Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Výukový program pro předmět UIR

Plzeň, 2014 Tomáš Sládek

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 9. května 2014

Tomáš Sládek

Abstract

Making of a new learning materials

This bachelor thesis describes a plan of making a new materials and modification of that ones which are currently in use for teaching topics from AI field at University of West Bohemia. Further it describes and evaluates results of realization of this plan. The result is an extensive presentation for teaching programming language called Prolog and other supporting materials. All of that is attached to this thesis.

Abstrakt

Tvorba nových výukových materiálů

Tato práce popisuje plán pro vytvoření nových a úpravu stávajících výukových materiálů z oblasti umělé inteligence, užívaných na Západočeské univerzitě v Plzni. Dále popisuje a hodnotí výsledky uskutečnění tohoto plánu. Výsledkem práce je rozsáhlá prezentace pro výuku programovacího jazyka Prolog a další doplňující materiály. Vše je zahrnuto do příloh práce.

Obsah

Kapitola 1

Úvod

Předmět katedry informatiky *Umělá inteligence a rozpoznávání (KIV/UIR)* (dále jen UIR), určený pro studenty 3. ročníku bakalářského studia, se od akademického roku 2014/2015 rozdělí na dva předměty – původní předmět UIR a nový předmět *Úvod do znalostního inženýrství (KIV/UZI)* (dále jen UZI) pro studenty třetího ročníku bak. studia. Do jeho sylabu bude přesunuta část problematiky původně probíraná v UIR, která se týká znalostních systémů.

Aby byla zachována přiměřená obsáhlost obou předmětů, musí být pochopitelně také doplněny o nová témata (či probírání již obsažené tématiky prohloubeno). To si žádá také přípravu nových materiálů k výuce. Jedním z nově zavedených témat je i programování v Prologu. Tento programovací jazyk je užitečným nástrojem pro vývoj znalostních systémů, nicméně, v předmětu UIR a dalších předmětech katedry byl dosud probírán jen okrajově – spíše v tomto směru došlo k omezování.

Prolog měl být původně zařazen na uvolněné místo v sylabu předmětu UIR na konci semestru. Předmětem této práce tudíž měly být úpravy předmětu UIR, sestávající především z vytvoření výukových materiálů k Prologu a dále z menších úprav existujících materiálů (aktualizací, doplňků, či odstranění drobnějších nedostatků). Během zpracování byl však Prolog přeřazen do navazujícího předmětu UZI (kde má být následně i využit), tudíž se zadání poněkud rozpadlo na vytvoření materiálů k Prologu a s ním v podstatě nesouvisející úpravy materiálů pro UIR.

Hlavním cílem práce tedy zůstalo vytvoření materiálů k výuce Prologu v předmětu UZI, vedlejším cílem pak bylo nalezení vhodných úprav či rozšíření existujících materiálů k předmětu UIR a podle možností i jejich provedení.

Kapitola 2

Konkrétní cíle

2.1 Vytvoření výukových materiálů pro programovací jazyk Prolog

Hlavním předmětem práce bylo vytvoření podkladů pro výuku programovacího jazyka Prolog, která je naplánována do předmětu UZI, nově začleněného do učebních plánů. Základní znalosti programování v Prologu budou studenti dále potřebovat pro podrobné studium znalostních systémů v tomto předmětu.

Především bylo nutno vytvořit podklady pro prezentaci trvající celkem tři přednášky, z nichž každá je v předmětu UZI plánována na tři vyučovací hodiny. Celková doba vyhrazená pro výklad Prologu je tedy devět vyučovacích hodin, což je 405 minut.

Vzhledem k tomu, že do předmětu UZI budou převedeny některé již hotové materiály pro předmět UIR, grafická úprava prezentace by mělo pokud možno dodržet styl zavedený v předmětu UIR, aby nebylo nutno ostatní materiály předělávat. Dalším důvodem je také fakt, že přednášejícím předmětu UZI bude pan prof. Matoušek, který dosud přednášel předmět UIR a kterému zavedený styl vyhovuje.

Protože norma Prologu je poměrně slabá a u jednotlivých implementací běžně dochází k nestandardním rozšířením, bylo nanejvýš vhodné zvolit konkrétní implementaci, která bude vyučována a poté v předmětu používána. Pro použití v předmětu UZI byla zvolena implementace prologu B–Prolog. Tuto volbu jsem neprovedl já – na přípravě předmětu UZI se podílí více lidí – pouze jsem se podle ní řídil.

Vzhledem k tomu, že prezentace je koncipována jako úvod do logického programování a Prologu od úplných základů (nepředpokládá se předchozí znalost Prologu), většina získaných znalostí by studentům měla být užitečná, i pokud budou používat jiné implementace Prologu. Běžnější nestandardní rozšíření jednotlivé implementace také často sdílejí, takže mohou být navzájem zaměnitelné. V ostatních případech snad znalost řešení z B–Prologu studentům alespoň pomůže najít vhodnou alternativu, která je součástí dané implementace.

2.2 Navržené úpravy současných výukových materiálů

Jak již bylo řečeno v úvodu, druhotným cílem bylo provést úpravy existujících materiálů a samotného webu UIR v případech, kde by to bylo potřebné, což je víceméně pozůstatkem původně zamýšleného rozdělení témat mezi oběma předměty. S Prologem probíraným v UIR by cílem práce bylo kompletní přepracování předmětu UIR, zatímco předmětem UZI by se nijak nezabývala.

S novým rozvržením probíraných témat v sylabech je kompletní úprava předmětu UIR již nad rámec této práce (stejně jako předmětu UZI), tudíž jsem se omezil jen na zvážení několika konkrétních návrhů, které jsem zrealizoval, zavrhl, nebo pouze ponechal jako nápady či doporučení k posouzení tomu, kdo předmět UIR převezme.

2.2.1 Převedení obsahu textové povahy na text

V existujících prezentacích se občas objevují obrázky s textem namísto obyčejného textu – to může např. působit problémy při automatickém hledání a obecně to není vhodné, pokud má dokument vzbuzovat dojem kvality. Často jde pouze o krátké úseky, u nichž je použití obrázku omluvitelné (složitě formátované texty či různé grafické objekty spolu s textem, kde okolní text může sloužit jako dostatečná anotace pro vyhledávání).

V prezentaci zabývající se výrokovou a predikátovou logikou, která byla v nedávné době celá převedena ze starých nascanovaných materiálů, však

zůstaly dlouhé obrázkové pasáže. Zaměřil jsem se tedy na úpravu či spíše dokončení převodu této prezentace.

2.2.2 Zopakování teoretické informatiky na cvičení UIR

Navrhuji vyhradit jedno z úvodních cvičení předmětu UIR, nebo alespoň jeho část, zopakování znalostí z teoretické informatiky, na které UIR navazuje v některých později probíraných oblastech. Někteří si předmět *Teoretická informatika* mohou zapsat až po absolvování UIR, krátké zopakování by však mohlo prospět i těm, kteří teoretickou informatiku již absolvovali.

Na druhou stranu, základní datové struktury, opakované na prvních cvičeních, by měl každý znát z PPA2 a jiných předmětů. Proto by se část prostoru na cvičení mohla věnovat několika příkladům na konečné automaty a gramatiky. Jedná se o jednoduchou problematiku, kterou lze z několika ukázek rychle pochopit. Nejsou k tomu nutné širší znalosti z teoretické informatiky.

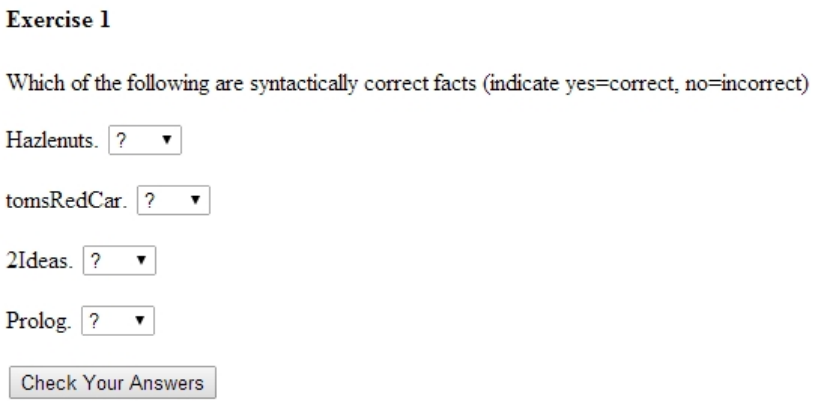
Pokud by se tímto nemělo zabývat na žádném cvičení, několik příkladů by alespoň mohlo být vystaveno na webových stránkách předmětu.

