神奈川大学理学部 情報科学科 馬谷研究室

論理型言語 miniKanren とホスト言語の間の連携機能の提案

二村 忠輝 202103337

1 はじめに

本研究で扱う miniKanren とは論理型言語として関係プログラミングに焦点を当てたものであり, 非決定的な論理推論を通じて複数の解を同時に探索することにより, 柔軟で豊かな表現で解を生成することのできるものである. 関係を用いることでユーザーは具体的な値ではなく未知の変数を与えることが可能になり, 可能性のある解を全て列挙させるといった表現方法を扱うことも可能となる. 本研究では関数型プログラミング言語である Racket 言語をホスト言語とする miniKanren の実装において定義される関係と Racket 関数との相互連携機能を構築することでユーザ側の利便性の向上を図る.

2 miniKanren 言語

miniKanren 言語は論理型のプログラミング言語であり, 関係の定義とクエリの実行を用いてプログラムを論理的に表現することができる. 以下にプログラムの一例と実行結果を示す.

```
(defrel (car° p a)
  (fresh (d)
    (== (cons a d) p)))

(car '(a b c d))
>a
(car° '(a b c d) 'a)
>#s
(car° '(a b c d) 'b)
>#u
```

car は引数としてリストを受け取り, 最初の要素が返り値となる. 一方 car° は引数としてリストまたはシンボルを 2 つ受け取り, 第 2 引数が第 1 引数の最初の要素と一致していたならこの関係は#s(success) となり, 一致しない場合この関係は#u(unsuccess) となる.(car° (a b c) 'b) の例では関係を満たさないため失敗となるのが確認できる.

```
(run* q
 (car° '(a b c d) q))
> (a)
```

またこのように関係では引数として未定義の変数を与えることも可能であり, その場合未定義な変数に解として可能な結果を束縛し, その結果のリストが返り値となる.

3 設計

本研究では関数型プログラミングによって定義された関数と変数を論理型プログラミング上で動作可能な形にする関数 r → m と, 論理型プログラミング上で関連付けられた関係と変数を関数型プログラミング上で動作可能とする関数 m → r を実装する. それぞれの機能の動作は以下のようになる.

- r → m は引数を 2 つ期待する. 第 1 引数に関数または変数を受け取り第 1 引数として関数を受け取った場合. 第 2 引数に未束縛の変数を受け取り追加引数として関数の式を受け取る. 動作としては第 2 引数の変数に関数の式の結果を束縛したリストを生成する. 第 1 引数に変数を受け取った場合. 第 2 引数に変数に定義済みの値を受け取り束縛したリストを生成する.
- m → r は引数を 1 つ期待する. 引数として関係を受けとった場合, 追加引数として関係式を受け取り, その関係式が成功するなら#t, 失敗するなら#f を生成する. 引数として変数を受け取った場合, 式は変数名の box を生成する.

実行結果として以下のような結果が出力される.

```
(run* x
 ((r->m car x)
  (lambda(q) (q '(a b c)))))
>(a)
(define p 10)
(run* x
 (== (r->m p 10) x))
>(10)
```

```
((m->r ==)
 (lambda (q) (q 'rm 'rm)))
>#t
((m->r ==)
 (lambda (q) (q 'rm 'mr)))
>#f
(m->r x)
>x
```

4 実装

ここでは実装した r → m と m → r について説明を行う. まず実装を行うに際して用いた関数について説明する.

- `(unify x y empty-s)`: 引数 x,y に対して束縛を行う.`(変数 x . y)` または `(変数 y . x)` という形で束縛を生成する.x,y がともに値の場合`#f` となる.
- `(== x y)`:`unify` に x,y を渡し, その結果で束縛を含むリストか空リストを結果として返す.
- `(walk* x y)`: 引数として変数 x と束縛のリスト y を受け取り,y の中の x に束縛されている値を探索し結果として返す.

以下が関数 r → m の定義である.

```
(define (r->m p x)
  (cond
    ((procedure? p) (lambda (q)
                      (== x (q p))))
    (else (let ((p (var 'p)))
            (let ((q (unify p x empty-s)))
              (walk* p q))))))
```

まず p が関数であるかどうかの判定を `procedure?`関数を用いて行う.p が関数の場合式は lambda 式を返す.lambda 式は引数の x と q に関数 p を適用したものをもちいて束縛を生成する.p が変数の場合 `let` を用いて p に値 p を束縛し,p に値 x を束縛したものを生成しそれを q に束縛する.`(walk* p q)` にて p に束縛している値 x を結果として返す.r → m は `unify` を用いて束縛を生成することで関係のような挙動をするように実装している. 次に関数 m → r の定義を説明する. 以下が関数 m → r の定義である.

```
(define (m->r p)
  (cond
    ((procedure? p) (lambda (q)
                      (if (null? (q p))
                          #f
                          #t)))
    (else (box p))))
```

r → m の定義と同様 `procedure?`関数を用いて p が関係か変数かの判定を行う.p が関係の場合は式は lambda 式を返す.lambda 式は q に関係 p を適用した結果束縛が生成されたかどうかを `null?`で判定し, 束縛が生成されている場合に関係は満たされていないとして`#f`, 束縛が生成されず空リストの場合`#t` を返す.p が変数の場合は式は p を初期値とする `box` を生成する.

5 まとめ

本研究では論理型言語である miniKanren とホスト言語である Racket 言語の相互連携機能を構築, 実装を行った. しかし今回の実装では未束縛変数の表現を実装することが不十分であったため, これが今後の課題となる.

参考文献

[1] Daniel P.Friedman, William E. Byrd, Oleg Kiselyov, and Jason Hemann *The Reasoned Schemer Second Edition,* 2016 The MIT Press.