.

Z důvodu přesnějšího porovnání byl ale experiment ještě jednou zopakován, kdy dostal SPASS-XDB na svůj vstup také specifikaci externího zdroje *Mathematica* a axiomů, které říkají vztahy mezi nerovnostmi. V tomto exprimentu se SPASS-XDB nenechal zmást a žádnou otázku do externího zdroje neodeslal, protože si ji dokáže rychleji spočítat sám. Přesto ale prováděl nějaká logická odvození a mohl kromě výpočtu chvíli provádět inference i s axiomy o nerovnostech, což ho zdrželo. Za těchto podmínek SPASS-XDB úlohy řešil 36,7 vteřin.

SPASS-XDB řeší tedy při obou variantách spuštění tyto problémy rychleji než LeanCoP-XDB. Ten je však pomalý především kvůli době odezvy od *Mathematica*. Jak velkého zrychlení by bylo možné dosáhnout, kdyby LeanCoP-XDB mohl výpočet provádět také lokálně nebylo zjištěno bylo by to však nejvýše zrychlení o 36 sekund, dle doby blokování *Mathematicou*.

4.9.2 Příklady *easymath*

Ve skupině *easymath* bylo celkem 66 příkladů takových, že i SPASSXDB k jejich vyřešení potřebuje *Mathematicu*. Byla provedena měření, která ukazují rychlost obou dokazovačů na těchto příkladech.

LeanCoP-XDB vyřešil všech 61 příkladů dohromady 79,6 sekund, SPASS-XDB celkem za 279,1 sekund. Aby bylo možné odůvodnit, proč je LeanCoP-XDB rychlejší, bylo dále zkoumáno množství odeslaných otázek a přijatých odpovědí. LeanCoP-XDB odeslal celkem 65 dotazů a na ně obdržel 61 odpovědí. To bylo způsobeno tím, že u 2 úloh požádal 2 krát *Mathematicu* o najití dvojice čísel mezi, kterými by platili protichůdné nerovnosti. Ve zbylých 59 úlohách byl LeanCoP-XDB neomylný a vždy během prvního dotazu poslal do zdroje *Mathematica* správný dotaz o vyřešení celého příkladu najednou. Na tento dotaz pak získání jediné odpovědi, znamenalo úspěšné doplnění schématické odpovědi.

SPASS-XDB, přestože dokáže také odesílat víceliterálové otázky, tak ne vždy využil možnosti nechat si celou úlohu vyřešit zdrojem *Mathematica*. Odeslal totiž 328 otázek, na které obdržel jen 122 odpovědí.

Nejsložitější byla pro SPASS-XDB úloha:

$$\exists X : X * \frac{9}{12} = \frac{3}{7} \wedge \frac{1}{2} < X$$

Pro její vyřešení použil SPASS-XDB 15 dotazů do zdroje *Mathematica* na které přišli 4 odpovědi.

SPASS-XDB pokládal následující otázky:

1. Jaké existují zlomky menší než $\frac{1}{2}$?

2. Platí $\frac{1}{1} * \frac{9}{12} = \frac{3}{7}$?
3. Platí $\frac{1}{2} < \frac{1}{1}$?
4. Platí $\frac{1}{2} \geq \frac{1}{1}$?
5. Je $\frac{1}{2}$ celé číslo?
6. Je $\frac{1}{1}$ celé číslo?
7. Lze zlomek $\frac{1}{2}$ zkrátit na $\frac{1}{1}$?
8. Jaká existují čísla větší nebo rovná $\frac{1}{2}$?

Takto SPASS-XDB pokračoval se zdánlivě nahodilými otázkami než pak po 15 dotazech nalezl důkaz. LeanCoP-XDB jako obvykle poznal, že lze celé zadání vyřešit v *Mathematice* a celé ho tam odeslal a úlohu vyřešil jediným dotazem.

Podrobnější záznamy ze spuštění všech těchto *easy* a *easymath* problémů jsou součástí netištěných příloh.

5. Závěrečné shrnutí

Analýza z 2. kapitoly ukazuje, že některé aspekty protokolu tptp_qa jsou pro lean dokazovače nevýhodné. Především možnost externí zdroje univerzálně a existenčně kvantifikovat se ukázala nevhodná. Z analýzy je vidět, že univerzální kvantifikace pro lean dokazovače znamenají omezení možností paralelizace a volnosti v uspořádávání schématických odpovědí. Pro saturační dokazovač jako je SPASS-XDB tento prostředek naopak znamená nutnou a téměř jedinou možnost, jak se vyhnout zahlcení z velkého množství odpovědí a jak dokazovači zabránit posílat příliš obecné dotazy. Experimenty ukazují, že k dosažení podobného efektu lean dokazovače tento prostředek nepotřebují, pokud použijí myšlenku schématických důkazů, tak jak bylo v této práci navrhnuto. Na příkladech *AbeMammmal* nebo *Alžběta II. - František Bavorský* je dobře vidět, že aniž by byl externí zdroj univerzálně kvantifikován, tak LeanCoP-XDB odesílá přirozeně nejprve více specifické otázky, než začne hledat složitější důkazy. Na příkladu *AbeMammal a k němu nepotřebné axiomy navíc* je vidět, že použití univerzálně kvantifikovaného zdroje dokonce může LeanCoP-XDB zpomalit i bez použití paralelizace viz obrázek 4.5 na straně 45.

Provedení úprav protokolu tptp_qa tak, aby k těmto problémům nedocházelo, by mělo být jednou z budoucích prací na tomto projektu. Změnou komunikačního protokolu by se dal LeanCoP-XDB dále ještě více zrychlit. Bylo by třeba zohlednění aspekty z analýzy ve 2. kapitole na straně 24 a přidat následně navrhované informace do komunikačního protokolu.

