Resolution and Unification in Logic Programming

Michael McThrow

San Jose State University
Computer Science Department
CS 152 – Programming Paradigms

October 26, 2020

Table of Contents

Table of Contents

Ground and Non-ground Terms

- Terms that lack variables are ground.
- Terms that contain variables are non-ground.

Examples of Ground and Non-ground Terms

- Ground Terms:
  - `evolution(bulbasaur, ivysaur).`
  - `fire(charmander).`
  - `plus(s(s(0)), s(s(s(0))), s(s(s(s(s(0)))))).`
- Non-ground Terms:
  - `evolution(ivysaur, X)?`
  - `plus(0, X, 0).`

Table of Contents

Definition of Resolution

Definition (Resolution)

"Resolution is the process of matching facts and rules to perform *inferencing*, the derivation of logical conclusions from the rules" [from Professor Tom Austin's Spring 2019 CS 152 slides].

When we issue a query, the interpreter *resolves* it.

The simple case is when we are issuing ground queries; i.e., queries with no variables such as `evolution(bulbasaur, ivysaur)?`.

How do we resolve ground queries?

Resolution Algorithm for Ground Queries and Terms

Input: A ground goal G and a program P

Output: *yes* if G is a logical consequence of P,
no otherwise

Algorithm: Initialize the resolvent to G.
while the resolvent is not empty *do*
  choose a goal A from the resolvent
  choose a ground instance of a clause $A' \leftarrow B_1, \ldots, B_n$ from P
    such that A and A' are identical
   (if no such goal and clause exist, exit the while loop)
  replace A by $B_1, \ldots, B_n$ in the resolvent
If the resolvent is empty, *then* output *yes*, *else* output *no*.

Figure 1.1 An abstract interpreter to answer ground queries with respect to logic programs

Figure: Resolution algorithm for ground queries and terms [Sterling and Shapiro 1994, p. 22]

Let's go to the whiteboard for an example.

Each iteration of the while loop in the resolution algorithm is called a *reduction*.

Definition of a Reduction

Definition (Reduction)

"A reduction of a goal G by a program P is the replacement of G by the body of an instance of a clause in P, whose head is identical to the chosen goal" [Sterling and Shapiro, p. 23].

This resolution algorithm works for ground queries given a program consisting of ground rules. But how do we deal with non-ground queries and rules?

Table of Contents

Common Instance

Definition (Common Instance)

"A term t is a common instance of two terms t_1 and t_2, if there exist substitutions θ_1 and θ_2 such that t equals $t_1\theta_1$ and $t_2\theta_2$" [Sterling and Shapiro, p. 88].

Definition of "More General"

Definition

"A term *s* is more general than a term *t* is *t* is an instance of *s* but *s* is not an instance of *t*" [Sterling and Shapiro, p. 88].

Definition of "More General"

Definition

"A term *s* is more general than a term *t* is *t* is an instance of *s* but *s* is not an instance of *t*" [Sterling and Shapiro, p. 88].

For example, `fire(X)` is more general than `fire(charizard)`, since `fire(charizard)` is an instance of `fire(X)`, but `fire(X)` is not an instance of `fire(charizard)`.

Alphabetic Variants

Definition (Alphabetic Variant)

"A term s is an alphabetic variant of a term t if both s is an instance of t and t is an instance of s" [Sterling and Shapiro, p. 88].

Alphabetic Variants

Definition (Alphabetic Variant)

"A term *s* is an alphabetic variant of a term *t* if both *s* is an instance of *t* and *t* is an instance of *s*" [Sterling and Shapiro, p. 88].

For example, these terms are alphabetic variants of each other:

- `member(X, tree(Left, X, Right))`
- `member(Y, tree(Left, Y, Z))`

Unification

- "A unifier of two terms is a substitution making the terms identical. If two terms have a unifier, we say they unify" [Sterling and Shapiro, p. 88].
- "Any unifier determines a common instance, and conversely, any common instance determines a unifier" [p. 88].

Unification

- "A unifier of two terms is a substitution making the terms identical. If two terms have a unifier, we say they unify" [Sterling and Shapiro, p. 88].
- "Any unifier determines a common instance, and conversely, any common instance determines a unifier" [p. 88].

Example: Given the terms `append([1,2,3],[3,4],List)` and `append([X|Xs],Ys,[X|Zs])`, the unifying substitution is `X=1, Xs=[2,3], Ys=[3,4], List=[1,Zs]` and the common instance is `append([1,2,3],[3,4],[1|Zs])`.

Most General Unifier (MGU)

Definition (Most General Unifier)

"A most general unifier, or MGU, of two terms is a unifier such that the associated common instance is most general" [p. 88].

Most General Unifier (MGU)

Definition (Most General Unifier)

"A most general unifier, or MGU, of two terms is a unifier such that the associated common instance is most general" [p. 88].

The goal of unification is to identify the MGU of two terms. The resolution algorithm will use the MGU when resolving queries.

Unification Algorithm [Sterling and Shapiro 1994, p. 90]

Algorithm: Initialize the substitution θ to be empty,
the stack to contain the equation $T_1 = T_2$,
and failure to *false*.

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in θ
add $X = Y$ to θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in θ
add $Y = X$ to θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i, i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output θ.

Let's go to the whiteboard for an example.

Table of Contents

Resolution with Unification

Input: A goal G and a program P

Output: An instance of G that is a logical consequence of P, or *no* otherwise

Algorithm: Initialize the resolvent to G.

while the resolvent is not empty *do*

  choose a goal A from the resolvent

  choose a (renamed) clause $A' \leftarrow B_1,\ldots,B_n$ from P

    such that A and A' unify with mgu θ

   (if no such goal and clause exist, exit the *while* loop)

  replace A by $B_1,\ldots,B_n$ in the resolvent

  apply θ to the resolvent and to G

If the resolvent is empty, *then* output G, *else* output *no*.

Figure 4.2 An abstract interpreter for logic programs

Figure: Resolution algorithm with unification [Sterling and Shapiro 1994, p. 93]

Table of Contents

In the past few lectures, we've been going over pure logic programming with Prolog syntax. Beginning Wednesday, we will begin covering some more concrete elements of Prolog, including arithmetic and its execution model.

Reading Assignment

Please read chapters 6, 7, and 8 of *The Art of Prolog*.