2.2.3 Kvízy pro domácí cvičení

Při vlastním studiu Prologu jsem po inspiraci z některých internetových výukových materiálů [12] navrhl vytvořit kvíz, který by řešiteli předkládal krátké příklady z Prologu a vyžadoval by doplnit např. výsledek nebo nějaký kus kódu pro správnou funkci programu (viz obr. 2.1).

Kvíz by měl být zaměřen především na záludnější problémy, kde nejsou všechna úskalí na první pohled zřejmá a člověk nemusí znát řešení, i když má pocit, že danou problematiku ovládá. Úplným začátečníkům by však měl nabídnout i velmi jednoduché otázky. V případě špatné (a patrně i správné) odpovědi by kvíz nabídl vysvětlení, tudíž by zároveň doplňoval chybějící znalosti.

Důvodem pro tento návrh je to, že ve snaze o oživení vlastních základních znalostí o Prologu (se kterým jsem se v minulosti setkal jen krátce) se mi podobně koncipované materiály jevily jako mnohem efektivnější, než snaha o pouhý pasivní příjem informací ze skript či prezentačních materiálů.



Obrázek 2.1: *Ukázka kvízu, který inspiroval tento návrh [12]*

Pokud by takový kvíz vznikl, nebyl by zřejmě důvod omezovat jeho rozsah pouze na Prolog. Proto jsem rozsah navrhovaného kvízu posléze rozšířil na všechny vyučované oblasti. Stejně tak by se nemusel omezovat pouze na předmět UIR nebo předmět UZI – podobný doplněk výuky by byl dost možná použitelný ve všech předmětech.

Za nejlepší způsob realizace považuji vytvoření webového systému, který by umožnil přidávat nové otázky a náhodně z nich sestavovat kvízy pro otestování znalostí z vybrané oblasti nebo oblastí. Také by mohl sledovat a porovnávat výsledky jednotlivých uživatelů. Úspěšné vyvolání soutěživosti může přispět k vylepšení znalostí studentů.

Systém s podobnou funkcionalitou již na ZČU existuje – je to systém TRIAL, fungující na KMA. Jednou z jeho základních funkcí je předkládat studentům cvičné matematické úlohy k vyřešení, přičemž k příkladu může nebo nemusí být poskytnuto i správné řešení – míra využití v jednotlivých předmětech KMA samozřejmě záleží na vyučujících.

Pokud bychom předpokládali, že máme potřebné zázemí, vytvoření základního systému bez požadavků na další funkce nevypadá jako zvlášť náročná záležitost. Lze však předpokládat, že společně s doladěním všech detailů a finálním nasazením by vytvoření systému i tak spotřebovalo nezanedbatelný podíl času, který je potřeba na splnění důležitějších cílů. Další problém pak představuje vytvoření obsahu, tedy dostatečné množiny kvízových otázek. Pokud chceme dostatečně kvalitní a pro výuku hodnotný obsah,

jeho vytvoření by patrně nebylo otázkou několika týdnů, možná ani měsíců. Navíc by systém zřejmě vyžadoval stálého správce – nejen kvůli doplňování obsahu, ale také pro řešení případných potíží – byla by jím tedy dlouhodobě zatížena i další osoba.

Z uvedených důvodů jsem tento bod zavrhl již ve fázi návrhu. Snaha o jeho realizaci by odčerpala příliš mnoho času a zabránila vytvoření kvalitních materiálů k výuce Prologu. I přesto bych systém vlastními silami pravděpodobně nedovedl do použitelné podoby. Návrh zde nicméně ponechávám uvedený pro případ, že by ho někdo jiný shledal zajímavým a rozhodl se ho později v nějaké podobě ujmout.

Kapitola 3

Plán obsahu přednášek z Prologu

V této kapitole je uveden přibližný návrh obsahu přednášek – Co by mělo být zahrnuto a kolik času by které oblasti mělo být věnováno. U každého tématu je zhruba popsáno, co konkrétně sem spadá a zda je něčemu přikládána menší či větší důležitost. Konkrétní časový odhad pro každou oblast je uveden na konci jejího popisu.

3.1 Úvod do deklarativního programování

Logické programování a programování omezujícími podmínkami, což jsou dva hlavní přístupy užívané v Prologu, patří mezi deklarativní paradigmata. Deklarativní programování představuje programovací paradigma značně odlišné od běžně užívaného imperativního paradigmatu a jeho subparadigmat (dnes typicky procedurální programování v kombinaci s objektovým paradigmatem).

Mnoho i poměrně zkušených programátorů nemusí být s tímto přístupem k programování vůbec seznámeno, nebo jim byl představen jen velmi povrchně. Z tohoto důvodu jsem usoudil, že bude vhodné začít obecným úvodem do deklarativního programování, zatím bez přímého vztahu k Prologu.

Časový odhad: 30 minut

3.2 Úvod do Prologu

Součástí úvodu do programovacího jazyka by zřejmě mělo být pár slov o jeho historii. Nepovažuji však za nutné věnovat této části mnoho času. Pokud hledáme něco, co je vhodné rozvést, mohou to být zajímavá fakta, která pomohou oživit přednášku.

V případě méně populárního programovacího jazyka, jakým je Prolog, považuji naopak za důležité věnovat se v úvodu osvětlení jeho významu a použitelnosti. Konkrétně v případě Prologu se lze poměrně často setkat s názorem, že Prolog se hodí pouze na hraní, případně k použití v laboratorních projektech s malým praktickým využitím. Cílem tohoto úvodu by tak mělo být vyvrátit nebo alespoň znejistit tento názor u studentů, kteří už o Prologu v takto negativním světle slyšeli. V případě těch, kteří se s Prologem seznamují poprvé, by měl posloužit jako motivace a případně snad zabránit přijetí podobného názoru v budoucnu, pokud se s Prologem sami nebudou nadále setkávat v praxi.

Časový odhad: 45 minut

3.3 Syntax Prologu a postup při hledání řešení

V úvodu této kapitoly by měly být zároveň zopakovány syntaktické prvky predikátové logiky, ze kterých vychází syntax Prologu. Uvedením ekvivalentního zápisu v Prologu potom může být objasněn původ a význam Prologovského syntaxu.

Pochopení původu používaných zápisů sice není nezbytné k použití Prologu jako programovacího jazyka, ale podle mého názoru může studentům poskytnout určitou "radost z objevování" a snad opět trochu zvýšit pozornost.

Dále už by mělo být přesně uvedeno, co všechno syntax Prologu umožňuje, z jakých prvků se Prologovský program může skládat a jaký je jejich význam. Postup řešení a možné záludnosti s ním spojené by se měly v podstatě prolínat i s dalšími částmi přednášek, kde budou probírány příkazy dostupné v B–Prologu. To proto, že s novými možnostmi přicházejí i nové problémy, o kterých je třeba vědět. Ve výsledku by toto téma mělo být před-

mětem podstatné části přednášek.

Časový odhad: 130 minut

3.4 B–Prolog

Protože k použití v předmětu UZI byl zvolen B–Prolog, který je volně k použití pro jednotlivce a libovolné nekomerční účely, krátká část přednášky by měla být věnována návodu, jak si B–Prolog opatřit, zprovoznit a použít interpret ke spouštění programů. Vzhledem k tomu, že stažení B–Prologu z oficiálních webových stránek a jeho instalace i použití jsou poměrně jednoduché, mělo by být plně dostačující věnovat tomuto tématu jen několik stran.

Časový odhad: 10 minut

3.5 Příkazy

Kromě základních příkazů by měly zahrnuty také nestandardní příkazy B–Prologu (tedy ty, které nejsou specifikovány ISO normou) – vzhledem k tomu, že Prolog není příliš standardizovaný a kompatibilita jednotlivých implementací je omezená, pokud chceme předvést větší část možností B–Prologu, nevyhneme se značně nestandardním příkazům.