Další možnosti, jak LeanCoP-XDB urychlit, se objevili během experimentů. LeanCoP-XDB zahazuje schématické odpovědi, které jádro dokazovače vygeneruje v případě, že se opakují. Bylo by ale možné zahazovat nejen opakující se schématické odpovědi, ale i permutace odpovědí, které již dorazily. Toho by bylo možné snadno dosáhnout setříděním atomů schématické odpovědi před tím, než se z ní udělá hash, kterým se kontrolují v hashovací tabulce duplicity.

Další možností, jak LeanCoP-XDB urychlit je neignorovat jen permutace dříve ověřených schématických odpovědí, ale zahazovat i jejich nadmnožiny. Situace, kdy by to bylo výhodné, se objevovala v experimentu *AbeMammal* (v jeho první variantě bez modifikací). Situace je popsána na straně 42. Zde docházelo k tomu, že byl stejný fakt jak v interních axiomech, tak se také mohl nacházet v externím zdroji. Proto byla nalezena jednou schématická odpověď, která daný axiom obsahovala a jednou taková, která ho neobsahovala.

K takovémuto filtrování schématických odpovědí, které jsou nadmnožinou jiných, by bylo potřeba navrhnout vhodné datové struktury a zvážit, zda jejich režie vyváží možný užitek ze zrychlení. Situaci z příkladu *AbeMammal* by bylo možné řešit tak, že by se použití externího zdroje zvažovalo až ve chvíli, kdy by se daný atom nepovedlo dokázat pomocí interních axiomů. Případně by také bylo možné toho omezení ještě nějak upřesnit a zjemnit například podmínkou na to, jak dlouho se má dokazovač daný fakt snažit odvodit z interních axiomů, než se rozhodne použít externí zdroj. To by mohl být jeden z dalších budoucích cílů této práce.

Většina těchto problémů by byla částečně zmírněna a další úlohy by byly rychlejší, pokud by bylo implementováno cacheování odpovědí. Tím, že byly ca-

cheovány odesílané dotazy bylo zjištěno, jak velké množství otázek bylo odesláno zbytečně. Téměř všechny experimenty, které vedly na doplnění více než jedné schématické odpovědi, ukázaly, že by s použitím cache mohla být část komunikace s externími zdroji ušetřena. Největší rozdíl byl zaznamenán u úlohy *Elizabeth obec kraj*, kde dokazovač položil 86666 dotazů do externích zdrojů a z toho 85181 dotazů by nemusel pokládat, pokud by měl nachaeované odpovědi.

I přes tyto nedostatky, které by mohly být do budoucna vylepšeny LeanCoP-XDB, ukázal překvapivou výkonost a v téměř všech prováděných experimentech překonával current state of art dokazovač umožňující komunikaci s externími zdroji, SPASS-XDB.

Závěr

Tato práce ukázala, jak upravit lean dokazovače tak, aby mohly používat externí zdroje axiomů. Navržený přístup byl ukázkově implementován programem LeanCoP-XDB. Ten s externími zdroji axiomů komunikuje protokolem tptp_qa, což umožňuje vzájemné porovnání obou programů a otestování na stejných příkladech, což bylo provedeno ve 4. kapitole.

Analýza z 2. kapitoly ukazuje, že některé aspekty protokolu tptp_qa jsou pro lean dokazovače nevýhodné. Proto byly v rámci této práce předloženy návrhy, jak komunikační protokol upravit a jaké informace do něj přidat.

I přestože, LeanCoP-XDB nedosahuje maximální možné efektivity, jaké by mohl, tak ve vzájemném porovnání provedeném ve 4. kapitole dosahuje překvapivě dobrých výsledků. Porovnání s current state of art dokazovačem, umožňujícím používat externí zdroje, SPASSem-XDB, bylo provedeno na všech sedmi příkladech, které jsou dostupné na jeho vlastní webové stránce. Na prvním z těchto příkladů, *AbeMammal*, bylo ukázáno dopodrobna, jak se oba dokazovače chovají a jaký přístup k řešení použijí. Ukázalo se, že LeanCoP-XDB navržený v rámci této práce používá při komunikaci s externími zdroji méně obecné dotazy, což má pozitivní vliv na celkovou efektivitu.

Z důvodu otestování obou dokazovačů na méně triviálních úlohách byly v rámci této práce navrženy další experimenty na dalších 8 úlohách, které byly s touto prací připraveny. Z toho 3 z nich byly modifikací příkladu *AbeMammal*. Ten byl parametrizován tak, aby používal různě velký externí zdroj dle zadaného parametru. Experimenty ukázaly závislost, jak LeanCop-XDB, navržený v této práci, a SPASS-XDB reagují na změny množství axiomů v externím zdroji. Tyto experimenty ukázaly, že zvyšování množství axiomů má překvapivě málo zpomaluje LeanCoPu-XDB.

V experimentech byly použity 4 mediátory nově vytvořené v rámci této práce, které komunikují s 6 externími zdroji v této práci vytvořenými[1].

Přestože aritmetika nebyla hlavním cílem této práce, tak byl LeanCoP-XDB testován na 99 převzatých aritmetických úlohách. LeanCoP-XDB nemá interně žádná aritmetická inferenční pravidla, ale dokázal aritmetické úlohy řešit s použitím externího zdroje axiomů. V 4.9.2 bylo ukázáno, že středně obtížné úlohy, které nedokáže SPASS-XDB vyřešit jednoduchou inferencí, řeší dokonce LeanCoP-XDB za menší čas než SPASS-XDB, který má aritmetické inference integrované.

