PŘÍRODOVĚDECKÁ FAKULTA UNIVERZITY PALACKÉHO
KATEDRA INFORMATIKY

DIPLOMOVÁ PRÁCE

Implementace expertního systému v jazyce Common Lisp

2013 Jakub Kaláb

Anotace

Expertní systémy mají v praxi bohaté využití. Jejich smyslem je asistovat expertovi na danou problematiku, či jej plně nahradit. V příloze bakalářské práce implementuji prázdný expertní systém s dopředným řetězením inspirovaný systémem CLIPS jako knihovnu v programovacím jazyku Common Lisp tak, aby jej bylo možno plně integrovat do dalších programů.

Děkuji Mgr. Martinu Dostálovi, Ph.D. za vedení této diplomové práce.

Obsah

Seznam obrázků

Seznam příkladů

1. Úvod

Ve své bakalářské práci jsem implementoval základní knihovnu pro tvorbu expertních systémů (tzv. prázdný expertní systém) s dopředným řetězením v jazyce Common Lisp. Cílem této diplomové práce je tuto knihovnu rozšířit o následující:

- syntaktický režim pro zajištění přiměřené kompatibility se systémem CLIPS,
- možnost vrácení provedených změn včetně odvozovacích kroků,
- podpora pro ladění s jednoduchým grafickým uživatelským rozhraním pro prostředí LispWorks™,
- rozšíření odvozovacího aparátu o základní zpětné řetězení.

Pojem expertního systému spadá do oblasti umělé inteligence. Jde o počítačový systém, který simuluje rozhodování experta nad zvolenou problémovou doménou. Expertní systém může experta zcela nahradit, nebo mu při rozhodování asistovat.

Jazyk Common Lisp[1] (případně jiné dialekty Lispu) je častou volbou pro implementaci umělé inteligence díky svým schopnostem v oblasti symbolických výpočtů (manipulace symbolických výrazů), na nichž řešení těchto problémů často staví. Navíc jde o velmi vysokoúrovňový, dynamicky typovaný jazyk, díky čemuž je programový kód stručný, snadno pochopitelný a tudíž jednoduše rozšiřitelný.

Synax systému CLIPS[2] byla zvolena proto, že jde o reálně používaný systém[3], jehož syntax je Lispu velmi blízká, takže není těžké ji v Lispu napodobit.

Přestože běžnou praxí je začínat diplomovou práci teoretickou částí, definovat jednotlivé pojmy a principy a ty poté v praktické části uplatnit, rozhodl jsem se postupovat opačně, tedy začít práci praktickou částí. Domnívám se totiž (také na základě zkušeností nabytých při vypracování bakalářské práce), že je podstatně snazší (minimálně v řešené problematice) pochopit příklady bez detailní znalosti teorie, než snažit se pochopit teorii bez příkladů, na nichž si lze popisované pojmy a principy představit. V praktické části tedy uvedu jen minimální množtví teorie nutné pro pochopení aktuálního problému, načež se k ní v teoretické části textu vrátím, pojmy zadefinuji přesně a rozšířím o souvislosti.

[1] http://en.wikipedia.org/wiki/Common_Lisp
[2] http://clipsrules.sourceforge.net
[3] http://clipsrules.sourceforge.net/FAQ.html#Q6

2. Praktická část

Tato sekce popisuje knihovnu ExiL[1], která je výsledkem této diplomové práce. Nejprve popíšu její instalaci, pak v uživatelské příručce nastíním základní možnosti a typickou strukturu programu, který ji využívá. Poté v referenční příručce projdu všechny možnosti, které knihovna poskytuje. Načež v části věnované implementaci popíšu architekturu jejího zdrojového kódu a zmíním zajímavé části kódu implementující jednotlivá rozšíření. Nakonec uvedu několik větších příkladů použití knihovny a rozeberu několik dalších možných rozšíření a co by obnášela z pohledu implementace.

2.1. Instalace

2.1.1. Získání zdrojového kódu

Zdrojový kód knihovny je přiložen k této diplomové práci a lze jej také získat zklonováním[2] gitového[3] repozitáře na adrese `git@github.com:Incanus3/ExiL.git`. Kód knihovny se nachází v podadresáři `src`, ten budu dále nazývat kořenovým adresářem knihovny či projektu.

2.1.2. Prerekvizity

Pro práci s knihovnou ExiL potřebujeme lispový interpreter[4] [5], vývojové prostředí (s interpreterem bychom si ve skutečnosti vystačili, ale přímá práce s ním není většinou příliš pohodlná) a knihovny umožňující dávkové načtení celého projektu včetně závislostí.

Knihovnu jsem vyvíjel v prostředí SLIME[6], což je plugin pro textový editor GNU Emacs[7] (poskytující mimo jiné pomůcky pro editaci lispového zdrojového kódu, REPL[8] a debugger[9]) s interpreterem SBCL[10] a tuto kombinaci mohu vřele doporučit. V operačním systému Debian GNU Linux, který jsem pro vývoj použil, lze Emacs, SLIME i SBCL nainstalovat z výchozího repozitáře a aktivovat úpravou inicializačního souboru Emacsu, viz http://www.common-lisp.

