Linear Logic

Lecture 18: Linear Logic Programming
March 17, 1998

1. Solitaire
2. Simple Database

Solitaire, Program

```
jump (s N) :-
          b X Y peg , b (s X) Y peg , b (s (s X)) Y empty ,
          ((b X Y empty , b (s X) Y empty , b (s (s X)) Y peg)
           -o jump N).
jump (s N) :-
          b X Y peg , b (s X) (s Y) peg , b (s (s X)) (s (s Y)) empty ,
          ((b X Y empty , b (s X) (s Y) empty , b (s (s X)) (s (s Y)) peg)
           -o jump N).
jump (s N) :- b X Y peg , b X (s Y) peg , b X (s (s Y)) empty ,
          ((b X Y empty , b X (s Y) empty , b X (s (s Y)) peg)
           -o jump N).
jump (s N) :-
          b (s (s X)) Y peg , b (s X) Y peg , b X Y empty ,
          ((b (s (s X)) Y empty , b (s X) Y empty , b X Y peg)
           -o jump N).
jump (s N) :-
          b (s (s X)) (s (s Y)) peg , b (s X) (s Y) peg , b X Y empty ,
          ((b (s (s X)) (s (s Y)) empty , b (s X) (s Y) empty , b X Y peg)
           -o jump N).
jump (s N) :-
          b X (s (s Y)) peg , b X (s Y) peg , b X Y empty ,
          ((b X (s (s Y)) empty , b X (s Y) empty , b X Y peg)
           -o jump N).
jump z :- erase.
```

Solitaire, Initial State

```
LINEAR b z z peg.
LINEAR b (s z) z peg.
LINEAR b (s z) (s z) peg.
LINEAR b (s (s z)) z peg.
LINEAR b (s (s z)) (s z) empty.
LINEAR b (s (s z)) (s (s z)) peg.
LINEAR b (s (s (s z))) z peg.
LINEAR b (s (s (s z))) (s z) peg.
LINEAR b (s (s (s z))) (s (s z)) peg.
LINEAR b (s (s (s z))) (s (s (s z))) peg.
LINEAR b (s (s (s (s z)))) z peg.
LINEAR b (s (s (s (s z)))) (s z) peg.
LINEAR b (s (s (s (s z)))) (s (s z)) peg.
LINEAR b (s (s (s (s z)))) (s (s (s z))) peg.
LINEAR b (s (s (s (s z)))) (s (s (s (s z)))) peg.
```

Simple Database

```
MODULE database.

db :- write_sans "Command: ", read Command, do Command.
db :- write_sans  "Try again.", nl, db.

do (enter E)      :- entry E -o db.
do (commit E)     :- entry E => db.
do (retract E)    :- entry E,
                     ({entry E} ->
                       (write E, write_sans  " is a necessary fact, and ",
                        write_sans  "cannot be retracted.", nl,  db)
                     | (write E, write_sans  " was retracted.", nl,  db)).

do (update E N)   :- entry E,
                     ({entry E} ->
                       (write E, write_sans  " is a necessary fact, and ",
                        write_sans  "cannot be updated.", nl,        db)
                     | (write E, write_sans  " was updated to ",
                        write N, write_sans  ".", nl, entry N -o db)).
do (check Q)      :- entry Q, erase, write Q,
                     write_sans  " is an entry.", nl & db.
do (necessary Q) :- {entry Q}, erase, write Q,
                     write_sans  " is a necessary entry", nl & db.
do quit           :- abort.
```

Database, Interaction

```
?- database --o db.
Command: enter (student josh).
Command: enter (student ramesh).
Command: commit (student josh).
Command: enter (professor jean).
Command: enter (professor dale).
Command: check (professor jean).
professor jean is an entry.
Command: necessary (professor jean).
Try again.
Command: update (student ramesh) (professor ramesh).
student ramesh was updated to professor ramesh.
Command: update (student josh) (professor josh).
student josh is a necessary fact, and cannot be updated.
Command: quit.
aborted...
```