Další zajímavé poznatky, které byly zjištěny v této práci, jsou shrnuty v 5. kapitole.

[1]Externí zdroje axiomů, které nahrazují *YAGOSUMO* jsou sice fyzicky rozděleny na více dílu, ale započítány jsou jako jediný zdroj.

Literatura

[1] Česká republika. Zákon č. 314/2002 Sb. ze dne 13. června 2002 o stanovení obcí s pověřeným obecním úřadem a stanovení obcí s rozšířenou působností. *Sbírka zákonů České republiky*, (Částka 114):6630–6633, 2002.

[2] Bernhard Beckert and Joachim Posegga. leantap: Lean tableau-based deduction. *Journal of Automated Reasoning*, 15:339–358, 1995. 10.1007/BF00881804.

[3] W. Bibel. Automated Theorem Proving. *Artificial Intelligence. Vieweg, Braunschweig/Wiesbaden*, second edition, 1987.

[4] Douglas B. Lenat. Cyc: a large-scale investment in knowledge infrastructure. *Commun. ACM*, 38(11):33–38, November 1995.

[5] R. Letz and G. Stenz. Model elimination and connection tableau procedures. *In A. Robinson, A. Voronkov, eds., Hand- book of Automated Reasoning*, pages 2012–2114, Elsevier, Amsterdam, 2001.

[6] Carolyn E. Lipscomb. Medical subject headings (mesh). *Bull Med Libr Assoc.*, 2000. 88(3): 265–266.

[7] George A. Miller. Wordnet: A lexical database for english. *Communications of the ACM*, 38:39–41, 1995.

[8] Ian Niles and Adam Pease. Towards a standard upper ontology. In *Proceedings of the international conference on Formal Ontology in Information Systems - Volume 2001*, FOIS '01, pages 2–9, New York, NY, USA, 2001. ACM.

[9] Jens Otten. leancop 2.0 and ileancop 1.2 : High performance lean theorem proving in classical and intuitionistic logic (system descriptions). In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated Reasoning*, volume 5195 of *Lecture Notes in Computer Science*, pages 283–291. Springer Berlin / Heidelberg, 2008.

[10] Jens Otten. Restricting backtracking in connection calculi. *AI Commun.*, 23(2-3):159–182, April 2010.

[11] Jens Otten and Wolfgang Bibel. leancop: lean connection-based theorem proving. *Journal of Symbolic Computation*, 36(1–2):139 – 161, 2003. First Order Theorem Proving.

[12] Piotr Rudnicki. An overview of the mizar project. In *University of technology, Bastad*, pages 311–332, 1992.

[13] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A Core of Semantic Knowledge. In *16th international World Wide Web conference (WWW 2007)*, New York, NY, USA, 2007. ACM Press.

[14] Martin Suda, David Stanovský, and Geoff Sutcliffe. SPASS-XDB goes mathematical. Nebylo publikováno. Dostupné z internetu na `http://www.karlin.mff.cuni.cz/~stanovsk/math/spassmath.pdf`.

[15] Martin Suda, Geoff Sutcliffe, Patrick Wischnewski, Manuel Lamotte-Schubert, and Gerard De Melo. External sources of axioms in automated theorem proving. In *Proceedings of the 32nd annual German conference on Advances in artificial intelligence*, KI'09, pages 281–288, Berlin, Heidelberg, 2009. Springer-Verlag.

[16] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[17] Geoff Sutcliffe, Martin Suda, Alexandra Teyssandier, Nelson Dellis, and Gerard de Melo. Progress towards effective automated reasoning with world knowledge, 2010.

[18] Ministerstvo vnitra České republiky. Četnost jmen a příjmení. 2011. `http://www.mvcr.cz/clanek/cetnost-jmen-a-prijmeni-722752.aspx`.

[19] Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. Spass version 3.5. In RenateA. Schmidt, editor, *Automated Deduction – CADE-22*, volume 5663 of *Lecture Notes in Computer Science*, pages 140–145. Springer Berlin Heidelberg, 2009.

[20] V Švejdar. Logika: neúplnost, složitost a nutnost. *Academia, Praha*, 2002.

A. Uživatelská dokumentace

A.1 Požadavky na systém

Program byl vytvořen pro platformu Unix a na ní byl i textován. Pro svou funkčnost předpokládá použitelnost běžných unixových utilit. Speciálně je potřeba bash, sed, tee, read a grep. Dále je potřeba mít nainstalovaný SWI-prolog a Javu. Je potřeba mít správně nastavenou systémovou proměnnou `PATH`, tak aby oba tyto programy bylo možné spustit z libovolného pracovního adresáře pomocí příkazů `swipl` a `java`.

LeanCoP-XDB byl úspěšně testován[1] na počítačích s následujícími konfiguracemi. Vyvíjen byl na prvním z nich:

Jádro Linuxu	**Distribuce a vydání**	**verze SWI-prologu**	**verze Javy**
2.6.32-37-generic	Ubuntu 10.04.3 LTS	5.8.0 for i386	1.6.0_20
2.6.24-27-server	Ubuntu 8.04.4 LTS	5.6.47 for i386	1.6.0_18
3.0.35	Gentoo 2.1	6.1.7 for i686-linux	1.6.0_31
2.6.36	openSUSE 11.3 (x86_64)	6.0.2 for x86_64-linux	1.6.0_18

Jelikož, ale ani program v Javě, ani kódy v prologu nebo v bashi nepoužívají žádné proprietární funkce, lze očekávat, že bude vše funkční i na počítačích s rozdílnými konfiguracemi.