V případě příkazů (respektive operátorů a vestavěných predikátů), u kterých není na první pohled jasné jejich chování nebo nemusejí být patrné všechny možnosti použití, by měl být uveden také jeden nebo více jednoduchých příkladů (obvykle nejvýše na několik řádků), které jejich využití objasní.

Celkový časový odhad: 130 minut

3.6 Rozbor příkladů

V souvislosti s probíranými příkazy a možnostmi Prologu obecně by měly být také rozebrány některé komplexnější příklady, které vše ukáží v širším kontextu a trochu více prakticky. Během přednášek by patrně měla být rozebrána spíše menší část příkladů, zatímco více by se jim měla věnovat cvičení. Čas přednášek je třeba věnovat zejména výše uvedeným záležitostem a cvičení pro toto skýtají lepší podmínky.

Zde uvedený časový odhad počítá pouze s časem přednášek. V případě cvičení se předpokládá, že rozebírání a řešení příkladů bude věnován v podstatě veškerý dostupný čas.

Celkový časový odhad: 60 minut

Kapitola 4

Praktické použití Prologu

Provedení vlastního průzkumu v "terénu" s věrohodnými výsledky by bylo náročné jak časově, tak pravděpodobně i finančně. Protože to v žádném případě není předmětem této práce, spolehl jsem se ohledně praktického využití Prologu na obsah odborných internetových diskusí [16][17][18]. Zdá se, že lidi zabývající se programováním lze rozdělit na tři skupiny.

První skupinu tvoří ti, kteří o Prologu nikdy neslyšeli nebo nemají dost dobrou představu, o co se jedná, aby se k jeho použití mohli vyjádřit. Druhá skupina Prolog zná, ale zastává názor, že jeho využití se omezuje jedině na akademickou půdu a k řešení praktických problémů se Prolog nehodí. Třetí skupina Prolog buď sama průmyslově využívá, nebo si je jeho využití alespoň vědoma.

Poměrnou velikost těchto skupin si na základě zběžného průzkumu netroufám odhadovat, ale nemyslím si, že by skupina zástupců Prologu byla zanedbatelně malá.

Využitelnosti Prologu v průmyslu nasvědčuje také fakt, že existuje řada pravidelně udržovaných komerčních implementací (Loga některých z nich jsou na obrázku 4.1). Většina z nich nabízí i verzi zdarma – Stejně je tomu u mnoha dalších komerčních produktů podobného charakteru.

Podle mého názoru můžeme předpokládat, že málokterá výdělečná organizace by byla ochotna vynakládat prostředky na vývoj softwaru, pro který na trhu není místo a tudíž jeho prodeje ani nepokryjí náklady. Zřejmě tedy musí existovat dostatek uživatelů Prologu, kteří jsou ochotni investovat nezanedbatelnou sumu do kvalitní implementace a tedy pravděpodobně z jejího užívání sami budou mít zisk.

Obrázek 4.1: *Logo SICStus Prologu, LPA WIN–Prologu, Visual Prologu a B–Prologu, které patří mezi komerční implementace.*

Převzato ze stránek: https://www.sics.se/projects/sicstus–prolog–leading–prolog–technology, http://www.lpa.co.uk/win.htm, http://www.visual–prolog.com, http://www.probp.com

Nakonec by ve prospěch Prologu mohl mluvit také fakt, že zatímco v minulosti byl výkon počítačů pro logické programování příliš nízký, dnes již výkon nepředstavuje problém a interprety prologu jsou efektivnější. Úvahami, zda Prologu patří budoucnost, se však již dostáváme zcela do oblasti spekulací.

Kapitola 5

Použité prostředky

Vzhledem k tomu, že všechny dosavadní prezentace pro předmět UIR jsou vytvořeny s použitím textového procesoru Microsoft Word, použil jsem tento program i při své práci. To by mělo zajistit, že dokument bude možno aktualizovat stejným způsobem jako již existující prezentace. Zároveň mi existující dokumenty posloužily jako šablona.

Při tvorbě vzhledu prezentace o Prologu jsem vycházel z prezentace zabývající se výrokovou a predikátovou logikou (téma v UIR dosud označované číslem 6), která byla dosud nejnovějším vytvořeným dokumentem. Soustředil jsem se přitom na udržení jednotného vzhledu v celém rozsahu dokumentu a co možná největší usnadnění případných změn. To by mělo být umožněno důsledným používáním kaskádových stylů pro formátování textu, které MS Word podporuje.

Může se zdát zcela zbytečné na toto upozorňovat, ale v některých starších prezentacích nejsou možnosti stylů využity – každý nebo většina textů s jiným než základním vzhledem je formátována zvlášť, což jakékoliv rozsáhlejší úpravy dokumentu značně znepříjemňuje.

Stránky dokumentu jsou samozřejmě zpracovány jako snímky prezentace. Obsah každé stránky je rozložen tak, aby vhodně vyplnil dostupný prostor a související texty či obrázky jsou pokud možno uvedeny na stejné stránce. To znamená, že například změnou fontu či velikosti písma může dojít k přetečení obsahu na další stránky či jiné nežádoucí deformace. Také se projevily problémy v případě otevření dokumentu v novější verzi aplikace MS Word. V některých případech tedy může být nutno opravit rozložení prvků v dokumentu, ale například změna barvy zvýrazněného textu se obejde s pouhou změnou nastavení stylu.

Veškeré grafy a další ilustrace vektorového charakteru byly také vytvořeny přímo ve Wordu, rastrové ilustrační obrázky byly získány z internetových zdrojů. Pro jakékoliv potřebné úpravy obrázků jsem použil bitmapový grafický editor PhotoFiltre[1], který je pro nekomerční účely zdarma.

[1] http://www.photofiltre–studio.com/pf7–en.htm

Kapitola 6

Prezentace o Prologu

Cílem prezentace je seznámit s programováním v Prologu studenty, kteří na počátku nemusejí mít žádné znalosti deklarativního programování a Prologu.

Cílem této kapitoly je však zmapovat podstatné části prezentace a případně objasnit, proč jsou v prezentaci uvedeny. To často obnáší alespoň stručné vysvětlení dané problematiky, aby bylo možno její probírání zdůvodnit, nicméně se předpokládá, že čtenář následujícího textu už určité znalosti Prologu má, nebo přinejmenším není jeho cílem je získat.

Při tvorbě prezentace a v důsledku i obsahu této kapitoly bylo z velké části čerpáno z oficiální dokumentace B–Prologu – zvolené implementace Prologu [1].

Následující podkapitoly nemapují přesně všechny kapitoly a podkapitoly přednášek. Především jsou opomenuty části, které se soustředí pouze na uvedení a vysvětlení dalších možností Prologu, ale nezabývají se žádnými novými postupy. Popisování všech méně podstatných kapitol by zřejmě nemělo velký přínos a zbytečně by prodlužovalo tento text.

6.1 Imperativní a deklarativní programování

V této úvodní kapitole je krátce a bez zbytečných podrobností nastíněn rozdíl mezi imperativním a deklarativním programováním. Je uvedeno, že imperativní programování dává větší svobodu při řešení problémů, protože v podstatě stojí blíže strojovému kódu a hardwaru. Umožňuje tedy implementovat libovolný algoritmus s vysokou efektivitou kódu. Na druhou stranu, řešení problému často vyžaduje hodně času a přemýšlení.

Mnohé deklarativní jazyky (Prolog není výjimkou) oproti tomu přicházejí s již hotovým, dosti pokročilým mechanismem hledání řešení, který ovšem není vhodný pro všechny typy problémů. Například úlohu, která vyžaduje složité matematické výpočty, bychom nejspíše řešili v imperativním jazyce.

Pro přiblížení deklarativního přístupu programátorům znalým imperativního programování jsou také uvedeny příklady deklarativního přístupu v imperativních jazycích – cyklus for–each, který ve své podstatě nespecifikuje konkrétní způsob průchodu polem nebo jinou strukturou, a mapování do pole funkcí [15], které je součástí základních knihoven především ve skriptovacích jazycích, jako je Javascript či PHP.

