Control in Prolog

CS3100 Fall 2019

Review

Previously

- Programming with Lists, Arithmetic, Backtracking, Choice Points

This lecture

- Control in Prolog
  - Rule order and Goal order
  - An abstract interpreter for logic programs
    - Unification, Substitution

Algorithm = Logic + Control

- Logic: facts, rules and queries
- Control: how prolog chooses the rules and goals, among several available options.

There are two main control decisions: **Rule Order & Goal Order.**

Algorithm = Logic + Control

- **Rule order**: Given a program with a collection of facts and rules, in which order do you choose to pick rule to unify.
  - SWI-Prolog chooses the **first** applicable rule in the order in which they appear in the program.
- **Goal order**: Given a set of goals to resolve, which goal do you choose
  - SWI-Prolog: chooses the **left-most** subgoal.

Rule order and goal order influences program behaviour.

Substring in Prolog

```
<-------X-------->
+--------------------------------+
|        |     S     |                |
+--------------------------------+
<--------------Z---------------->
```

We can specify this is seemingly equivalent ways.

- prefix X of Z and suffix S of X.
- suffix S of X and prefix X of Z.

Substring in Prolog

The corresponding prolog queries are:

In [1]:

```
append([],Q,Q).
append([H | P], Q, [H | R]) :- append(P, Q, R).
prefix(X,Z) :- append(X,Y,Z).
suffix(Y,Z) :- append(X,Y,Z).
```

```
Added 4 clauses(s).
```

Substring in Prolog

They usually produce the same result:

In [2]:

```
?- prefix([a,b,c],[a,b,c,d]), suffix(S,[a,b,c]).
```

```
S = [ a, b, c ] ;
S = [ b, c ] ;
S = [ c ] ;
S = [  ] .
```

In [3]:

```
?- suffix(S,[a,b,c]), prefix([a,b,c],[a,b,c,d]).
```

```
S = [ a, b, c ] ;
S = [ b, c ] ;
S = [ c ] ;
S = [  ] .
```

Substring in Prolog

Their answers however differ in other cases:

In [4]:

```
?- prefix(X,[b]), suffix([a],X).
```

```
false.
```

In [5]:

```
?- suffix([a],X), prefix(X,[b]).
```

```
ERROR: Caused by: '  suffix([a],X), prefix(X,[b])'. Returned: 'error(r
esource_error(stack), dict(stack_overflow, 1, choicepoints, 23646806,
depth, 4, environments, 738963, globalused, 0, localused, [Functor(532
877,3,23646806,:(user, append(_262, [1], _266)),[]), Functor(532877,3,
2,:(system, <meta-call>(<garbage_collected>)),[]), Functor(532877,3,
1,:(user, pyrun(<garbage_collected>, [1])),[]), Functor(532877,3,0,:(s
ystem, $c_call_prolog),[])], stack, 1048576, stack_limit, 0, trailuse
d))'.
```

Goal order changes solutions

Consider the query:

In [6]:

```
?- suffix([a],L), prefix(L,[a,b,c]) {1}.
```

```
L = [ a ] .
```

In [7]:

```
?- suffix([a],L), prefix(L,[a,b,c]) {2}.
```

```
ERROR: Caused by: '  suffix([a],L), prefix(L,[a,b,c]) '. Returned: 'er
ror(resource_error(stack), dict(stack_overflow, 1, choicepoints, 23646
800, depth, 4, environments, 738963, globalused, 0, localused, [Functo
r(532877,3,23646800,:(user, append(<garbage_collected>, [1], _296)),
[]), Functor(532877,3,2,:(system, <meta-call>(<garbage_collected>)),
[]), Functor(532877,3,1,:(user, pyrun(<garbage_collected>, [1])),[]),
Functor(532877,3,0,:(system, $c_call_prolog),[])], stack, 1048576, sta
ck_limit, 0, trailused))'.
```

Exercise: Trace by hand.

Goal order changes solutions

Consider the query:

In [8]:

```
?- prefix(L,[a,b,c]), suffix([a],L).
```

```
L = [ a ] .
```

has precisely one answer.

Exercise: Trace by hand.

Rule order affects the search for solutions

Consider the definition `appen2` which reorders the rules from `append` .