Jediným důvodem, proč je vyžadován SWI-prolog je ten, že byl SWI-prolog vyžadován již LeanCoPem, který jsme v této práci upravovali. Upravovali jsme totiž LeanCoP verze 2.1 a ten z důvodu rychlosti existuje pro různé implementace prologu vícekrát. Jiné varianty LeanCoPu by šli upravit obdobným způsobem, jako bylo popsáno v 2. kapitole. To nebylo implementováno.

A.2 Seznam optionů

Chování LeanCoPu-XDB lze ovlivňovat pomocí několika optionů. Způsob doplňování schématických odpovědí byl implementován jen jediný. Lze ale měnit množství výpisů, které program během výpočtů píše.

Dokazovač při standardním nastavení vypisuje jen `proved`, pokud se důkaz povede nalézt.

[1] Dále byl LeanCoP-XDB testován na Windows 7 a spouštěn přes cygwin. Zde však docházelo ke komplikacím. LeanCoP-XDB úspěšně překládal cesty ve specifikacích externích zdrojů mezi linuxovým zápisem a zápisem pro Windows. Dále ale mediátory mohly spouštět další programy, kde už cesty nebyly vhodně přeložené, což způsobuje varování. Mimo to na Windows neplatí obvyklá poučka Unixu, že se skončením parent procesu jsou ukončeny i všechny child processy. Z tohoto důvodu se může stát, že na Windows zůstane některý pomocný proces například z nějakého mediátoru běžet i po najití důkazu.

Přestože by tedy bylo hypoteticky možné spustit LeanCoP-XDB na Windows, tak to není doporučeno a mediátory, které příklady z ukázek používají, na to nejsou připraveny.

Funkčnost pod Windows však nebyla cílem této práce. V komunitě automatického dokazování je standardem Unix a právě pro něj je práce zamýšlena.

`-p` (jako proof)
: Tímto optionem lze zapnout, aby program při svém úspěšném ukončení navíc vypsal doplněný schématický důkaz.

`-s` (jako statistics)
: Takto lze při úspěšném ukončení nechat vypsat statistiky použití externích zdrojů. Ty vypadají podobně jako tabulky použité v textu práce.

`-n` (jako new)
: Ukáže nové schématické důkazy. Jsou vypsány, tak jak je LeanCoP generuje po setřídění. Opakované výskyty nejsou zobrazovány.

`-v` (jako verbose)
: Nastavuje, aby byla vypisována veškerá komunikace s externími zdroji a navíc nové schématické důkazy. Toto nastavení tedy vypisuje:

  1. Nové schématické důkazy, tak jako při použití `-n`.
  2. Odesílané otázky.
  3. Přijímané odpovědi.
  4. Všechny další informace, které externí zdroje posílají formou komentářů.[2]

`-q` (jako quiet)
: Toto nastavení ještě více ztišuje standardní nastavení. To totiž může vypisovat varování například v případě, že specifikace externího zdroje obsahuje dodatečné informace, které LeanCoP-XDB neumí zpracovat. Touto volbou se vypne zobrazování těchto varování.

`-g`
: Při standardním nastavení LeanCoP-XDB si při spuštění pouze načte specifikace externích zdrojů axiomů. Spojení do daného zdroje se otevírá až ve chvíli, kdy je pro něj vygenerovaná nějaká otázka. Volba -g toto chování mění tak, aby do všech dostupných zdrojů byla spojení otevřena okamžitě.
  Jelikož LeanCoP současně pracuje na generování schématických odpovědí je možné, že by použití tohoto optionu běh programu v některých případech zrychlilo.
  V experimentech tato volba nebyla použita.

`-EPrint=1`
: Zobrazuje důkaz a statistiky. Je ekvivalentem k `-ps`.

`-EPrint=2`
: Jako `-EPrint=1` a navíc vypisuje informaci o tom, co program dělá. Konkrétně je vypsána informace, že je program blokován čekáním na novou schématickou odpověď a nebo informace o opaku - tedy že má program, co dělat a že pracuje na doplňování schématické odpovědi.

`-EPrint=3`
: Jako `-EPrint=2`, navíc vypisuje schématické odpovědi jako při `-n`.

`-EPrint=4`
: Jako `-EPrint=3`, navíc vypisuje všechny externí komunikace jako při použití `-v`.

[2]Například Q&A server, přes který se většina použitých příkladů spojuje s externími zdroji, vypisuje prosbu o sponzorskou dotaci s varováním, že jinak bude přidáváno náhodné zpoždění. Všechny takovéto informace jsou při standardním nastavení filtrovány a ignorovány.

A.3 Spuštění LeanCoPu-XDB

Před spuštěním LeanCoPu-XDB je třeba zkopírovat program, mediátory a případně externí zdroje z přiloženého CD do počítače nebo případně rozbalit z archivu.

Vše je nastaveno tak, aby při zachování adresářové struktury, nebylo potřeba nic nastavovat. LeanCoP-XDB, mediátory a všechny další přiložené programy tuto strukturu adresářů očekávají. Před spuštění je potřeba zajistit, aby u všech souborů byla nastavena práva, a to jak právo pro spuštění ke všem spustitelným programům a skriptům, tak práva pro čtení a zápis. Práva pro zápis jsou potřeba speciálně do složky logs, kde jsou zaznamenávány komunikace se všemi externími programy (mediátory a zdroji).

Dále LeanCoP-XDB a všechny pomocné programy, skripty a mediátory očekávají, že bude při spouštění nastaven pracovní adresář (working directory) do kořene této adresářové struktury.

