Logic Programming and Clojure

Yang Shouxun

June 23, 2013

Clojure China User Group, Beijing

Outline

- Logic
- Logic Programming
- Prolog
- core.logic

Logic

- Propositional logic
  - Statement
  - Truth: true or false
  - $\neg p$
  - $p \land q$
  - $p \lor q$
  - $p \to q$

Logic

- Propositional logic
  - syllogism

$$\begin{array}{lll} p \to q & \qquad & p \to q \\ p & & \neg q \\ \text{-----------} & & \text{---------} \\ q & & \neg p \end{array}$$

Logic

- Predicate logic
  - $p(x)$, $q(x)$
  - universal quantifier

    $\forall x.p(x)$
  - existential quantifier

    $\exists x.p(x)$

Logic

- Horn clause
  - a disjunction of literals with at most one postive literal
    - definite clause: exactly one postive:

      $\neg p \lor \neg q \lor \ldots \lor \neg t \lor u$
    - fact: u
    - goal clause: $\neg p \lor \neg q \lor \ldots \lor \neg t$

Logic Programming

- PROLOG by Colmerauer in 1973
- Theoretical work by Kowalski in 1974

$$(p \wedge q \wedge \ldots \wedge t) \rightarrow u$$

$$u \leftarrow (p \wedge q \wedge \ldots \wedge t)$$

- Datalog
- Oz
- Mercury
- miniKaren
- core.logic/mini-kanren in Clojure

Prolog

- Relational databases
- Mathematical logic
- Abstract problem solving
- Natural language processing
- Artificial Intelligence

Prolog

- Facts
  - father(terach,abraham).
  - father(abraham,issac).
- Questions
  - ?- father(terach,abraham).
- Variables
  - ?- father(abraham,X).
  - ?- father(X,abraham).

Prolog

- Conjunctions
  - ?- father(terach,abraham), father(abraham,issac).
- Rules
  - grandfather(X,Y) :- father(X,Z), father(Z,Y).
  - ?- grandfather(terach,X).
  - ?- grandfather(X,issac).
- Structures and trees
  - book(moby_dick, author(herman, melville))
- Lists
  - [a, b, c]

Prolog

- Recursion
  - member
- Difference structures
- Backtracking and the cut
  - foo :- a, b, !, c.

Prolog

- non-relational operators
  - cut (!)
  - var/1
  - copy_term/2

core.logic

- project.clj

```
:dependencies [[org.clojure/clojure "1.5.1"]
               [org.clojure/core.logic "0.8.3"]]
```

- clojure code using core.logic

```
(ns yourcode.core
  (:refer-clojure :exclude [==])
  (:use [clojure.core.logic]))
```

core.logic

```
(defrel father p1 p2)


(fact father 'Terach 'Abraham)
(fact father 'Terach 'Nachor)
(fact father 'Terach 'Haran)
(fact father 'Abraham 'Issac)
(fact father 'Haran 'Lot)
(fact father 'Haran 'Milcah)
(fact father 'Haran 'Yiscah)
```

- defrel
- fact

core.logic

```
(run* [x]
    (father 'Abraham 'Issac))

(run* [_]
    (father 'Abraham 'Issac))

(run* [_]
    (father 'Abraham 'Haran))
```

- queries: yes or no
- wildcards

core.logic

```
;; father(abraham,X)?
(run* [x]
      (father 'Abraham x))

;; father(X,haran)?
(run* [x]
      (father x 'Haran))
```

- queries: who
- bidirectional

core.logic

```
(defrel male p)


(fact male 'Terach)
(fact male 'Abraham)
(fact male 'Nachor)
(fact male 'Haran)
(fact male 'Issac)
(fact male 'Lot)
```

core.logic

```
;; father(haran,X), male(X)?
(run* [x]
      (father 'Haran x)
      (male x))

;; father(terach,X), father(X,Y)?
(run* [x y]
      (father 'Terach x)
      (father x y))
```

- conjunction
- variable scope

core.logic

```
;; son(X,Y) <- father(Y,X), male(X).
(defn son
  [x y]
  (fresh [_]
         (father y x)
         (male x)))

;; son(lot,haran)?
(run* [_]
      (son 'Lot 'Haran))
```

- rules

core.logic

```
;; grandparent(X,Y) <- parent(X,Z), parent(Z,Y).
(defn grandparent  [x y]
  (fresh [z]
         (parent x z)
         (parent z y)))


(run* [x]
      (grandparent 'Terach x))


(run* [x]  (grandparent x 'Milcah))
```

- rules

core.logic

```
(defrel female p)

(fact female 'Sarah)
(fact female 'Milcah)
(fact female 'Yiscah)

(defrel mother p1 p2)

(fact mother 'Sarah 'Issac)
```

core.logic

```
;; parent(X,Y) <- father(X,Y).
;; parent(X,Y) <- mother(X,Y).
(defn parent
  [x y]
  (fresh [_]
         (conde
          [(father x y)]
          [(mother x y)])))
```

- alternative rules

core.logic

```
;; ancestor(Ancestor, Descendant) <-  parent(Ancestor, Descendant).
;; ancestor(Ancestor, Descendant) <-  parent(Ancestor, Person), ancestor(Person, Descendant)
(defn ancestor  [anc desc]
  (conde
   [(parent anc desc)]
   [(fresh [p]
          (parent anc p)
          (ancestor p desc))]))

(run* [x]  (ancestor x 'Yiscah))
```

- recursive rules

core.logic

```
(run* [x]
    (conda
     [(== 'Olive x)]
     [(== 'Oil x)]))

(run* [x]
    (conda
     [(== 'Virgin x) (== true false)]
     [(== 'Olive x)]
     [(== 'Oil x)]))
```

- soft cut

```
(run* [q]
    (condu
     [(== true false)]
     [alwayso])
    (== true q))
```

Logic Puzzles

1. There are five houses in a line, each with an owner, a pet, a cigarette, a drink, and a color.
2. The Englishman lives in the red house.
3. The Spaniard owns the dog.
4. Coffee is drunk in the green house.
5. The Ukrainian drinks tea.
6. The green house is immediately to the right of the ivory house.
7. The Winston smoker owns snails.
8. Kools are smoked in the yellow house.
9. Milk is drunk in the middle house.
10. The Norwegian lives in the first house on the left.
11. The man who smokes Chesterfields lives next to the man with the fox.
12. Kools are smoked in the house next to the house with the horse.
13. The Lucky Strike smoker drinks orange juice.
14. The Japanese smokes Parliaments.
15. The Norwegian lives next to the blue house.

The questions to be answered are: who drinks water and who owns the zebra?

CLP(fd)

```
   send
+ more
---------
 money
```

Reading List

- The Art of Prolog, 2nd ed.
- Programming in Prolog, 5th ed.
- Concepts Techniques and Models of Computer Programming
- The Reasoned Schemer
- Relational Programming in miniKanren: Techniques, Applications, and Implementations