In [9]:

```
appen2([H | P], Q, [H | R]) :- appen2(P, Q, R).
appen2([],Q,Q).
```

```
Added 2 clauses(s).
```

Rule order affects the search for solutions

Consider the query:

In [10]:

```
?- append(X,[c],Z) {5}.
```

```
X = [  ], Z = [ c ] ;
X = [ _520 ], Z = [ _520, c ] ;
X = [ _520, _532 ], Z = [ _520, _532, c ] ;
X = [ _520, _532, _544 ], Z = [ _520, _532, _544, c ] ;
X = [ _520, _532, _544, _556 ], Z = [ _520, _532, _544, _556, c ] .
```

In [11]:

```
?- appen2(X,[c],Z) {1}.
```

```
ERROR: Caused by: '  appen2(X,[c],Z) '. Returned: 'error(resource_erro
r(stack), dict(stack_overflow, 4194188, choicepoints, 4194189, depth,
4194190, environments, 196603, globalused, 753643, localused, [Functor
(532877,3,4194189,:(user, appen2(_178, [1], _182)),[]), Functor(53287
7,3,4194188,:(user, appen2([1], [1], [1])),[])], non_terminating, 1048
576, stack_limit, 65534, trailused))'.
```

goes down an infinite search path.

Exercise: Trace by hand.

Occurs check problem

Consider the query

In [ ]:

```
?- append([],E,[a,b|E]).
```

goes down an infinite search path.

Exercise: Trace by hand to verify why.

Occurs check problem

Occurs check problem

Consider the query

```
?- append([],E,[a,b | E]).
```

- In order to unify this with, `append([],Y,Y)` , we will unify `E = [a,b | E]` , whose solution is `E = [a,b,a,b,a,b,...]`.
- In the name of efficiency, most prolog implementations do not check whether `E` appears on the RHS term.
  - infinite loop on unification.
- Some versions of prolog creates cyclic terms (like OCaml recursive values), but most don't.

Occurs check problem

You can explicitly turn on occurs check in SWI Prolog.

In [1]:

```
?- set_prolog_flag(occurs_check,true).
```

```
true.
```

In [2]:

```
?- append([],E,[a,b | E]).
```

```
false.
```

Occurs check problem

You can explicitly turn occurs check in SWI Prolog to an **error**.

In [3]:

```
?- set_prolog_flag(occurs_check,error).
```

```
true.
```

In [4]:

```
?- append([],E,[a,b | E]).
```

```
ERROR: Caused by: '  append([],E,[a,b | E])'. Returned: 'error(occurs_
check(_1792, [Atom('282629'), Atom('222853')]), context(:(lists, /(app
end, 3)), _1812))'.
```

Abstract interpreter for logic programs

We can precisely define the influence of rule and goal orders by describing an **abstract interpreter** for logic programs.

First, we will start off with some definitions of ideas that we have informally seen earlier.

Substitution

A substitution is a finite set of pairs of terms $\{X_1/t_1, \ldots, X_n/t_n\}$ where each t_i is a term and each X_i is a variable such that $X_i \neq t_i$ and $X_i \neq X_j$ if $i \neq j$.

The empty substitution is denoted by ϵ.

For example, $\sigma = \{X/[1,2,3], Y/Z, Z/f(a,b)\}$ is substitution.

Quiz

Is this a valid substitution?

$$\{X/Y, Y/X, Z/Z, A/a1, A/a2, m/n\}$$

Quiz

Is this a valid substitution?

$$\sigma = \{X/Y, Y/X, Z/Z, A/a1, A/a2, m/n\}$$

No.

- Z/Z should not be in σ.
- Variable A has two substitutions $A/a1$ and $A/a2$, which is incorrect.
- m/n is not a valid substitution; m should be a variable.
- $X/Y, Y/X \in \sigma$ is fine.

Application of substitution

The application of substitution σ to a variable X, written as $X\sigma$ is defined

$$X\sigma = \begin{cases} t \text{ if } X/t \in \sigma \\ X \text{ otherwise} \end{cases}$$

Application of substitution

Let σ be a substitution $\{X_1/t_1, \ldots, X_n/t_n\}$ and E a term or a formula. The application $E\sigma$ of σ to E is obtained by **simultaneously** replacing every occurrence of X_i in E with t_i.