LeanCoP-XDB se spouští příkazy:

`cd` *příslušný adresář*

`./start.sh` *[optiony] vstupní_soubor.tptp [nepovinně další soubory]*

Například pro spuštění úlohy *Sobota*, popisované v textu práce, změření času, jak dlouho ji bude LeanCoP-XDB řešit a pro vypsání Obce, která je řešením, lze program spustit takto:

```
./start.sh -ps problems/myxdb/Sobota.p
```

Nebo takto:

```
./start.sh -p -s problems/myxdb/Sobota.p
```

Na tom, zda se píšou jednopísmenkové optiony dohromady (`-ps`) nebo odděleně (`-p -s`), nezáleží.

Volba `-p` zajišťuje, že program vypíše doplněnou schématickou odpověď. Ta by měla vypadat jako v ukázce 2.2 na straně 21. Současně ale budou všechny časy měřeny a dalším výstupem bude statistika jako v ukázce 4.1 na straně 37.

`./start.sh -np problems/myxdb/elizabeth.tptp` Například tento příkaz spouští LeanCoP-XDB na úloze *Alžběta II. - František Bavorský*. Volba `-n` říká, že mají být vypisovány tvary schématických odpovědí, které jsou doplňovány. Tak byl vygenerován seznam různých příbuzenských vztahů ze strany 53. Tam byly vztahy přes jednu generaci (rodič, dítě, sourozenci) a vztahy přes dvě generace (prarodiče, teta/strýc, neteř, vnouče, sestřenice a bratranec). Spuštěním tohoto příkazu lze nechat program pokračovat ve výpisech příbuzenských vztahů i přes více generací. Takto pokračuje, dokud nenajde schématickou odpověď, kterou se mu povede doplnit na důkaz. Volba `-p` zajistí, že je pak následně vypsána doplněná schématická odpověď. Ta se skládá z 24 atomů s funktorem *dite*, které ukazují, že je Alžběta II. příbuzná s Františkem, vévodou bavorským. Z důvodu své délky není v textu práce uvedena. Čtenář si ji však může vygenerovat sám spuštěním tohoto příkazu.

Všechny příklady zmíněné v textu této práce lze nalézt ve složce problems. Při jejich spuštění je však třeba dbát na výběr správné varianty daného příkladu, pokud se tam vyskytuje ve více variantách. K příkladům, které byly testovány pro různě velké externí zdroje axiomů jsou přiloženy skripty pro vygenerování zdroje zadané velikosti.

A.4 Příprava vlastních příkladů

Nová zadání pro LeanCoP-XDB si uživatel může vyrobit sám. Problémy je třeba zapisovat v jazyce TPTP viz. [16]. Do zadání problému je pak možné připsat specifikaci externího zdroje tak, jak je popsáno v sekci C. Je možné umístit specifikace všech externích zdrojů do samostatného souboru, tím je oddělit od zadání problémů a pak využít toho, že LeanCoP-XDB může číst více vstupních souborů, pokud budou v parametrech příkazové řádky zadány.

Uživatel může pro tento účel použít i soubory, které byly pro tento účel připraveny `externals.ax` a `mathematica.ax`. V souboru `mathematica.ax` je specifikováno, jak se spojit s externím zdrojem *Mathematica* a s jakými predikátovými symboly lze dotazy pokládat. V souboru `externals.ax` jsou specifikace dalších zdrojů vyjmenovaných ve 3. kapitole.

Speciálně si tedy uživatel může bez nutnosti specifikovat externí zdroje napsat jen problém v jazyce TPTP a spustit ho pomocí následujícího příkazu.

`./start.sh` *vstup.tptp* `mathematica.ax externals.ax`

Tím říká LeanCoPu-XDB, že může k vyřešení zadání ze souboru *vstup.tptp* používat *Mathematicu* a další externí zdroje ze 3. kapitoly[3].

Dále, je možné si dle specifikace tptp_qa C naprogramovat vlastní mediátor a komunikovat s vlastnímími externími zdroji v libovolném formátu.

Pro velké množství experimentů lze použít i nějaký z mediátorů, které byli s LeanCoPem-XDB připraveny. Například lze použít *prolog mediator*. Uživatel tedy může sestavit zadání problému v jazyce TPTP a k němu přidat specifikaci říkající, že bude použit jako externí zdroj program v prologu, se kterým se bude komunikovat pomocí *prolog mediator*u.

Specifikace externího zdroje obsahují binární predikát p, s možností odpovídat na libovolně kvantifikované dotazy se napíše takto:

```
fof(predikat_p, external,
        ? [X,Y] : p(X,Y),
        external(
                exec,
                'mediators/prolog_mediator.sh soubor.pl',
                tptp_qa
        )
).
```