[1] TODO: původ názvu

[2] http://git-scm.com/docs/git-clone

[3] http://git-scm.com/

[4] http://en.wikipedia.org/wiki/Interpreter_(computing)

[5] lispové interpretery jsou většinou zároveň kompilátory[6], označením interpreter tedy budu nazývat obojí

[6] http://en.wikipedia.org/wiki/Compiler

[6] http://www.common-lisp.net/project/slime/

[7] http://www.gnu.org/software/emacs/

[8] http://en.wikipedia.org/wiki/Read-eval-print_loop

[9] http://en.wikipedia.org/wiki/Debugger

[10] http://www.sbcl.org/

net/projects/slime/doc/html/Installation.html. Prostředí poté můžeme v Emacsu spustit voláním příkazu `slime` (`M-x slime<enter>`). Při prvním spuštění se kód prostředí kompiluje, což může chvíli trvat, pak už se v editoru otevře buffer[11] s lispovým REPLem.

Knihovnu jsem testoval také ve vývojovém prostředí LispWorks®[12] Personal Edition 6.1, pro které jsem také vytvořil minimalistické grafické uživatelské rozhraní. Součástí prostředí LispWorks je i lispový interpret. Prostředí můžeme nainstalovat podle návodu zde lispworksinstallation.

Pro efektivní načtení knihovny včetně závislostí potřebujeme ještě dvě knihovny:

- ASDF[13] je knihovna umožňující snadnou definici struktury projektu a jeho dávkové načtení,
- quicklisp[14] staví na knihovně ASDF a umožňuje pohodlně stáhnout a načíst knihovny třetích stran z internetové databáze.

Knihovna ASDF je součástí instalace interpreteru SBCL i prostředí LispWorks. Knihovnu quicklisp jsem k projektu přiložil a pokud není součástí prostředí, je automaticky načtena před načtením ExiLu.

2.1.3. Načtení knihovny

V prostředí SLIME načteme knihovnu načtením souboru `load.lisp` z kořenového adresáře knihovny, tedy zadáním

```
(load "cesta/k/projektu/src/load.lisp")
```

v REPLu). Tento soubor nejprve načte knihovnu `quicklisp`, je-li potřeba, a s její pomocí poté načte celý projekt ExiL včetně závislostí. Nakonec soubor definuje výchozí prostředí, viz dále.

V prostředí LispWorks načítání pomocí knihovny `quicklisp` nefunguje správně, knihovnu je proto třeba načítat načtením souboru `load-manual.lisp` (opět z kořenového adresáře projektu). Načíst můžeme opět voláním `load` v REPLu, nebo vybráním položky `Load...` v nabídce `File` menu libovolného okna prostředí.

[11] emacs buffer
[12] http://www.lispworks.com/
[13] http://common-lisp.net/project/asdf
[14] http://www.quicklisp.org/beta/

2.2. Common Lisp

- základní znalosti lispu nutné pro používání knihovny
  - symbol, seznam, atom
  - prefixová syntax, S-expressions
  - funkce, makro
  - načítání souborů.
  - quotování (u funkčních alternativ, ale ty se stejně používají spíš z jiného kódu, který knihovnu volá - tudíž uživatel evidentně lisp zná)
- odkaz na practical common lisp, clhs

2.3. Uživatelská příručka

2.3.1. Základní pojmy

Nyní stručně zadefinuji základní pojmy, nutné pro pochopení fungování knihovny ExiL a práci s ní. K těmto pojmům se posléze v teoretické části textu vrátím a jejich popis rozšířím o další souvislosti.

problémová doména	množina pojmů relevantních pro řešení určité skupiny problémů
fakt	elementární statická znalost - tvrzení
(odvozovací) pravidlo	elementární odvozovací znalost - pokud víme, že (ne)platí nějaká tvrzení, můžeme odvodit, že platí i nějaká další
znalost (v ExiLu)	množina faktů a pravidel
znalostní baze	výchozí znalost
working memory	aktuální množina faktů
production memory	aktuální množina pravidel
inference	odvozování - postupná aplikace odvozovacích pravidel (s případnými zásahy uživatele) (+ výpočet splněných pravidel, jejich výběr)

Pojmy working memory a production memory lze samozřejmě doslova přeložit, ale žádný z překladů mi nepřijde příliš názorný (zvlášť vzhledem k tomu, že ve skutečnosti nejde o paměť, nýbrž aktuální obsah pomyslné paměti), budu tedy používat anglická spojení užívaná v literatuře. Představa pojmů bude jasnější, jakmile si je ukážeme na příkladech.

