Programming Languages

2nd edition
Tucker and Noonan

Chapter 15
Logic Programming

Q: How many legs does a dog have if you call its tail a leg?
A: Four. Calling a tail a leg doesn't make it one.
Abraham Lincoln

Contents

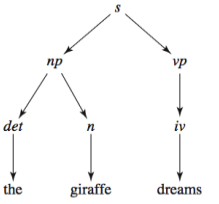
15.3.3 Natural Language Processing

- BNF can define natural language (e.g., English) syntax.
  - *This was the original purpose of BNF when it was invented by Chomsky in 1957.*
- A Prolog program can model a BNF grammar.
  - *This was an original purpose of Prolog when it was designed in the 1970s.*
- A Prolog list can model a sentence.
  - *E.g.,* [the, giraffe, dreams]
- So running the program can parse a sentence.

NLP Example

Consider the following BNF grammar and parse tree:

$s \rightarrow np\ vp$
$np \rightarrow det\ n$
$vp \rightarrow tv\ np$
$\rightarrow iv$
$det \rightarrow$ the
$n \rightarrow$ giraffe
$\rightarrow$ apple
$iv \rightarrow$ dreams
$tv \rightarrow$ eats



Here, *s*, *np*, *vp*, *det*, *n* , *iv*, and *tv* denote "sentence," "noun phrase," "verb phrase," "determiner," "noun," "intransitive verb," and "transitive verb."

Prolog Encoding (naïve version)

```
s(X, Y) :- np(X, U), vp(U, Y).
np(X, Y) :- det(X, U), n(U, Y).
vp(X, Y) :- iv(X, Y).
vp(X, Y) :- tv(X, U), np(U, Y).
det([the | Y], Y).
n([giraffe | Y], Y).
n([apple | Y], Y).
iv([dreams | Y], Y).
tv([eats | Y], Y).
```

The first rule reads, "list X is a sentence leaving tail Y if X is a noun phrase leaving tail U and U is a verb phrase leaving tail Y."

Example Trace

The query asks, "can you resolve [the, giraffe, dreams] as an s, leaving tail []?"

```
?- s([the, giraffe, dreams],[]).
Call: ( 7) s([the, giraffe dreams], []) ?
Call: ( 8) np([the, giraffe, dreams], _L131) ?
Call: ( 9) det([the, giraffe, dreams], _L143) ?
Exit: ( 9) det([the, giraffe, dreams], [giraffe, dreams]) ?
Call: ( 9) n([giraffe, dreams], _L131) ?
Exit: ( 9) n([giraffe, dreams], [dreams]) ?
Exit: ( 8) np([the, giraffe, dreams], [dreams]) ?
Call: ( 8) vp([dreams], []) ?
Call: ( 9) iv([dreams], []) ?
Exit: ( 9) iv([dreams], []) ?
Exit: ( 8) vp([dreams], []) ?
Exit: ( 7) s([the, giraffe, dreams], []) ?
Yes
```

The result is success.

Definite Clause Grammars (DCGs)

```
s --> np, vp.
np --> det, n.
vp --> iv.
vp --> tv, np.
det --> [the].
n --> [giraffe].
n --> [apple].
iv --> [dreams].
tv --> [eats].
```

This replaces
`s(X, Y) :- np(X, U), vp(U, V).`
but it means the same thing.

Note: This form looks more like a series of BNF rules, and it's simple to write!

Generating Parse Trees

Terms and variables can be added to the DCG rules so that a parse tree can be generated from a query.

e.g., if we change

```
s --> np, vp.
```

to

```
s(s(NP, VP)) --> np(NP), vp(VP).
```

the variables NP and VP can capture intermediate subtrees.

When all rules are augmented in this way, the query

```
?- s(Tree, [the, giraffe, dreams], []).
```

delivers the parse tree as a parenthesized list:

```
Tree = s(np(det(the), n(giraffe)), vp(iv(dreams)))
```


15.3.4 Semantics of Clite

Program state can be modeled as a list of pairs. E.g.,

```
[[x,1], [y,5]]
```

Function to retrieve the value of a variable from the state:

```
get(Var, [[Var, Val] | _], Val).
get(Var, [_ | Rest], Val) :- get(Var, Rest, Val).
```

e.g.,

```
?- get(y, [[x, 5], [y, 3], [z, 1]], V).
V = 3
```


State transformation

Function to store a new value for a variable in the state:

```
onion(Var, Val, [[Var, _] | Rest], [[Var, Val] | Rest]).
onion(Var, Val, [Xvar | Rest], [Xvar | OState]) :-
onion(Var, Val, Rest, OState).
```

Note: second and third arguments denote the input and output states, resp.

E.g.,

```
?- onion(y, 4, [[x, 5], [y, 3], [z, 1]], S).
S = [[x, 5], [y, 4], [z, 1]]
```


Modeling Clite Abstract Syntax

Skip	skip
Assignment	assignment(target, source)
Block	block([s1, …, sn])
Loop	loop(test, body)
Conditional	conditional(test, thenbranch, elsebranch)
Expression	
Value	value(val)
Variable	variable(id)
Binary	operator(term1, term2)
	where operator is one of plus, times, minus, div, lt, le, eq, ne, gt, ge

Semantics of Statements

General form: minstruction(statement, inState, outState)

```
minstruction(skip, State, State).
minstruction(assignment(Var, Expr), InState, OutState) :-
    mexpression(Expr, InState, Val),
    onion(Var, Val, InState, OutState).
```

Notes: skip leaves the state unchanged.