Given $\sigma = \{X/[1,2,3], Y/Z, Z/f(a,b)\}$ and $E = f(X,Y,Z)$, $E\sigma = f([1,2,3], Z, f(a,b))$.

Now, $E\sigma$ is known as an **instance** of E.

Composition of substitutions

Consider two substitutions θ and σ. Then, the composition is defined as $\theta\sigma$. Intuitively, we will apply the substitution θ before σ in $\theta\sigma$.

Consider $\theta = \{X/Y, Z/a\}$ and $\sigma = \{Y/X, Z/b\}$. Then, $\theta\sigma = \{Y/X, Z/a\}$.

Let $\theta = \{X_1/s_1, \ldots, X_n/s_n\}$ and $\sigma = \{Y_1/t_1, \ldots, Y_n/t_n\}$ be two substitutions. The composition $\theta\sigma$ is the set obtained from the set:

$$\{X_1/s_1\sigma, \ldots, X_n/s_n\sigma, Y_1/t_1, \ldots, Y_n/t_n\}$$

- by removing all $X_i/s_i\sigma$ for which X_i is syntactically equal to $s_i\sigma$ and
- by removing those Y_i/t_i for which Y_i is the same as some X_j.

Properties of substitutions

Let θ, σ and γ be substitutions, ϵ be empty substitution, and let E by a term or a formula. Then:

- $E(\theta\sigma) = (E\theta)\sigma$
- $(\theta\sigma)\gamma = \theta(\sigma\gamma)$
- $\epsilon\theta = \theta\epsilon = \theta$.
- $\theta = \theta\theta$ iff $Dom(\theta) \cap Range(\theta) = \emptyset$.

In general, composition of substitutions is not commutative. For example,

$$\{X/f(Y)\}\{Y/a\} = \{X/f(a), Y/a\} \neq \{Y/a\}\{X/f(Y)\} = \{Y/a, X/f(Y)\}$$

Unifer

Let s and t be two terms. A substitution σ is a unfier for s and t if $s\sigma$ and $t\sigma$ are syntactically equal.

Let $s = f(X, Y)$ and $t = f(g(Z), Z)$. Let $\sigma = \{X/g(Z), Y/Z\}$ and $\theta = \{X/g(a), Y/a, Z/a\}$. Both σ and θ are unfiers for s and t.

A substitution is σ is more general than another substitution θ if there exists a substitution ω such that $\theta = \sigma\omega$.

In the previous example, $\theta = \sigma\{Z/a\}$. Hence, σ is more general than θ.

Most general unifier

A unfier is said to be the most general unfier (mgu) of two terms if it is more general than any other unfier of the terms.

A pair of terms may have more than one most general unifier. For example, for the terms $f(X, X)$ and $f(Y, Z)$, the unifiers $\theta = \{X/Y, Z/Y\}$ and $\sigma = \{X/Z, Y/Z\}$ are both most general unifier.

$\theta = \sigma\{Z/Y\}$ and $\sigma = \theta\{Y/Z\}$.

If the unfiers θ and σ are both mgus, then there is a substitution $\gamma = \{X_1/Y_1, \ldots, X_n/Y_n\}$ where X_i and Y_i are variables such that $\theta = \sigma\gamma$.

Intuitively, θ can be obtained from σ by **renaming** the variables.

Quiz

What is the mgu of $f(X, X, Y)$ and $f(Y, Z, a)$

1. $\{X/a, Y/a, Z/a\}$
2. $\{X/b, Y/b, Z/b\}$
3. $\{X/Y, Z/Y\}$
4. ϵ

Quiz

What is the mgu of $f(X, X, Y)$ and $f(Y, Z, a)$

1. $\{X/a, Y/a, Z/a\}$ ✓
2. $\{X/b, Y/b, Z/b\}$
3. $\{X/Y, Z/Y\}$
4. ϵ

Algorithm for computing mgu

Input: Two terms T_1 and T_2 to be unified

Output: θ, the mgu if T_1 and T_2 or *FAIL*.