Ukázka imperativního přístupu v Javascriptu

```
var numbers = [1,2,3,4,5];
var squares = [];

for (var i = 0; i < numbers.length; i++) {
  squares.push(numbers[i] * numbers[i]);
}

console.log(squares); // [1,4,9,16,25]
```

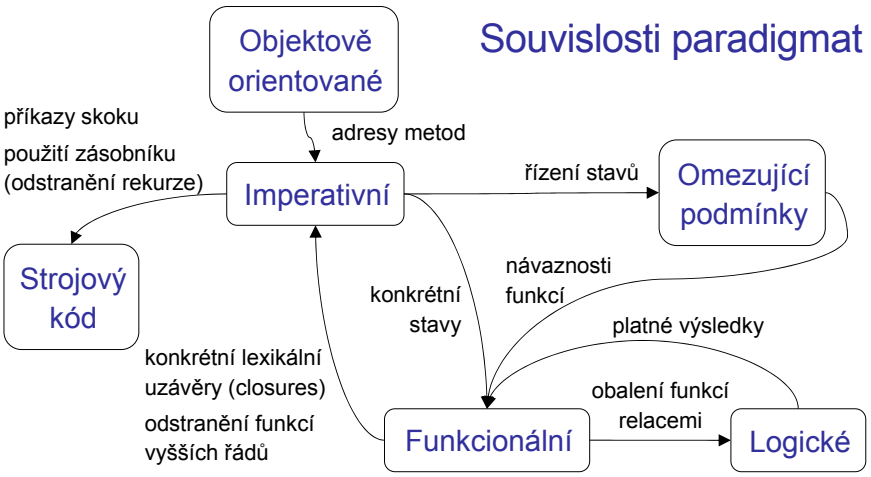
Ukázka deklarativního přístupu v Javascriptu

```
var numbers = [1,2,3,4,5];

var squares = numbers.map(function(n) {
  return n * n;
})

console.log(squares); // [1,4,9,16,25]
```

Na závěr jsou uvedeny některé jiné deklarativní jazyky (kromě Prologu), konkrétně SQL [14][20] pro svou, i když na první pohled nezřetelnou, podobnost s Prologem [24] a Lisp, který je nejpoužívanější alternativou Prologu při programování systémů umělé inteligence [19].



Obrázek 6.1: *Zjednodušený graf souvislostí paradigmat, který je také uveden v prezentaci [13]. Graf především znázorňuje větší blízkost imperativního paradigmatu a strojového kódu ve srovnání s deklarativními paradigmaty. Imperativní program tedy má blíže k tomu, co se ve stroji skutečně děje.*

6.2 Historie Prologu

Zde je stručně uvedeno, kdy a kde Prolog vznikl a kdo se na jeho tvorbě především podílel (Alain Colmerauer jakožto tvůrce jazyka a Robert Kowalski jakožto autor procedurální interpretace hornových klauzulí, ze které Prolog vychází [21][5]).

Především jako zajímavost je zde více rozebrán patrně nejvíce ambiciózní pokus o využití Prologu v historii. Jedná se o japonský projekt počítače páté generace, který probíhal v 70. letech. Japonsko, zvláště v tehdejší době technologická velmoc, nebylo spokojeno se svou pozicí pouhého konzumenta v oblasti rozvoje výpočetní techniky. Japonské ministerstvo mezinárodního obchodu a průmyslu[1] proto schválilo odvážný plán, podle něhož měli japonští výzkumníci do deseti let vyvinout vysoce výkonný počítač, založený čistě na deklarativním programování. Protože tehdejší počítače byly označovány

[1] Japanese Ministry of International Trade and Industry (MITI)

za třetí generace[2], označení pátá generace mělo evokovat pokrokovost a nadčasovost projektu [23].

Přestože projekt skončil velkým neúspěchem, vyvolal další vývoj deklarativního programování. Za hlavní příčiny neúspěchu se považuje nedostatečný výkon tehdejšího hardwaru pro úspěšné použití deklarativního přístupu a zcela špatný odhad dalšího vývoje. Japonští odborníci předpokládali, že samostatné výpočetní jednotky brzy dosáhnou svého maximálního potenciálu a masivní paralelizace tak bude jedinou cestou ke zvýšení výkonů, zatímco ve skutečnosti nové mikroprocesory rychle překonaly každý paralelní systém, který postavili.

Protože koncepce počítače páté generace nápadně připomíná superpočítače objevující se v dobových sci-fi, pro oživení jsem prezentaci doplnil o obrázky těchto imaginárních strojů, jako je například počítač HAL–9000 z knih a filmů 2001: Vesmírná odysea a 2010: Druhá vesmírná odysea.

6.3 Použití Prologu v současnosti a příklady

V této části jsem se pokusil shrnout zjištěná tvrzení formou otázek, které by mohly ohledně použití Prologu padnout, a co možná nejlepších odpovědí na ně. Nápadná je podobnost s "Často kladenými dotazy (FAQ)", což je sekce oblíbená především napříč všemi možnými webovými stránkami.

Dále jsem zahrnul konkrétní útržky (respektive citace) z internetových diskusí, které hovoří o užívání a efektivitě Prologu.

Aby motivační část nebyla omezena jen na neurčitá tvrzení, uvedl jsem v prezentaci kompletní ukázky praktického použití Prologu. Za nejvíce reprezentativní považuji použití v systému Clarissa pro vesmírnou stanici ISS[3] a při programování superpočítače pro zpracování přirozené řeči firmy IBM, pojmenovaném Watson[4].

[2]Rozdělení podle použité technologie: první generace – vakuové trubice; druhá generace – tranzistory; třetí generace – integrované obvody; čtvrtá generace – mikroprocesory. Rozdělení podle způsobu programování: první generace – strojový kód; druhá generace – jazyky symbolických adres; třetí generace – strukturované programovací jazyky; čtvrtá generace – jazyky pro specifické použití (SQL, TeX).

[3]International Space Station

[4]Často uváděn jako IBM Watson

Jako vhodné ukázky jsem tato užití označil proto, že se jedná o poměrně nedávné projekty (Clarissa z roku 2005 a Watson z roku 2011), které byly poměrně dost zpopularizované v médiích a lze o nich tak najít poměrně dost informací.

Clarissa je hlasem ovládaný asistenční systém, který astronautům usnadňuje jejich úkoly na stanici. Posádka totiž musí průběžně provádět zhruba 12 000 rutinních úkonů jen pro udržení stanice v normálním provozu a navíc se věnovat momentálně probíhajícím vědeckým projektům. Protože patrně není v lidských silách, zapamatovat si všechny nutné úkony, byla při práci odkázána na sledování manuálů. To je však vyčerpávající, odvádí to pozornost od úkolu samotného a může to vést i k chybám. Stížnosti astronautů na tento problém podnítily vývoj systému interaktivního hlasového systému, který během provádění procedur poskytuje rady [9][10].

IBM Watson je superpočítač, vyznačující se pokročilou schopností porozumět přirozenému jazyku. Vývojový tým IBM si vybral televizní soutěž *Jeopardy!* jako novou výzvu pro umělou inteligenci. Tato hra připomíná hru Riskuj!, soutěžícím však nejsou otázky předkládány běžnou formou, ale jako tzv. nápovědy (clues). Jedná se vlastně o popis určitého předmětu, osoby, pojmu, místa... Aby soutěžící získal body, musí zformulovat otázku ve tvaru kdo/co je X, která se na popis hodí. Před účastí v soutěži Watson předzpracoval velké množství dostupných textů v přirozeném jazyce (např. obsah Wikipedie), ze kterých si vytvořil databázi vědomostí. Během soutěže potom hledá asociace, které ho dovedou ke správné odpovědi. [7][8].

Dále jsem v prezentaci krátce rozebral projekt sovětského raketoplánu Buran (přestože je staršího data), při jehož programování byl Prolog dle dostupných údajů také využit [11][22]. Pokud je mi známo, technické podrobnosti nejsou snadno dohledatelné – uvedená fakta slouží spíše jako zajímavost. Dalších několik použití je uvedeno spíše jen heslovitě, především pro nedostatek údajů.

Ve většině případů je totiž obtížné vůbec dohledat software, při jehož programování byl Prolog použit. Pokud se toto podaří, stále je o takovém projektu k dispozici jen minimum informací. Obvykle se jedná o rozsáhlé a nákladné projekty, určené pro velké organizace nebo státní správu, které se ocitají mimo zájem veřejnosti a utajení podrobností také často bývá předmětem smlouvy mezi zákazníkem a dodavatelem, či přímo firemní politikou výrobce.

