Prolog

Ross (Tate)

Filling in the Blanks

• Rather than reverse((a,b)) returns (b,a)
• What X makes reverse((a,b), X) true?
  – The blank X gets filled with (b,a)
• X = 5.
  – Fills X with 5.
• X = 5, X = 6.
  – Fails since no value can equal both 5 and 6.
• “=” unifies the left and right to the same value
Database of Rules (and Facts)

- Prolog is a database of rules (and facts)
- name(params) :- further requirements.
- A “fact” is simply a rule with no further requirements
- name(params). (name starts with lower case letter)

isparent(ross, colleen).
child(Parent, Child) :- isparent(Child, Parent).

- Lower case words denote “atoms”
- Upper case words are variables or “blanks”

Backtracking

- Rules in Prolog can be satisfied many ways
- So, which way should be chosen?
- Prolog starts at the top and finds the first way for the rule to be satisfied
- It also remembers where it was satisfied
- Say the results of that fail later
- Then Prolog simply starts where it left and finds more ways until none are left (fails)
Structures

- Structures are a way to combine things
- They take the form: name(parts)
- pair(x, y) is one such structure
- The parts can be anything: atoms, variables, or even structures
- Structures are stored as trees in Prolog, with the structure name being the name of the node and the parts being the children

Lists and Recursion

- The empty list: [] (a special atom)
- [head | tail] is how lists are stored (linked list)
  - [a, b, c] is shorthand for [a | [b | [c | [] ] ] ]
  - [a, b | tail] is shorthand for [a | [b | tail] ]

- element([X | _], X). (_ is unnamed variable)
- element([_ | Tail], X). :- element(Tail, X).
- The second rule is the recursive call
Redundancy

- `element([a,b,a], a), X = 2.`
  - Returns twice! `X = 2 AND X = 2`.
- ‘a’ is an element of `[a,b,a]` twice
- `element` finds every way ‘a’ can be an element of `[a,b,a]`, so with backtracking it finds 2 ways
- What if instead of `X = 2` there was a really computation intensive goal?
- That goal would be calculated for the same input twice, finding the same answer twice, which is a waste of time

Cuts (!)

- Cuts prevent backtracking
- Cannot be backtracked through
- Makes the parent goal’s result final (no searching for more answers for the same question)
- `contains([X | _ ], X) :- !.`
- `contains([ _ | Tail], X) :- contains(Tail, X).`
- `contains` will only find one answer, unlike `element`, because of the cut (!)
fail, true, and repeat

- fail is a special goal which always fails
  - often used after a cut:
    - not(Goal) :- Goal, !, fail; true.
- true is a special goal which always passes
- repeat is a special goal which always passes and continues to pass when backtracked into:
  - repeat.
  - repeat :- repeat.

Arithmetic

- $X = 1 + 1$. produces $X = 1 + 1$, not $X = 2$.
- $X$ is $1 + 1$ does produce $X = 2$.
  - “is” evaluates the right then unifies with the left
- $<, >, \ldots$ evaluate both sides and then compare
- In order to be evaluated there cannot be any unknowns, otherwise an error will be thrown.
- length([], 0).
bagof and setof

- Rather than having to hit ‘;’ to get all the answers, why not have a function that does it for you?
- bagof(Variables, Goal, Results) will repeatedly attempt Goal and add the unification of Variables to Results (fails if Goal not possible)
- setof is like bagof but sorts Results and removes duplicates