For an assignment, the expression is first evaluated and the variable Val is instantiated.

Next, the target variable (Var) is assigned this value using the onion function.

Conditional, Loop, and Block

```
minstruction(conditional(Test, Thenbranch, Elsebranch),      InState,
    OutState) :-
/* First, evaluate test.  If it succeeds, Outstate is the meaning of
    Thenbranch in inState.  Otherwise, it is the meaning of Elsebranch. */

minstruction(loop(Test, Body), InState, OutState) :-
/* First, evaluate test.  If it succeeds, OutState is the meaning of loop with
    InState the meaning of Body in InState.  Otherwise, OutState is the
    InState. */

minstruction(block([Head | Tail]), InState, OutState) :-
/* If Tail = [], then OutState is the meaning of Head.  Otherwise, it is the
    meaning of block([Tail]) with InState the meaning of Head. */
```


Expression

```
mexpression(value(Val), _, Val).
mexpression(variable(Var), State, Val) :-
    get(Var, State, Val).
mexpression(plus(Expr1, Expr2), State, Val) :-
    mexpression(Expr1, State, Val1),
    mexpression(Expr2, State, Val2),
    Val is Val1 + Val2.
…
```

Note the creation of temporary variables Val1 and Val2, and the final assignment of their sum.

To Do:

1. Show that these definitions give 5 as the meaning of y+2 in the state ((x 5) (y 3) (z 1)). I.e., show that

   ```
   mexpression(plus(variable(y), value(2)), [[x,5], [y,3], [z,1]], Val)
   …
   Val = 5
   ```

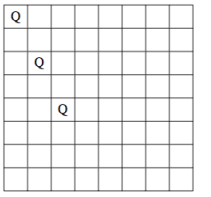
2. Complete the definition of minstruction for conditionals, loops, and blocks.

15.3.5 Eight Queens Problem

A backtracking algorithm for which each trial move's:

1. Row must not be occupied,
2. Row and column's SW diagonal must not be occupied, and
3. Row and column's SE diagonal must not be occupied.

If a trial move fails any of these tests, the program backtracks and tries another. The process continues until each row has a queen (or until all moves have been tried).



Modeling the Solution

- Board is NxN. Goal is to find all solutions.
- For some values of N (e.g., N=2) there are no solutions.
- Rows and columns use zero-based indexing.
- Positions of the queens in a list Answer whose *i*th entry gives the row position of the queen in column *i*, in reverse order.

  E.g., Answer = [4, 2, 0] represents queens in (row, column) positions (0,0), (2,1), and (4,2); see earlier slide.
- End of the program occurs when Answer has N entries or 0 entries (if there is no solution).
- Game played using the query:

  ```
  ?- queens(N, Answer).
  ```


Generating a Solution

```
solve(N, Col, RowList, _, _, RowList) :-
    Col >= N.

solve(N, Col, RowList, SwDiagList, SeDiagList, Answer) :-
    Col < N,
    place(N, 0, Col, RowList, SwDiagList, SeDiagList, Row),
    getDiag(Row, Col, SwDiag, SeDiag),
    NextCol is Col + 1,
    solve(N, NextCol, [Row | RowList], [SwDiag | SwDiagList],
        [SeDiag | SeDiagList], Answer).
```

Generating SW and SE Diagonals

```
getDiag(Row, Col, SwDiag, SeDiag) :-
    SwDiag is Row + Col, SeDiag is Row - Col.
```


Generating a Safe Move

```
place(N, Row, Col, RowList, SwDiagList, SeDiagList, Row) :-
    Row < N,
    getDiag(Row, Col, SeDiag, SwDiag),
    valid(Row, SeDiag, SwDiag, RowList, SwDiagList, SeDiagList).

place(N, Row, Col, RowList, SwDiagList, SeDiagList, Answer) :-
    NextRow is Row + 1,
    NextRow < N,
    place(N, NextRow, Col, RowList, SwDiagList, SeDiagList, Answer).
```

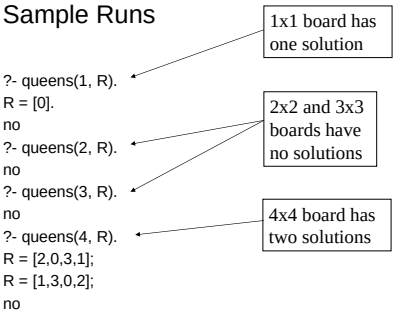

Checking for a Valid Move

```
valid(_, _, _, [ ]).

valid(TrialRow, TrialSwDiag, TrialSeDiag, RowList,
          SwDiagList, SeDiagList) :-
    not(member(TrialRow, RowList)),
    not(member(TrialSwDiag, SwDiagList)),
    not(member(TrialSeDiag, SeDiagList)).
```

Note: RowList is a list of rows already occupied.

Sample Runs




```
?- queens(1, R).
R = [0].
no
?- queens(2, R).
no
?- queens(3, R).
no
?- queens(4, R).
R = [2,0,3,1];
R = [1,3,0,2];
no
```