Jsem tedy přesvědčen o tom, že existuje podstatně více významných softwarových produktů, než kolik se mi jich podařilo dohledat a kolik jich je uvedeno v prezentaci.

6.4 Syntax Prologu a postup hledání řešení

Na začátku kapitoly je stručně zopakováno, jakou podobu musí mít logická formule, aby byla klauzulí a co musí splňovat klauzule, aby byla hornovou klauzulí – Především pomocí příkladů, který považuji za nejlepší způsob připomenutí. Dále jsou zopakovány logické termy, o které výrokovou logiku rozšiřuje predikátová logika – tedy proměnné, predikáty a kvantifikátory.

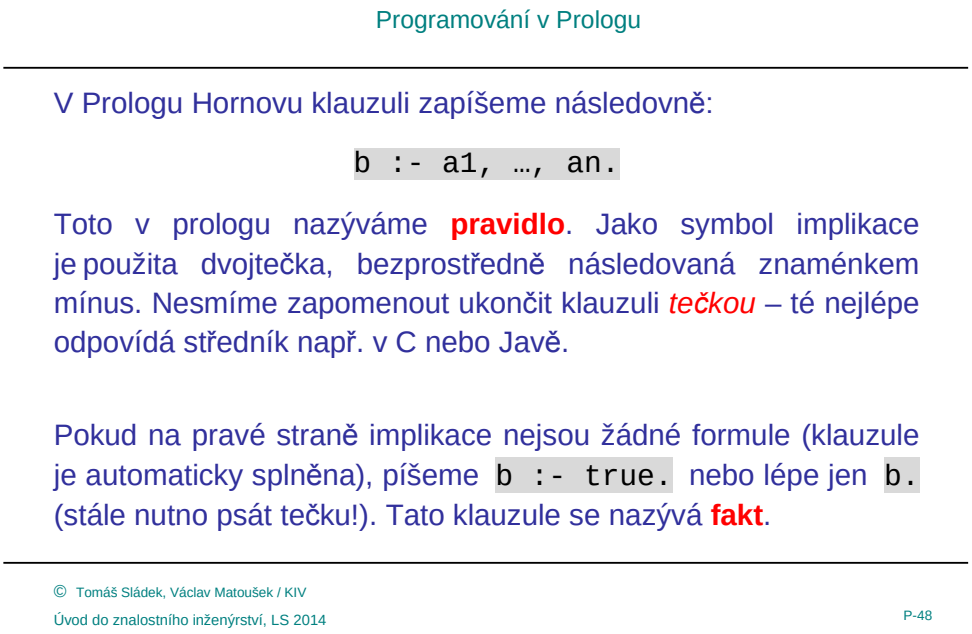
Tyto znalosti jsou potřebné pro pochopení, jak vznikl syntax Prologu, který je další náplní této kapitoly. Je popsáno, co je v Prologu pravidlem, co je fakt a co je cíl (viz obr. 6.2). Dále je uveden první jednoduchý program v Prologu, kde je vytvořena malá databáze faktů a popsáno, co se děje při dotazování [6].

Krátká podkapitola dále podrobně popisuje, s jakými typy termů se v Prologu lze setkat (většina z nich už studentům nejspíše bude známá z predikátové logiky) a jak přesně se v Prologu zapisují – aby studenti měli možnost sžít se s Prologovskou syntaxí už předem, stejné značení je použito už od začátku kapitoly – i v části, která se ještě Prologu nevěnuje.

6.5 Průvodní příklad

Tento příklad nespadá do jedné konkrétní kapitoly prezentace. Jde o mírně složitější příklad, než jsou běžně uváděné jedno či několikařádkové příklady, užívané pro demonstraci fungování predikátů a operátorů. Nejjednodušší podoba programu, zatím velmi omezená, je uvedena ihned po probrání základů Prologu. Ten je pak v etapách upravován blíže k dokonalosti až téměř do konce prezentace.

S každou další probranou oblastí Prologu je tento příklad rozšířen s využitím nově nabytých znalostí. Tento vztah funguje i opačně – na příkladu je nejprve představen problém, k jehož řešení potřebujeme nějaké nové znalosti. To může sloužit jako motivace ke studiu dalších oblastí Prologu. Cílem použití průvodního příkladu je průběžné předvádění různých možností Prologu trochu praktičtějším způsobem.



Obrázek 6.2: *Kompletní snímek z prezentace. Ukázka je z úvodu do Prologu. Pro označení důležitých pojmů je používáno tučné červené písmo. Další důležité části textu jsou zvýrazněny červenou kurzívou. Ukázky zdrojového kódu (či výpisy z konzole) jsou psány neproporcionálním písmem s šedým podkladem.*

Program slouží pro velmi zjednodušené hledání cesty na mapě světa – nejmenším prvkem cesty je stát. Databáze faktů se tedy skládá z definic sousednosti států. Zejména zpočátku je v databázi definováno jen velmi malé množství států – Je zřejmé, že pro účely demonstrace není nutné zdrojový kód zbytečně prodlužovat doplňováním velkého množství států. Počáteční varianty programu jsou navíc velmi omezené (používají jen zcela základní možnosti Prologu) a s kompletnější mapou světa by ani nebyly schopny správně pracovat.

Počáteční podoba příkladu

```
sousedi('Kanada','USA').
sousedi('USA','Mexiko').
sousedi('Rusko','Cina').
sousedi('Cina','Indie').

cesta(X,Y) :- sousedi(X,Y).
cesta(X,Y) :- sousedi(X,Z), cesta(Z,Y).
```

Následující konečná podoba příkladu je v prezentaci pochopitelně uvedena mnohem později a nenachází se ve stejné kapitole, protože, jak již bylo řešeno, tento příklad prakticky provází celou přednáškou. Vzhledem k evidentní souvislosti ji však pro srovnání uvádím zde.

Konečná podoba příkladu

```
sousedi('Kanada','USA').
sousedi('USA','Mexiko').
sousedi('Rusko','Cina').
sousedi('Rusko','Mongolsko').
sousedi('Mongolsko','Cina').
sousedi('Cina','Kazachstan').
sousedi('Kazachstan','Rusko').
sousedi('Cina','Indie').

prechod(X,Y) :- sousedi(X,Y).
prechod(X,Y) :- sousedi(Y,X).

sekvence(N,N,N) :- !.
sekvence(K,K,_).
sekvence(I,K,N) :- I1 is I+1, sekvence(I1,K,N).
sekvence(K,N) :- sekvence(1,K,N).

cesta(X,X,_,[X],_) :- !.
cesta(X,Y,CP,[X|FP],L) :-
    not(member(X,CP)), L > 0, L1 is L-1,
    prechod(X,Z), cesta(Z,Y,[X|CP],FP,L1).
cesta(X,Y,P) :- sekvence(L,8), cesta(X,Y,[],P,L), !.
```

Toto je ve skutečnosti pouze jedna z navržených konečných podob příkladu, které jsou v prezentaci uvedeny. Zdůrazňuje výhodu použití iterativního prohlubování s pevnou maximální hloubkou v Prologu, pokud očekáváme, že řešení leží ve stromě poměrně mělko. Protože algoritmus iterativního prohlubování je postaven na prohledávání do hloubky, umožňuje využít nativní hledání řešení v Prologu.