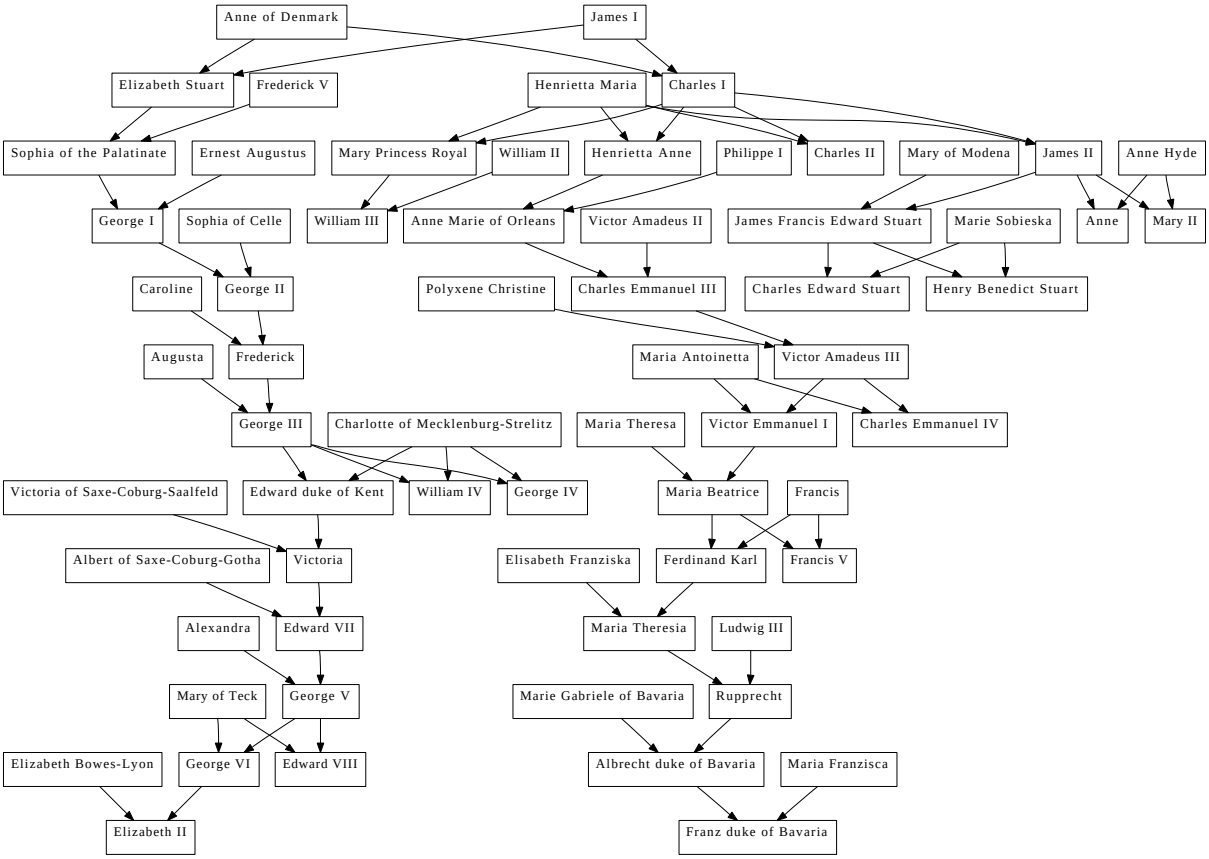
Do souboru s názvem `soubor.pl` si pak uživatel může umístit libovolná data zapsaná v prologu. Může je zapsat buď jako jednoduché fakty, např `p(a,b).` nebo může v prologu napsat složitější program, který platnost daných axiomů ověřuje.

Tato opatření už plně postačují k tomu, aby se LeanCoP spojil s uživatelem vytvořeným externím zdrojem v souboru `soubor.pl`. K ladění svých zadání může uživatel použít optiony, které zvyšují množství výpisů. Dále je možné komunikace s mediátory sledovat v příslušných souborech ve složce logs. Například při použití *prolog mediator*u, jako zde bylo ukázáno, lze vstupy a výstupy z tohoto mediátoru sledovat v souborech `logs/prolog_mediator.in` a `logs/prolog_mediator.out`.

[3]Pro lepší čitelnost textu mohly být některé predikátové symboly přejmenovány či zkráceny, proto je při použití `externals.ax` doporučeno do tohoto souboru nahlédnout a najít specifikaci daného zdroje, o kterém uživatel předpokládá, že bude použit, a zkontrolovat zda se predikátové symboly shodují.

B. Rodokmen anglického krále Jakuba I. Stuarta

V příklad *Alžběta II. - František Bavorský* ze strany 52 a v dalších menších příkladech byl použit zdroj *rodokmen*. Tento externí zdroj zachycuje příbuzenské vztahy potomků anglického krále Jakuba I. Stuarta. Ty jsou zaznamenány pomocí predikátu *dite*, který říká, kdo se komu narodil. Obsah dat v tomto zdroji je znázorněn na následujícím obrázku, který byl dle tohoto zdroje vygenerován:



C. Protokol tptp_qa

Dostupnost externích axiomů se specifikuje formulí v následujícím tvaru:

`fof(`*external_name*`, external,`
    allextemplate`,`
    `external(`*type*`,` *access*`,` *protocol*`,` *useful info for externals*`),`
    useful info for prover
`).`

external_name značí jméno externího zdroje axiomů. *allextemplate* obsahuje informaci o tom, jaké axiomy může zdroj poskytnout a případně, za které proměnné musí být v dotazu dosazeno. Nechť *L1* a *L2* jsou seznamy proměnných. Povoleny jsou následující tvary:

`!` *L1* `:` `?` *L2* `:` *atom*

respektive pokud je *L1* nebo *L2* prázdný, tak

`?` *L2* `:` *atom* nebo `!` *L1* `:` *atom*

respektive jen

atom

v případě, že je i *L2* prázdný. Syntaxe seznamů i proměnných je podobná jako v prologu. Seznam je uzavřený v hranatých závorkách a jednotlivé prvky se oddělují čárkou. Proměnná vždy začíná velkým písmenkem. Například toto je možný seznam proměnných: `[X,Y,Z,Alpha2]`

