logic-tutorials-kr

출시 0.1

Seonho Kim

2016년 04월 02일

Contents

클로저와 클로저 스크립트를 위한 논리 프로그래밍 라이브러리인 core.logic은 관계형 프로그래밍, 제약 논리 프로그래밍과 같은 Prolog 언어와 유사한 기능을 제공한다. 이 라이브러리는 William Byrd의 논문 Relational Programming in miniKanren: Techniques, Applications, and Implementations 에 기술된 miniKanren을 그 기반으로 삼고 있으며, cKanren 과 αKanren 까지도 확장하고 있다. 또한 단순히 이들이 제공하는 기능을 넘어서 논리 프로그래밍을 쉽게 확장하는 것을 목표로 설계되었다.

(원문 출처: core.logic - README.md)

CHAPTER 1

릴리즈 및 의존성 정보

- 최신 안정 릴리즈(latest stable release): 0.8.10
  - 다른 릴리즈 버전
  - 개발 스냅샷 버전
- 라이닝언(Leiningen) 의존성 정보:

```
<dependency>
        <groupId>org.clojure</groupId>
        <artifactId>core.logic</artifactId>
        <version>0.8.10</version>
</dependency>
```

CHAPTER 2

튜토리얼 구성 개요

전체적으로는 다음의 세 파트로 구성되며, core.logic wiki, swannodette/logic-tutorial, frenchy64/Logic-Starter 의 내용을 각 파트에 맞게 재배치하고 필요에 따라 내용을 수정하여 구성한다.

- **파트 구성**
  - *core.logic 개요*
  - *core.logic 시작하기*
  - *기타 읽을거리*

2.1 core.logic 특징

예제 소스코드: src/logic_wiki/features.clj

2.1.1 단순한 인메모리 데이터베이스

때로는 질의 대상이 되는 사실(fact)들을 리스트로 생성하는 것이 유용할 수 있다. 이 때 `defrel` 과 `fact` 를 사용한다.

```
(ns logic-wiki.features
  (:use [clojure.core.logic])
  (:require [clojure.core.logic.pldb :as pldb]))

(pldb/db-rel man p)
(pldb/db-rel woman p)
(pldb/db-rel likes p1 p2)
(pldb/db-rel fun p)

(def facts0
  (pldb/db
   [man 'Bob]
   [man 'John]
   [man 'Ricky]

   [woman 'Mary]
   [woman 'Martha]
   [woman 'Lucy]

   [likes 'Bob 'Mary]
   [likes 'John 'Martha]
```

```
   [likes 'Richy 'Lucy]))

(def facts1 (-> facts0 (pldb/db-fact fun 'Lucy)))

(pldb/with-db facts1
  (run* [q]
    (fresh [x y]
      (fun y)
      (likes x y)
      (== q [x y]))))
;;=> ([Richy Lucy])
```

여기서 중요한 것은 사실(fact)의 갯수가 늘어남에 따라 질의 처리 시간이 함께 증가하지 않도록 관계(relationship)들을 인덱싱하는 것이다. 사실(fact) 튜플의 요소들을 위한 인덱스들을 생성해둘 수 있는데, 메모리를 소비하게 된다는 점은 주의해야 한다.

```
(pldb/db-rel likes ^:index p1 ^:index p2)
```

2.1.2 개별적 단일화(unification)

core.logic를 다룰 때는 core.unify와 상당히 유사하게 사용되는 단일화(unification)가 수반된다.

```
(unifier ['(?x ?y ?z) '(1 2 ?y)])
;;=> {?y 2, ?x 1, ?z 2}

(unify ['(?x ?y ?z) '(1 2 ?y)])
;;=> (1 2 2)

(run* [?x ?y ?z]
    (== [?x ?y ?z] [1 2 ?y]))
;;=> ([1 2 2])
```

2.2 제약 논리 프로그래밍

core.logic은 줄여서 CLP라 부르는 제약 논리 프로그래밍(Constraint Logic Programming)의 여러 유형들을 빠르게 지원해가고 있다. core.logic은 제약 도메인을 확장할 수 있도록 설계되었다. 특히 CLP(Tree)라 불리는 트리에 대한 비동일성 제약(disequality constraint)이나 CLP(FD)라 부리는 무한 도메인에 대한 제약도 지원하고 있다.

2.2.1 CLP(Tree)

CLP(Tree)는 아주 단순한데, != 연산자 단 한개만 추가된다. 주어진 두 항에 != 연산자를 사용하면 두 항이 절대 단일화될 수 없음을 나타내며, 이는 단일화 연산자 == 와 반대되는 의미라고 할 수 있다.

이 연산자를 사용하는 가장 간단한 예는 한 항이 어떤 값과 같지 않음을 확인하는 것이다.

```
(run* [q]
    (!= q 1))
;;=> ((_0 :- (!= (_0 1))))
```

특정한 값이 주어지지 않은 변수가 제약을 갖게 되면 위와 같이 이상한 값이 출력된다. 이 결과는 q(_0)에 1이 아닌 어떤 값이라도 올 수 있다고 해석할 수 있다.

물론 훨씬 더 복잡한 항들에 대해서도 비동일성 제약을 적용할 수 있다.

```
(run* [q]
    (fresh [x y]
    (!= [1 x] [y 2])
    (== q [x y])))
;;=> (([_0 _1] :- (!= (_1 1) (_0 2))))
```

이 코드의 의미는 얼핏 보고 생각했던 것과 좀 다를 수 있다. 이 코드는 "x가 2이면서(AND) y는 1이면 안된다"로 해석해야 한다. 따라서 y 가 3이면 제약 조건 전체를 폐기할 수 있다(x에 어떤 값이라도 올 수 있다). 그러나 y 가 1이면 제약 조건은 x 가 2가 되지 않는지 계속 확인하게 된다.

2.2.2 CLP(FD)

2.3 클로저로 논리 프로그래밍 해보기

2.4 Reasoned Schemer와의 차이점