Constructive Logic (15-317), Spring 2021
Recitation 12: Moded and Typed Programming (4-21-2021)

1 Modes

We often talk about modes in Prolog. Modes are a way of describing how a predicate will be used. We denote an argument of a predicate with + if that argument is provided to the predicate, and with - if that argument is outputted by the predicate. A defined Prolog predicate can often work with many different modes.

Note that not all possible modes work correctly for a given predicate in Prolog. Many predicates do not work with all arguments moded negatively. For example, given a predicate `permutation(L, P)`, where `L` is a list and `P` is a permutation of `L`, the mode `permutation(-L, -P)` will not terminate because there are infinite pairs `(L, P)` such that `P` is a permutation of `L`.

Task 1. Let the predicate `zip` be defined as follows:

```
zip([], [], []).
zip([X|L], [Y|M], [(X, Y)|P]) :- zip(L, M, P).
```

Which modes does `zip` work correctly with?

Solution 1: There are 5 modes that work correctly; the other 3 do not terminate.

- (+, +, +)
- (+, +, -)
- (-, +, +)
- (+, -, +)
- (-, -, +)

Task 2. Let the predicate `mult` be defined as follows:

```
nat(z).
nat(s(N)) :- nat(N).

plus(z, N, N).
plus(s(M), N, s(P)) :- plus(M, N, P).

mult(z, N, z).
mult(N, z, z).
mult(s(M), N, P) :- plus(Q, N, P), mult(M, N, Q).
```

Which modes does `mult` work correctly with?

Solution 2: Note that (+, +, -) does not work, even though it might be expected to, due to the infinite possibilities in the `plus(-, +, -)` mode. The mode (+, -, +) does not work either as `M` is not known after running `mult(z, M, z)`.

So only (+, +, +) work correctly.

Task 3. How might we rewrite `mult` if we wanted it to work correctly with the `mult(+, +, -)` modality?

Solution 3: Change the order of the `plus` and `mult` in the third clause of the predicate.

Constructive Logic (15-317), Spring 2021
Recitation 13: Datalog with chains (4-28-2021)

1 Forward Logic with Inference Rules

Task 1. Give the inference rules for a forward logic program length(l, n) which derives the atom **no** if and only if n is not the length of list l. You may assume that n and l are ground.

Solution 1:

$$\frac{length([X \mid L],\ 0)}{no}\ zero \qquad \frac{length([X \mid L],\ N)}{length(L,\ N-1)}\ dec$$

Task 2. Recall the grammar representing natural numbers:

$$n ::= z \mid s(n)$$

Give the inference rules for a forward logic program factor(m, n) which derives the atom **no** if and only if m does not evenly divide n. You may assume that m and n are ground.

Solution 2:

$$\frac{factor(m,n)}{div(m,n,m)}\ def \qquad \frac{div(s(m),s(n),d)}{div(m,n,d)}\ decr \qquad \frac{div(z,s(n),d)}{div(d,s(n),d)}\ rep \qquad \frac{div(s(m),z,d)}{no}\ nz$$

2 Functional Evaluation with Forward Chaining

Consider the language of the untyped lambda calculus.

$$e ::= x \mid \lambda x.e \mid e_1 e_2$$

We can write a set of rules using three predicates

$\mathsf{eval}(e)$	evaluate e
$e \mapsto^* e'$	e reduces to e'
$e \hookrightarrow v$	e evaluates to v

so that we can evaluate e with forward chaining, by seeding the system with $\mathsf{eval}(e)$ and waiting for a fact of the form $e \hookrightarrow v$ to appear.

Task 3. Define such a set of rules.

Solution 3: See `http://www.cs.cmu.edu/~fp/courses/lp/lectures/20-bottomup.pdf`.

3 Return to Focusing

Recall the rules for even and odd natural numbers:

$$\frac{}{\mathit{even}(z)}\ \mathit{ev}_z \qquad \frac{\mathit{odd}(N)}{\mathit{even}(s(N))}\ \mathit{ev}_s \qquad \frac{\mathit{even}(N)}{\mathit{odd}(s(N))}\ \mathit{od}_s$$

Task 4. Give a proof, using the focusing rules, of odd($s(s(s(z)))$).

Solution 4: Define Γ_{eo} = even(z), $\forall n.$ odd(n) $\supset$ even(s(n)), $\forall n.$ even(n) $\supset$ odd(s(n)).
Our goal sequent is $\Gamma_{eo} \longrightarrow$ even(s(s(z)))).

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{id}
\UnaryInfC{$\Gamma_{eo}, [\mathit{even}(s(s(z)))] \longrightarrow \mathit{even}(s(s(z)))$}
\AxiomC{}
\RightLabel{id}
\UnaryInfC{$\Gamma_{eo}, [\mathit{odd}(s(z))] \longrightarrow \mathit{odd}(s(z))$}
\AxiomC{}
\RightLabel{id}
\UnaryInfC{$\Gamma_{eo}, [\mathit{even}(z)] \longrightarrow \mathit{even}(z)$}
\RightLabel{focusL}
\UnaryInfC{$\Gamma_{eo} \longrightarrow \mathit{even}(z)$}
\RightLabel{blurR}
\UnaryInfC{$\Gamma_{eo} \longrightarrow [\mathit{even}(z)]$}
\RightLabel{$\supset L$}
\BinaryInfC{$\Gamma_{eo}, [\mathit{even}(z) \supset \mathit{odd}(s(z))] \longrightarrow \mathit{odd}(s(z))$}
\RightLabel{$\forall L$}
\UnaryInfC{$\Gamma_{eo}, [\forall n.\mathit{even}(n) \supset \mathit{odd}(s(n))] \longrightarrow \mathit{odd}(s(z))$}
\RightLabel{focusL}
\UnaryInfC{$\Gamma_{eo} \longrightarrow \mathit{odd}(s(z))$}
\RightLabel{blurR}
\UnaryInfC{$\Gamma_{eo} \longrightarrow [\mathit{odd}(s(z))]$}
\RightLabel{$\supset L$}
\BinaryInfC{$\Gamma_{eo}, [\mathit{odd}(s(z)) \supset \mathit{even}(s(s(z)))] \longrightarrow \mathit{even}(s(s(z)))$}
\RightLabel{$\forall L$}
\UnaryInfC{$\Gamma_{eo}, [\forall n.\mathit{odd}(n) \supset \mathit{even}(s(n))] \longrightarrow \mathit{even}(s(s(z)))$}
\RightLabel{focusL}
\UnaryInfC{$\Gamma_{eo} \longrightarrow \mathit{even}(s(s(z)))$}
\end{prooftree}
$$