Algorithm: $mgu(T_1, T_2)$.

```
Initialise
  Substitution θ = ∅,
  Stack Σ to T1 = T2,
  Failed = false.
while (Σ not empty && not Failed) {
  pop X = Y from Σ
  case
    X is a variable that does not occur in Y:
      substitute Y for X in Σ and in θ
      add X/Y to θ
    Y is a variable that does not occur in X:
      substitute X for Y in Σ and in θ
      add Y/X to θ
    X and Y are indentical constants or variables:
      continue
    X is f(X1,...,Xn) and Y is f(Y1,...,Yn):
      push Xi = Yi, i=1 to n to Σ
    otherwise:
      Failed = true
}
If Failed = true, then return FAIL else return θ
```

Trace

```
mgu(f(X,X,Y),f(Y,Z,a))
```

Initially, $\theta = \emptyset$, $\Sigma = [f(X, X, Y) = f(Y, Z, a)]$, Failed = false.

$$
\begin{array}{llll}
\rightarrow & \theta = \emptyset & \Sigma = [X = Y, X = Z, Y = a] & Failed = false \\
\rightarrow & \theta = \{X/Y\} & \Sigma = [Y = Z, Y = a] & Failed = false \\
\rightarrow & \theta = \{X/Z, Y/Z\} & \Sigma = [Z = a] & Failed = false \\
\rightarrow & \theta = \{X/a, Y/a, Z/a\} & \Sigma = [] & Failed = false
\end{array}
$$

Recursive Side-effect free algorithm for computing mgu

Input: Two terms T_1 and T_2 to be unified and the mgu θ.

Output: θ, the mgu if T_1 and T_2 or raises *FAIL* exception.

Algorithm: $mgu(T_1, T_2, \theta)$.

```
mgu(X,Y,θ) =
  X = Xθ
  Y = Yθ
  case
    X is a variable that does not occur in Y:
      return (θ{X/Y} ∪ {X/Y})
    Y is a variable that does not occur in X:
      return (θ{Y/X} ∪ {Y/X})
    X and Y are indentical constants or variables:
      return θ
    X is f(X1,...,Xn) and Y is f(Y1,...,Yn):
      return (fold (fun θ (X,Y) -> mgu(X,Y,θ)) θ [(X1,Y1),...,(Xn,Yn)])
    otherwise:
      raise FAIL
}
```

Abstract interpreter

Input: A goal G and a program P

Output: An instance of G that is a logical consequence of P, or **false** otherwise.

Algorithm: run(P,G)

```
Initialise resolvent to G.
while (the resolvent is not empty) {
  choose a goal A from the resolvent //goal order
  choose a (renamed) clause A' <- B1,...,Bn from P //rule order
      such that A and A' unify with mgu θ
    (if no such goal and clause exist, exit the while loop).
  replace A by B1,...,Bn in the resolvent
  apply θ to the resolvent and G
}
If the resolvent is empty, then output G, else output false.
```

Abstract interpreter is non-deterministic

Abstract interpreter is non-deterministic

Non-determinism is essential for correctness. Consider the program:

```
plus(1,3,4).
plus(2,2,4).
even(2).
```

and the goal `plus(X,Y,4), even(X)` .

- Both `plus(2,2,4)` and `plus(1,3,4)` unify with `plus(X,Y,4)` .
- But only `plus(2,2,4)` ensures that the second goal `even(X)` is satisfied.
- Since the abstract interpreter in non-deterministic, one of its behaviours is to choose `plus(2,2,4)` , which will lead to success without failure.

Abstract interpreter is non-deterministic

Non-determinism is essential for correctness.

Consider the program:

```
plus(1,3,4).
plus(2,2,4).
even(2).
odd(1).
```

and the goal `plus(X,Y,4), even(X)` .

OTOH, if the second goal `even(X)` is chosen as the first to resolve, then it will only unify with `even(2)` , which will change the other goal to `plus(2,Y,4)` which leaves only one choice.

Backtracking and choice points

- The abstract interpreter does not explicitly encode backtracking (recover from bad choices) and choice points (present more than one result).
- The program is said to be **deterministic**, if there is exactly one clause from the program to reduce each goal.
  - No backtracking and choice points are necessary.

Backtracking and choice points - Assignment 6

- Backtracking and choicepoints are encoded more naturally in a recursive formulation of the abstract interpreter.
  - Left as an exercise to you.
  - Take hints from how we transformed mgu algorithm to a recursive one; `List.fold_left` is your friend.
- You will implement an interpreter for prolog in OCaml with backtracking and choice points for Assignment 6.
  - 1/3 of the score in assignment 6 will test backtracking and choice point implementation of your interpreter.

Fin.