U definice běžných formulí znamená symbol `!` v jazyce TPTP univerzální kvantifikaci a symbol `?` existenční kvantifikaci. Význam symbolů `!` a `?` je tedy v této specifikaci mírně rozdílný. I přesto proměnným ze seznamu *L1* budeme říkat univerzálně kvantifikované proměnné a proměnným ze seznamu *L2* zase existenčně kvantifikované proměnné. V případě, že seznam *L1* obsahuje alespoň jednu proměnnou, říkáme, že daný zdroj je univerzálně kvantifikovaný. Celou specifikaci lze číst takto: Pro zadané hodnoty univerzálně kvantifikovaných proměnných je možné hledat hodnoty existenčně kvantifikovaných proměnných.

type je v protokolu rezervován pro budoucí použití. SPASS-XDB i LEANCOP-XDB zde zatím podporují jen `exec`, který znamená, že se se zdrojem bude komunikovat jako s procesem přes jeho standardní vstup a výstup. *access* závisí na *type* a obsahuje další informace k navázání spojení. V případě, že je *type* `exec`, *acceess* bude obsahovat příkaz, pomocí kterého se daný proces spustí. Za *protocol* se dosadí jméno protokolu, kterým se s daným zdrojem bude komunikovat. To je také rezervováno pro budoucí použití. SPASS-XDB i LEANCOP-XDB v současné době podporují jen formát TPTP question and answer, což se zde nastavuje jako `tptp_qa`, a takto budeme tento protokol zjednodušeně nazývat ve zbytku práce.

useful info for externals je seznam informací, který se má v nezměněném tvaru předávat externímu zdroji s každou otázkou. *useful info for prover* je seznam dodatečných informací o daném zdroji pro dokazovač. Dále probíhá komunikace dle zvoleného protokolu. Tedy zpravidla dle tptp_qa.

Žádosti o externí axiomy se posílají v tomto tvaru:

```
fof(request_name, question,
    extemplate,
    source,useful_info
).
```

request_name je jméno, které by mělo být pro každý dotaz do daného zdroje v rámci jednoho spojení jedinečné. *extemplate* je tvaru `? L :` *atom* resp. jen *atom*, kde L je seznam volných proměnných v *atom*u resp. *atom* neobsahuje žádné volné proměnné. *source* slouží k identifikaci toho, kdo otázku odesílá. Což může zdroj například použít k tomu, aby si vedl statistiky, jak často je kým využíván. *useful_info* je seznam dodatečných informací pro zdroj. Zde může být například uvedeno jaké prostředky má zdroj k odpovězení otázky použít. Případně zde může být zadané omezení na maximální počet axiomů, které se mají vrátit. SPASS-XDB i LEANCOP-XDB za tento seznam dosadí seznam, který je zadaný se specifikací daného zdroje.

Ten by měl při svém spuštění napsat:

```
% Service started
```

a dále na každý dotaz, který dokazovač pošle zareagovat odesláním následujícího balíku odpovědí:

```
% SZS status Success for source
% SZS answers start InstantiatedFormulae for source
fof(external_name,answer,
    atom,
    answer_to(request_name, [useful_info_1]),[useful_info_2]
).
fof(external_name,answer,
    atom,
    answer_to(request_name, [useful_info_1]),[useful_info_2]
).
...
% SZS answers end InstantiatedFormulae for source
```

source z odpovědi by se měl shodovat s *source* z dotazu. *external name* se nemusí shodovat s *external name* ze specifikace zdroje. *request_name* v odpovědi by se mělo shodovat s *request_name* z dotazu. V *useful_info_2* může zdroj dokazovači předat další dodatečné informace týkající se axiomu.

Při svém vypnutí externí zdroj axiomů vypíše

```
% Service over
```

C.1 Ukázka specifikace externího zdroje a komunikace s ním

Předpokládejme, že máme databázi CreatorDB, v ní tabulku s_creatorOf se sloupci Name a Thing a že v této tabulce jsou údaje o tom kdo, co stvořil. Dále předpokládejme, že máme program DBServer, který zpřístupňuje protokolem

tptp_qa jednotlivé řádky databáze, jejíž název dostal jako parametr. Dejme tomu, že nechceme zpřístupnit celou tabulku toho, kdo co stvořil, například z důvodu velikosti tabulky a že chceme dotazy do zdroje omezit jen na tvar: Pro zadou věc najdi její tvůrce. Pak by mohla specifikace externího zdroje vypadat například takto:

```
fof(creators, external
    ! [Thing] : ? [Name] : s__creatorOf(Name, Thing),
    external(exec, 'DBServer CreatorDB', tptp_qa,
        [xdb(limit,number(1000)),xdb(limit,cpu(4))]
    )
).
```

V seznamu `[xdb(limit,number(1000)),xdb(limit,cpu(4))` je zapsaná informace, že bychom chtěli, aby databázový server používal nejvýše 4 CPU a vracel na každý dotaz nejvýše 1000 axiomů jako odpověď. Co vše se může do tohoto seznamu uvést závisí na možnostech daného zdroje.

Poté, co dokazovač tuto specifikaci načte nebo potom, co se rozhodne, že tento zdroj axiomů bude chtít použít, tak ho spustí příkazem `DBServer CreatorDB`. Proces, který byl spuštěn zapíše na svůj standardní výstup `% Service started`. Tím dokazovači oznámí, že je připraven přijímat dotazy. Ten mu může nechat dát k ověření domněnku o tom, že daný člověk stvořil danou věc a nebo se ho ptát na tvůrce dané věci.

