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:

\[
\text{zip([], [], []).}
\]
\[
\text{zip([X|L], [Y|M], [(X, Y)|P]) : - zip(L, M, P).}
\]

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:

\[
\text{nat(z).}
\]
\[
\text{nat(s(N)) : - nat(N).}
\]
\[
\text{plus(z, N, N).}
\]
\[
\text{plus(s(M), N, s(P)) : - plus(M, N, P).}
\]
\[
\text{mult(z, N, z).}
\]
\[
\text{mult(N, z, z).}
\]
\[
\text{mult(s(M), N, P) : - plus(Q, N, P), mult(M, N, Q).}
\]

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.
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:

\[
\frac{\text{length}([X | L], 0)}{\text{no}} \quad \frac{\text{length}([X | L], N)}{\text{dec}}
\]

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:

\[
\frac{\text{factor}(m, n)}{\text{div}(m, n, m)} \quad \frac{\text{div}(s(m), s(n), d)}{\text{dec}} \quad \frac{\text{div}(z, s(n), d)}{\text{rep}} \quad \frac{\text{div}(s(m), z, d)}{\text{nz}}
\]

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

\[
\text{eval}(e) \quad \text{evaluate } e
\]

\[e \mapsto e' \quad e \text{ reduces to } e'
\]

\[e \mapsto v \quad e \text{ evaluates to } v
\]

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

Task 3. Define such a set of rules.

3 Return to Focusing

Recall the rules for even and odd natural numbers:

\[
\begin{align*}
\text{even} & \quad \quad \text{ev}_z \\
\text{odd} & \quad \quad \text{od}_s \\
\text{even} & \quad \quad \text{ev}_s \\
\text{odd} & \quad \quad \text{od}_s
\end{align*}
\]

Task 4. Give a proof, using the focusing rules, of \(\text{odd}(s(s(z)))\).

Solution 4: Define \(\Gamma_{eo} = \text{even}(z), \forall n. \text{odd}(n) \supset \text{even}(s(n)), \forall n. \text{even}(n) \supset \text{odd}(s(n))\).

Our goal sequent is \(\Gamma_{eo} \rightarrow \text{even}(s(s(z)))\).

\[
\begin{align*}
\Gamma_{eo}, [\text{even}(z)] & \rightarrow \text{id} \\
\Gamma_{eo} & \rightarrow \text{i}d
\end{align*}
\]