Jiné možné řešení používá predikáty náležející predikátové logice vyššího řádu – deklarativní cykly a predikát `findall` pro hledání kompletní množiny všech možných řešení. Pomocí těchto predikátů je realizován algoritmus hledání do šířky. Kromě jeho obvyklé nevýhody v podobě větší spotřeby paměti je také uvedeno, že se evidentně nehodí pro Prolog, protože výsledný program je o poznání složitější a v podstatě potlačuje nativní způsob hledání řešení, který je v Prologu implementován. S vynecháním faktů, která se shodují s předcházející variantou, vypadá tato alternativa následovně:

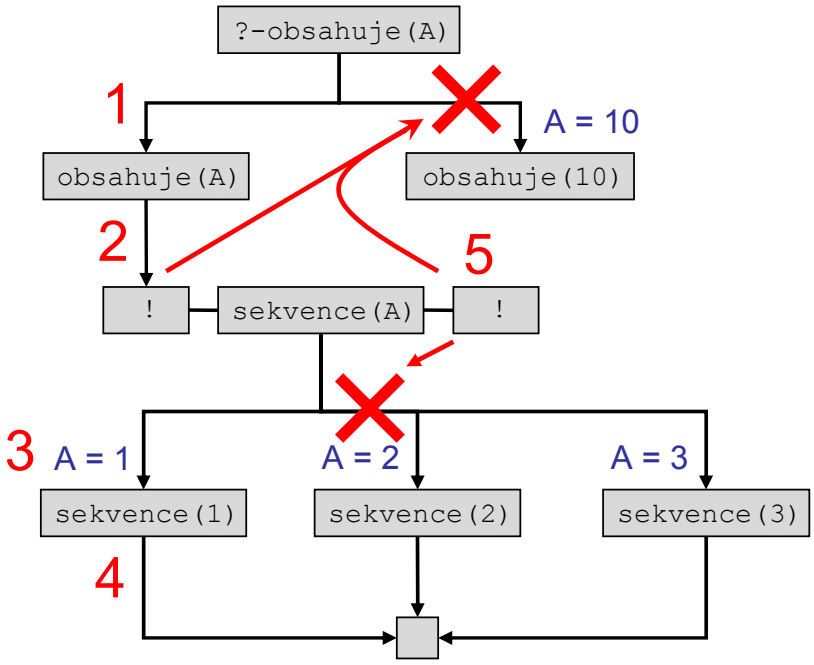
Alternativní podoba příkladu

```
bfscesta(Y,CPs,FP) :-
  foreach( %Průchod seznamu dosud zvažovaných cest
    CP in CPs, %Výběr jedné z cest
    ac(NPs,[]), %Při průchodu akumulujeme prodloužené cesty
    [X,Z,Ps,NPsi,_T], %Lokální proměnné
    (
      CP = [X|_T], %Výběr současné pozice z cesty
      findall([Z|CP],(prechod(X,Z), not(member(Z,CP))),Ps),
        %Najít všechna možná prodloužení
      foreach(P in Ps,ac(NPsi,NPs^0),NPsi^1 = [P|NPsi^0]),
        %K prodloužením předchozích cest přidat prodloužení
        %této cesty
      NPs^1 = NPsi
        %Nový stav akumulátoru s prodlouženími této cesty
    )
  ),
  NPs \= [], %Kontrola, zda nějakou cestu byla prodloužena.
        %Pokud ne, cesta do cíle neexistuje.
  bfscesta(Y,NPs,FP). %Žádné větvení, pouze další iterace
cesta(X,Y,P) :- bfscesta(Y,[[X]],P). %Stejný počet argumentů
                                     % - nutno změnit název.
```

6.6 Cut & fail

Jakožto jedno z prvních rozšíření nejstarší verze Prologu, predikát řezu `cut` (v Prologu zapisovaný vykřičníkem) ve spojení s predikátem `fail` je uveden jako první možnost, jak podle potřeby "utnout" hledání řešení [4] (viz obr. 6.3).

Tento poznatek je následně použit v průvodním příkladu, aby se předešlo zacyklení při prohledávání mapy. To je v podstatě problém ekvivalentní prohledávaní neorientovaného grafu. I když se z počátku jedná o acyklický graf, je nutno nějakým způsobem zabránit vracení po cestě, kudy program během hledání přišel.



Obrázek 6.3: *Ukázka použití grafu v prezentaci – ilustrace fungování řezu. Červené číslice naznačují posloupnost vyhodnocování predikátů. Přeškrtnutí větve znamená nevyhodnocení v důsledku řezu, od kterého vede šipka (v případě více řezů by stačil jeden z nich).*

6.7 Proměnné a datové typy

Zde je vysvětlen pojem volná a vázaná proměnná. Dále je rozveden princip ztotožňování a z něco plynoucí navazování proměnných a probrány jeho důsledky.

Protože Prolog je slabě typovaný jazyk, zabývat se typy termů obvykle není nutné. Přesto jsou v přednášce uvedeny predikáty, které jejich typy umožňují ověřit. Podrobnější prozkoumání a vyzkoušení záleží na zájmu studentů. V této části je nicméně zdůrazněno, že pravdivostní hodnota nepatří mezi možné typy termu – tak, jako v logice, i v Prologu existují termy, které mají vždy danou pravdivostní hodnotu (pravda/nepravda), ale pravdivostní hodnota samotná je teprve důsledkem jejich vyhodnocení a nemůže být nikdy ztotožněna z proměnnou.

6.8 Omezující podmínky

Současné verze Prologu nepoužívají jen logické programovací paradigma, ale jsou rozšířeny také o omezující podmínky. Jedná se v podstatě o predikáty a operátory, které v případě nesplnění daných podmínek okamžitě ukončí vyhodnocování pravidla, bez zkoumání dalších možností hlouběji ve stromě.

V této kapitole je uveden predikát `not`. Ten je významný tím, že umožňuje zjistit, co nejde, namísto toho, co jde – tedy určit, který term splnitelný není. Z tohoto termu lze odvodit další operátory pro zjišťování shody a neshody termů, které jsou v této kapitole uvedeny.

Význam omezujících podmínek je (vyjma malých vykonstruovaných ukázek) prezentován úpravou demonstračního programu do poněkud čitelnější podoby. Úprava také umožňuje následně odstranit konstrukci `cut & fail`, jejíž používání v programech je diskutabilní (viz následující téma v prezentaci – *Správné použití řezu*).

6.9 Správné použití řezu

Řez byl do Prologu zaveden za účelem urychlení výpočtu, aby se jeho použitím na vhodných místech předešlo hledání řešení ve větvích stromu, kde se určitě nacházet nebude. Použití řezu je s jistotou oprávněné pouze v případě, kdy po jeho odstranění bude program vyhodnocen stejným způsobem, až na to, že Prolog bude do úvahy brát více možností.

Tak, jako k tomu občas dochází u jiných technik napříč různými programovacími jazyky, bývá řez zneužíván k tomu, aby se docílilo žádoucího vyhodnocení pravidel. Nevýhodou je, že takové použití řezu ovlivní vyhodnocení dalších pravidel v jiných částech programu. Nutí tak programátora sledovat provádění programu sekvenčně, pokud chce zjistit, jaký bude výsledek. Tím vlastně do Prologu zavádí vlastnosti imperativního programování.

Kromě vysvětlení těchto skutečností je také uvedeno, že oprávněnost použití řezů záleží na situaci a při rozhodování není vhodné vždy slepě následovat napsaná pravidla.

6.10 Seznamy

Tak, jako v běžně užívaných jazycích pole, jsou seznamy v Prologu základním prostředkem, jak uchovávat předem neznámé množství hodnot. Proto je jim v prezentacích věnována značná pozornost.

Zvláštní pozornost je věnována problematice přidávání prvků na konec seznamu, které program může značně zpomalit. Je proto rozebrána činnost predikátu `append` – ten kvůli připojení na konec seznamu prochází celý původní seznam, protože v Prologu nelze jednoduše přistoupit na konec seznamu. V souvislosti s tím je uvedena technika rozdílových seznamů, která může rychlé přidávání prvků na konec v určitých případech řešit.

Technika spočívá v tom, že zbytek seznamu (tail) je volná proměnná, seznam je tedy otevřený. Navázáním této proměnné na další otevřený seznam lze původní seznam prodloužit v konstantním čase, přičemž přidané prvky budou na konci [3].

V souvislosti se seznamy jsou uvedeny také Prologovské řetězce, které jsou prakticky také seznamy, a možnosti převodu mezi řetězci a atomickými termy s odpovídajícím názvem, či naopak ukládání názvů termů či znaků z číselných hodnot do řetězců.

Použití seznamů je poté prezentováno v průvodním příkladu, kde slouží k ukládání dosavadní cesty. Díky tomu je možno rozpoznat již navštívené státy a předejít jejich opakovanému procházení. Také program konečně může sestavit celou cestu od začátku do konce a vypsat ji na výstup.

6.11 Rekurze a technika akumulátoru

V závěrečné části přednášek je uvedeno, co je pravá a levá rekurze (rekurzivní volání je umístěno na začátku nebo na konci těla pravidla, tedy vlevo nebo vpravo na řádce). Je vysvětleno, proč je pravá rekurze vhodnější (šetření paměti zásobníku díky znovupoužívání alokované paměti) a jak lze levorekurzivní pravidlo přepsat na pravorekurzivní s použitím akumulátoru [3].