Zeptejme se tedy na tvůrce YAGO ontologie.

```
fof(otazka1, question,
    ? [Name] : s__creatorOf(Name,s__YAGOOntology),
    leancopxdb,[xdb(limit,number(1000)),xdb(limit,cpu(4))]
).
```

Po načtení této otázky zdroj odpoví:

```
% SZS status Success for leancopxdb
% SZS answers start InstantiatedFormulae for leancopxdb
fof(creators,answer,
    s__creatorOf(s__FabianMartinSuchanek,s__YAGOOntology),
    answer_to(otazka1, []),[]
fof(creators,answer,
    s__creatorOf(s__GjergjiKasneci,s__YAGOOntology),
    answer_to(otazka1, []),[]
).
% SZS answers end InstantiatedFormulae for leancopxdb
```

Takto dokazovač klade další otázky a externí zdroj axiomů odpovídá do doby, než se dokazovač rozhodne skončit. Pak uzavře ukončí vstup do externího zdroje. Ten, poté co zjistí, že přečetl celý standardní vstup vypíše

```
% Service over
```

a vzápětí se vypne.

D. Obsah přiloženého CD

.

Zde budou vyjmenovány netextové přílohy práce. Adresáře budou odlišeny symbolem / za svým názvem. Symbolem * budeme za názvem budeme označovat více souborů současně se společným prefixem dle textu před *.

`./` Všechny zde uvedené cesty začínají v kořenovém adresáři netextových příloh.

`start.sh` Skript pro spuštění LeanCoPu-XDB.

`LeanCopXDB/` Adresář ve kterém je v Javě psaná část LeanCoPu-XDB.

`LeanCopXDB/src/` Zdrojové kódy modulů psaných v Javě.

`doc/html/` Vygenerovaná programátorská dokumentace. Tato dokumentace není součástí tištěných příloh.

`doc/html/index.html` Otevřením tohoto souboru se vygenerovaná programátorská dokumentace zobrazí.

`leancop21-xdb/` V tomto adresáři je modifikovaná verze LeanCoPu, tak aby generovala schématické důkazy.

`leancop21-xdb/navod.txt` Zde je stručný návod, jak spustit jádro dokazovače, modifikovaný LeanCoP, bez spuštění LeanCoPu-XDB. To lze využít ke sledování komunikace mezi oběma programy.

`problems/` Zadání úloh pro LeanCoP-XDB a SPASS-XDB

`problems/xdb/` Zadání úloh převzatých z webu SPASSu-XDB a jejich modifikace provedené v rámci této práce.

`problems/myxdb/` Zadání úloh, které byly nově vytvořeny v rámci této práce.

`problems/myxdb/Sobota.p` Příklad *Sobota*.

`problems/myxdb/elizabeth.tptp` Úloha *Alžběta II. - František Bavorský*.

`problems/myxdb/elizabeth_obec_kraj.p` Úloha *Elizabeth obec kraj*.

`problems/myxdb/dve_zeny.tptp` Úloha *Dvě ženy*.

`problems/myxdb/alespon_tri_deti.tptp` Úloha *Alespoň 3 děti*.

`problems/xdb/AbeMammal.p`,
`problems/xdb/AlPacino.p`,
`problems/xdb/CapitalLevelMoscow.p`,
`problems/xdb/CloudyCapitals.p`,
`problems/xdb/CuriePrizes.p`,
`problems/xdb/FloodingCopenhagen.p`,
`problems/xdb/ViennaTravel.p` Příklady, stažené z webu SPASSu-XDB. Jména příkladů odpovídají jménům souborů.

`problems/xdb/AbeMammalVelky*` Takto začínají názvy zadání pro modifikace úlohy AbeMammal s nepotřebnými axiomy navíc. Zadání je zde ve více variantách z důvodů různých variant zadání testovaných v textu práce.

`problems/xdb/AbeMammalDlouhy/` V této složce lze nalézt potřebné soubory k realizaci experimentu 4.5 *Zvětšení příkladu AbeMammal* ze strany 46.

`problems/xdb/AbeMammalDvePresneCesty/` V této složce lze nalézt potřebné soubory k realizaci experimentu 4.6 *Ještě více ztížená úloha AbeMammal* ze strany 49.

`mediators/` V tomto adresáři jsou mediátory vytvořené v rámci této práce.

`mediators_spass/` V této složce jsou mediátory, které nejsou součástí této práce, ale jsou součástí SPASSu-XDB a jsou potřeba k funkčnosti příkladů z jeho webu.

`bin/` V této složce jsou všechny pomocné programy a skripty, které jsou používány mediátory a dalšími části celého projektu.

`bin/spass_filter*` Program pro zpracování výstupu SPASSu-XDB a jeho zdrojové kódy.

`logs/` Do této složky jsou ukládány záznamy ze všech komunikací mezi různými procesy v rámci celé aplikace.

`statistics/` V této složce jsou záznamy z jednotlivých experimentů. Jsou zde výstupy z obou dokazovačů. Především pro úlohy, které byli postupně zvětšovány tu je nejvíce záznamů - pro každou měřenou velikost jeden. Podle těchto dat byli generovány grafy v textu práce. V této složce jsou dále skripty, které slouží k těmto účelům a dále skripty pro zpracování a filtrování výsledků a výstupů dokazovačů.

`ExternalSoures/` V této složce jsou umístěny externí zdroje, které byli v rámci této práce připraveny.

`mathematica.ax`, `externals.ax` Již dříve popsané soubory se specifikacemi externích zdrojů axiomů.

Tento výpis souborů a složek není úplný. Součástí příloh jsou další soubory, které LeanCoP-XDB používá nebo, které byli při vytváření práce použity. Vyjmenovány byly pouze ty zajímavé z nich.