TODO: dořešit, zda citovat zde, nebo až v teorii

(systém ve skutečnosti pracuje s *reprezentací* znalostí, ale to nás zatím nezajímá)

ExiL, stejně jako CLIPS, rozlišuje dva typy faktů - jednoduché (*simple, ordered*) a strukturované (*templated*). Stuktura jednoduchého faktu je udána pouze pořadím atomů, typickou volbou je např. `objekt-attribut-hodnota`:

```
(box color red),
```

či `relace-<zůčastněné objekty>`:

```
(in box hall).
```

Strukturované fakty mají naproti tomu explicitně pojmenované položky (sloty), typicky tedy popisují objekt s množinou atributů:

```
(box :color red :size small),
```

či relaci s jasně danými aktory:

```
(in :object box :location hall),
```

kde `box` a `in` jsou šablony, které je třeba definovat předem, jak záhy uvidíme. Na pořadí specifikace slotů u strukturovaných faktů pochopitelně nezáleží. Vyjadřovací síla obou skupin faktů je samozřejmě stejná, použitím explicitnějších strukturovaných faktů, ale docílíme lepší čitelnosti a jednoznačnější sémantiky exilového programu, zláště např. v případě relací na jedné množině:

```
(father john george).
```

2.3.2. Struktura programu

Příklad 1 (na straně 10) ukazuje typickou strukturu programu nad knihovnou ExiL (dále exilový program). Tento program používá strukturovaných faktů, začíná tudíž definicemi šablon, tedy formátu těchto faktů. Následuje definice skupiny faktů (jméno `world` této skupiny je pouze informativní). Tyto fakty jsou přidány do znalostní baze, ze které je po volání `(reset)` inicializována *working memory*.

```
;; knowledge structure
(deftemplate goal action object from to)
(deftemplate in object location)

;; initial knowledge
(deffacts world
  (in :object robot :location A)
  (in :object box :location B)
  (goal :action push :object box :from B :to A))

;; inference rules
(defrule move
  (goal :action push :object ?obj :from ?from)
  (in :object ?obj :location ?from)
  (- in :object robot :location ?from)
  ?robot <- (in :object robot :location ?)
  =>
  (modify ?robot :location ?from))

(defrule push
  (goal :action push :object ?obj :from ?from :to ?to)
  ?object <- (in :object ?obj :location ?from)
  ?robot <- (in :object robot :location ?from)
  =>
  (modify ?robot :location ?to)
  (modify ?object :location ?to))

(defrule stop
  ?goal <- (goal :action push :object ?obj :to ?to)
  (in :object ?obj :location ?to)
  =>
  (retract ?goal)
  (halt))

;; initiation
(reset)

;; inference execution
; (step)
(run)
```

Příklad 1: Základní struktura exilového programu

Working memory může být dále v průběhu inference modifikována třemi makry:

- `assert` přidává fakt(a) do working memory,
- `retract` fakt(a) z working memory odebírá a
- `modify` přímo modifikuje existující fakta.

Poté následuje definice tří odvozovacích pravidel - `move`, `push` a `stop`. Definice každého pravidla sestává z množiny podmínek, tedy předpokladů pro jeho splnění (a následnou aktivaci), a množiny důsledků, tedy libovolných lispových výrazů, které jsou při aktivaci pravidla vyhodnoceny. Tyto dvě množiny jsou od sebe odděleny symbolem `=>`. Podmínky pravidel jsou ve formě vzorů (*pattern*). Ty jsou velmi podobné faktům, ale mohou se v nich vyskytovat proměnné (atomy začínající symbolem otazníku). TODO: splnění podmínky pravidla Krom toho mohou být podmínky negovány, tedy podmínka je splněna tehdy, pokud neexistuje žádný fakt, který by jí odpovídal.

2.4. Referenční příručka

2.5. Implementace

3. Teoretická část

3.1. Expertní systémy

Reference

[1] Jackson, P.: *Introduction to Expert Systems*. Addison Wesley, 1998, ISBN 0-201-87686-8.

[2] Norvig, P.: *Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp*. Morgan Kaufmann Publishers, 1991, ISBN 1-55860-191-0.

[3] Doorenbos, R. B.: *Production Matching for Large Learning Systems*. Dizertační práce, Carnegie Mellon University, 1995.
http://reports-archive.adm.cs.cmu.edu/anon/1995/CMU-CS-95-113.pdf

[4] Siebel, P.: *Practical Common Lisp*. Apress, 2005, ISBN 1-59059-239-5.
http://www.gigamonkeys.com/book/

[5] CLIPS: *Tool for Building Expert Systems*. 2013.
http://clipsrules.sourceforge.net/OnlineDocs.html

[6] LispWorks Ltd.: *Common Lisp HyperSpec*. 2005.
http://www.lispworks.com/documentation/HyperSpec/Front/

[7] Expert system — Wikipedia, The Free Encyklopedia. 2013.
http://en.wikipedia.org/wiki/Expert_system

[8] Rete algorithm — Wikipedia, The Free Encyklopedia. 2013.
http://en.wikipedia.org/wiki/Rete_algorithm