V souvislosti se zkoumáním využití paměti jsou uvedeny také základní možnosti profilace kódu v B–Prologu.

Kapitola 7

Ostatní části práce

7.1 Příklady

Při výuce programovacího jazyka je samozřejmě nanejvýš vhodné uvádět také příklady programů. V prezentaci se vyskytuje průvodní příklad a velice krátké příklady, předvádějící jednotlivé probírané příkazy.

Formát prezentace, která je rozdělená na jednotlivé snímky a je psaná velkým písmem, aby byla čitelná bez přibližování i z velké vzdálenosti, však není příliš vhodný pro delší příklady. Další příklady jsou proto v samostatných souborech, které jsou rozděleny do dvou skupin.

7.1.1 Příkazy

Tyto soubory obsahují vesměs velmi jednoduché příklady, podobné těm, které jsou uvedeny v prezentaci. Některé jsou v prezentaci přímo použity v téměř nezměněné podobě, tyto soubory však umožňují jejich snadné vyzkoušení. Další uvedené příklady nabízejí další ukázky z dané problematiky, případně se zabývají dalšími tématy, které do rozsahu prezentace nevešly.

7.1.2 Ostatní

Ostatní příklady jsou vesměs o něco složitější programy, vytvořené, nebo převzaté, které mohou být předvedeny na přednášce, stranou od prezentace, probrány na cvičení, nebo pouze ponechány studentům ke stažení a samostatné prostudování – rozhodnutí záleží na posouzení vyučujícího.

Nejzajímavějším příkladem z této sbírky je patrně program, který provádí učení perceptronu a následné použití pro klasifikaci [2]. Uvádět zde

kompletní příklad považuji za zbytečné. Na ukázku pouze dva zajímavější predikáty z programu – predikát pro klasifikaci a predikát pro učení na základě výsledku klasifikace:

Ukázka z programu pro perceptron

```
%Zařadí bod do třídy 1 nebo -1 podle jeho souřadnic
%a nastavené váhy.
classify(Point, Weights, Class) :-
    Class is +1, dot_product(Point,Weights,Res), Res >= 0.
classify(_Point, _Weights, Class) :-
    Class is -1.

...

%Upraví váhy perceptronu, pokud bod nedává do správné třídy.
perceptron(LearnRate, Point, Desired, Weights, NewWeights)
:-
    classify(Point, Weights, Class),
    Delta is LearnRate * (Desired - Class), %Liší se třída?
    scalar_mul(Delta,Point,DeltaWeights),
    vector_sum(Weights,DeltaWeights,NewWeights).
```

7.2 Převedení textů v obrázcích

Úprava se týká již existující prezentace o logice. Tato prezentace vznikla za použití starších materiálů, které byly k dispozici pouze v nascanované podobě – v rastrovém formátu. Ne všechny použité materiály však byly převedeny do textu (viz výše). V dokumentu se tudíž vyskytovaly celostránkové obrázky, které na první pohled mohly vypadat jako text, ale nebylo možno je kopírovat, vyhledávat v nich a podobně. Proto jsem veškerý textový obsah, u kterého se tak dosud nestalo, přepsal z obrázků do prostého textu (příklad viz obr. 7.1).

7.3 Příklady z teoretické informatiky pro UIR

Připravil jsem několik jednoduchých příkladů k zopakování základních dovedností z předmětu *Teoretická informatika* (ukázka viz obr. 7.2). Konkrétně

se jedná o návrh konečných automatů a gramatik, což jsou jednoduché související problémy, jejichž princip lze pochopit z jedné či dvou ukázek. Tyto příklady mohou být se studenty řešeny v průběhu jednoho z úvodních opakovacích cvičení předmětu UIR.

Vzhledem k tomu, že u předmětu UIR dojde ke změně vyučujícího, bude záležet na něm, zda návrh na zopakování teoretické informatiky vůbec bude brát v úvahu, nicméně příklady jsou dostupné k zapracování do plánu cvičení.

ODVOZOVÁNÍ V PREDIKÁTOVÉ LOGICE 1. ŘÁDU

Podobně jako v případě výrokové logiky můžeme k axiomům přidat další formule – *mimologické axiomy*. Množinu mimologických axiomů T pak nazveme *teorií* jazyka predikátové logiky 1. řádu.

Dedukcí, tj. použitím odvozovacích pravidel, můžeme z teorie T dokázat další formule. Jestliže formule $\mathcal{A}$ je *formálně dokazatelná z teorie* T (je teorémem v T), píšeme

$$T \vdash \mathcal{A}\,.$$

Odvozování v predikátové logice 1. řádu

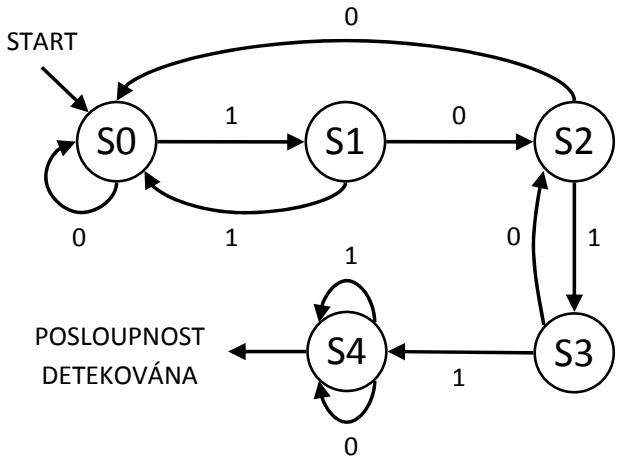
Podobně jako v případě výrokové logiky můžeme k axiomům přidat další formule – *mimologické axiomy*. Množinu mimologických axiomů T pak nazveme *teorií* jazyka predikátové logiky 1. řádu.

Dedukcí, tj. Použitím odvozovacích pravidel, můžeme z teorie T dokázat další formule. Jestliže formule $\mathcal{A}$ je *formálně dokazatelná* z teorie T (je teorémem v T), píšeme:

$$T \vdash \mathcal{A}$$

Obrázek 7.1: *Přednáška z logiky – porovnání původního snímku, s textem nascanovaným do obrázku (nahoře) a nové verze, předělané z obrázku do textu (dole).*

Příklad 1 Ukázka konečného automatu pro detekci posloupnosti 1011 v libovolné posloupnosti binárních čísel:



Cílový stav S4 je zároveň absorpční stav – bez ohledu na vstup už automat svůj stav nemění.

Obrázek 7.2: *Z příkladů pro UIR – řešený příklad na konečný automat*

Kapitola 8

Zhodnocení

Po posouzení připravené prezentace k výuce Prologu jsme já a vedoucí práce pan prof. Matoušek toho názoru, že obsažené informace jsou pro programátora, který nemá zkušenosti s logickým programováním a s Prologem, dostatečné k tomu, aby dokázal Prolog prakticky použít.

Přestože se jednotlivé kapitoly často zabývají v principu nesouvisející problematikou, použití průvodního příkladu do většiny prezentace vnáší určitou kontinuitu. Prezentaci tak lze vnímat jako nedělitelný celek, i jako oddělené samostatné části. Záleží na tom, zda má posluchač či čtenář zájem o prostudování veškeré problematiky, či jen nějaké konkrétní oblasti.

Rozsah výsledné prezentace mírně přesahuje 200 stran. Vypadá to jako velké množství, nicméně, v daném formátu může jedna strana pojmout jen relativně malé množství informací. Kromě toho, některé části není třeba během přednášky dlouho rozebírat, nebo je lze zcela vynechat, pokud nejsou podstatné. Prezentace jsou určeny i k samostudiu, což je v dnešní době, kdy bývají k dispozici ke stažení, běžné, takže neuvedení během přednášky neznamená, že dané informace zůstanou studentům skryté. Z těchto důvodů si myslím, že problematika je v rámci přednášek zvládnutelná.

Praktickou použitelnost prezentace při výuce bude pochopitelně možno plně ověřit až v následujícím akademickém roce, kde má být v průběhu přednášek z předmětu UZI poprvé využita.

Je docela dobře možné, že vyučující pocítí potřebu prezentaci před prvním použitím ještě mírně upravit. Bylo by například dobré začlenit do prezentace odkazy na stránky předmětu, kde zřejmě budou k dispozici další studijní materiály, to ale není dost dobře možné v situaci, kdy stránky předmětu za-

tím neexistují. Určitě by však nemělo být nutné provádět jakékoliv zásadní změny v obsahu prezentací.

Další návrhy určené k realizaci byly podle mého názoru také uskutečněny. Veškeré texty, které se v původních prezentacích o logice vyskytovaly v obrázkové podobě, byly přepsány do textové podoby. Změny v textu jsou zanedbatelné a v žádném případě nedošlo ke změně obsahu. Pro opakování teoretické informatiky v předmětu UIR jsem navrhl několik příkladů, jejichž zařazení do programu cvičení nicméně závisí na budoucím vyučujícím předmětu.

Kapitola 9

Přiložené soubory

Uvedené soubory jsou přiloženy na kompaktním disku:

`Prolog-2014.docx` – Zdrojový soubor prezentace z Prologu.

`Prolog-2014.pdf` – Výsledná prezentace z Prologu.

`Logika-2014.docx` – Zdrojový soubor upravené prezentace z logiky.

`Logika-2014.pdf` – Výsledná upravená prezentace z logiky.

`Automaty&gramatiky.docx` – Příklady pro UIR.

`priklady.zip` – Soubor příkladů doplňujících přednášky.

Kapitola 10

Závěr

Hlavním cílem práce bylo vytvoření prezentace pro přednášky o Prologu. Výsledek svým rozsahem pokrývá dané téma od úplných základů logického programování až po některé speciality zvolené implementace Prologu.

V úvodní části se prezentace zabývá také praktickým použitím Prologu v současnosti. Mezi odbornou veřejností je poměrně rozšířen názor, že Prolog se hodí pouze na hraní, či na akademické pokusy. Ukázkami využití v praxi se tedy prezentace snaží tento názor u posluchačů vyvrátit a motivovat ke studiu Prologu.

K motivaci je také využit průvodní příklad – program, který je postupně zdokonalován se získáváním nových znalostí o Prologu.

Prezentace byla vedoucím práce posouzena jako kvalitní. I v případě osoby se zkušenostmi s Prologem a s výukou je takové hodnocení vždy částečně subjektivní, ale jednoduchá objektivní měřítka se v tomto případě hledají jen těžko. Přesněji bude práci možno posoudit teprve ve chvíli, až bude prezentace prakticky použita při výuce.

Také byly naplánovány a splněny dva vedlejší cíle. Stávající prezentace o logice pro předmět UIR, která obsahovala nascanované obrázky s textem namísto obyčejného textu, byla zbavena tohoto nedostatku. Pro předmět UIR byly vytvořeno několik příkladů k zopakování znalostí z teoretické informatiky, na které předmět UIR navazuje.

Práce dále obsahuje návrhy dalších rozšíření výukových materiálů, jejichž uskutečnění by bylo nad rámec práce, nebo by bylo momentálně problematické, vzhledem k probíhajícím úpravám skladby předmětů.

Literatura

[1] NENG–FA, Zhou. AFANY SOFTWARE & CUNY & KYUTECH. B–Prolog User's Manual. Prolog, Agent, and Constraint Programming [online]. 2013. Dostupné z: www.probp.com

[2] CSENKI, Attila. VENTUS PUBLISHING APS. Applications of Prolog [online]. 2009. ISBN 978–87–7681–514–1. Dostupné z: bookboon.com

[3] CSENKI, Attila. VENTUS PUBLISHING APS. Prolog Techniques [online]. 2009. ISBN 978–87–7681–476–2. Dostupné z: bookboon.com

[4] BLACKBURN, Patrick, BOS, Johan, STRIEGNITZ, Kristina. Learn Prolog Now! [online]. 2014 [2014–05–03]. Dostupné z: http://learnprolognow.org/

[5] KOWALSKI, EMDEN. The Semantics of Predicate Logic as a Programming Language. University of Edinburgh [online]. 1976 [2014–05–03]. Dostupné z: http://www.doc.ic.ac.uk/~rak/papers/kowalski–van_emden.pdf

[6] JEŽEK, Karel. Logické programování – PROLOG (programming in Logic). Západočeská univerzita v Plzni [online]. 2012 [2014–05–05]. Dostupné z: http://www–kiv.zcu.cz/~jezek_ka/vyuka/PGS/Pro studenty/08Logicke.pdf

[7] LALLY, Adam a Paul FODOR. Natural Language Processing With Prolog in the IBM Watson System. Association for Logic Programming [online]. 2011 [2014–01–15]. Dostupné z: http://www.cs.nmsu.edu/ALP/2011/03/natural–language–processing–with–prolog–in–the–ibm–watson–system/

[8] The Watson Research Team Answers Your Questions. Building a Smarter Planet [online]. 2011 [2014–01–15]. Dostupné z: http://asmarterplanet.com/blog/2011/02/the–watson–research–team–answers–your–questions.html

[9] NewScientist – Space. Space station gets HAL-like computer [online]. 2005 [2014–04–29]. Dostupné z: http://www.newscientist.com/article/dn7584

[10] NASA SpaceFlight. ISS set to get chatty with Clarissa [online]. 2005 [2014–04–29]. Dostupné z: http://www.nasaspaceflight.com/2005/06/iss–set–to–get–chatty–with–clarissa/

[11] Soviet Software Productivity: Isolated Gains in an Uphill Battle [online]. 1990 [2014–04–29]. s. 7. Dostupné z: http://dl.dropboxusercontent.com/u/529693/Soviet Software Productivity Isolated Gains in an Uphill Battle.pdf

[12] TAMSIN, Treasure-Jones. Prolog Tutorial [online]. 1996 [2014–05–02]. Dostupné z: http://www.doc.gold.ac.uk/~mas02gw/prolog_tutorial/prologpages

[13] BERNARDY, Jean-Philippe. Programming Paradigms [online]. 2014 [2014–05–03]. Dostupné z: http://www.cse.chalmers.se/~bernardy/pp/Lectures.html#sec-9

[14] University of California SANTA CRUZ – Newscenter. Don Chamberlin to receive 2009 Fellow Award from Computer History Museum [online]. 2009 [2014–05–03]. Dostupné z: http://news.ucsc.edu/2009/07/3084.html

[15] ROBERTS, Philip. Imperative vs Declarative [online]. 2013 [2014–05–03]. Dostupné z: http://latentflip.com/imperative-vs-declarative/

[16] Real World Applications. COMPGROUPS.NET [online]. 2009 [2014–04–29]. Dostupné z: http://compgroups.net/comp.lang.prolog/real-world-applications

[17] coding.derkeiler.com. comp.lang.prolog [online]. 2009 [2014–04–29]. Dostupné z: http://coding.derkeiler.com/Archive/Prolog/comp.lang.prolog/2009-10/

[18] Skupiny Google. comp.lang.prolog > Real World Applications [online]. 2009 [2014–04–29]. Dostupné z: https://groups.google.com/d/topic/comp.lang.prolog/Wx_ipFUVWfM/discussion

[19] Wikipedia – The Free Encyclopedia. Lisp (programming language) [online]. 2014 [2014–05–03]. Dostupné z: http://en.wikipedia.org/wiki/Lisp_(programming_language)

[20] Wikipedia – The Free Encyclopedia. SQL [online]. 2014 [2014–05–03]. Dostupné z: http://en.wikipedia.org/wiki/SQL

[21] Wikipedia – The Free Encyclopedia. Prolog [online]. 2014 [2014–05–03]. Dostupné z: http://en.wikipedia.org/wiki/Prolog

[22] Wikipedia – The Free Encyclopedia. Buran (spacecraft) [online]. 2014 [2014–04–29]. Dostupné z: http://en.wikipedia.org/wiki/Buran_(spacecraft)

[23] Wikipedia – The Free Encyclopedia. Fifth generation computer [online]. 2014 [2014–05–03]. Dostupné z: http://en.wikipedia.org/wiki/Fifth_generation_computer

[24] Stack Overflow. Difference between SQL and Prolog [online]. 2014 [2014–05–03]. Dostupné z: http://stackoverflow.com/questions/2117651/difference-between-sql-and-prolog