Function symbols

Function symbols in Prolog are used to build new terms from old (much like a constructor in Haskell).

For example, a calendar date might be coded in Prolog by writing

```
date(14,2,2000)
```

for the 14th February 2000, say.

If we wish to compare dates then we might write a predicate

```
before(date(_,_,Y1),date(_,_,Y2)):- Y1 < Y2.
before(date(_,M1,Y),date(_,M2,Y)):- M1 < M2.
before(date(D1,M,Y),date(D2,M,Y)):- D1 < D2.
```

List notation

Lists are represented in Prolog as terms of a particular form:

- $[\,]$ is a list,
- if x is any term and xs is a list then $[x \mid xs]$ is the list with x at the head and xs as the tail.

We have the following scheme of abbreviation.

- $[a_0, a_1, \ldots, a_n]$ stands for $[a_0, a_1, \ldots, a_n \mid []]$
- $[a_0, a_1, \ldots, a_n \mid t]$ stands for $[a_0 \mid [a_1, \ldots, a_n \mid t]]$

Member

```
member(X,[X|_]).
member(X,[_|Xs]) :- member(X,Xs).
```

The predicate can be used to check whether an element is a member of a list

```
?- member(yellow,[red,blue,yellow,green]).
Yes
```

or to generate the members of the list

```
?- member(X,[apple,pear,orange]).
X = apple ;
X = pear ;
X = orange
```

Append

```
append([],Ys,Ys).
append([X|Xs],Ys,[X|Zs]) :- append(Xs,Ys,Zs).
```

The predicate can be used to join two lists together

```
?- append([b,a],[c,a,d],Zs).
Zs = [b, a, c, a, d]
```

or to split a list into parts

```
?- append(Xs,Ys,[a,b]).
Xs = []
Ys = [a, b] ;
Xs = [a]
Ys = [b] ;
Xs = [a, b]
Ys = []
```

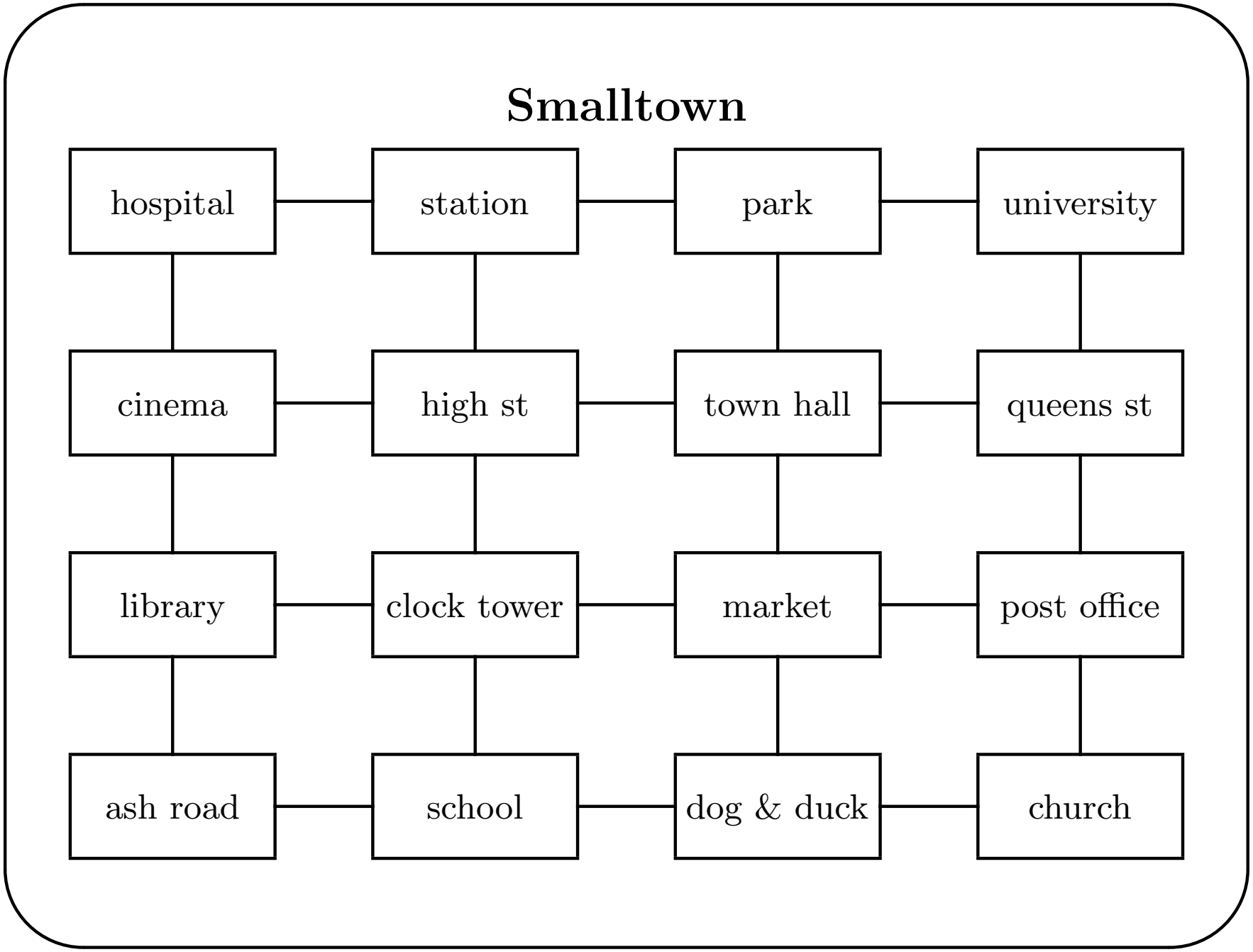
Select

The formula $select(x, l, k)$ holds if k is the result of deleting one occurrence of x from the list l.

```
select(X,[X|Xs],Xs).
select(Y,[X|Xs],[X|Zs]) :- select(Y,Xs,Zs).
```

An alternative definition

```
select(Y,Xs,Zs) :- append(As,[Y|Bs],Xs),
                   append(As,Bs,Zs).
```


Smalltown
hospital
station
park
university
cinema
high st
town hall
queens st
library
clock tower
market
post office
ash road
school
dog & duck
church

The map is stored as a series of facts:

```
go(east, hospital, station).
go(east, cinema, high_st).
...
go(north, ash_road, library).
go(north, school, clock_tower).
```

The corresponding information is given for south and west via

```
go(west, X,Y) :- go(east, Y,X).
go(south,X,Y) :- go(north,Y,X).
```

It might be useful to know whether one place is near to another.

```
near(X,Y) :- go(_,X,Y).
near(X,Y) :- perp(D1,D2), go(D1,X,Z), go(D2,Z,Y).
```

The next two predicates determine whether it is possible to walk in a straight line from one place to another.

```
walk(_,X,X).
walk(Dir,X,Y) :- go(Dir,X,Z), walk(Dir,Z,Y).

walk1(Dir,X,Y) :- go(Dir,X,Y).
walk1(Dir,X,Y) :- go(Dir,X,Z), walk1(Dir,Z,Y).
```

The layout of the town is such that every location can be reached in (at most) two straight line walks.

```
walk2(D1,D2,X,Y) :- perp(D1,D2), walk(D1,X,Z),
                    walk(D2,Z,Y).
```

```
?- near(hospital,X), near(clock_tower,X).
X = cinema ;
X = high_st

?- walk(east,ash_road,Y).
Y = ash_road ;
Y = school ;
Y = dog_and_duck ;
Y = church

?- walk(D, market, park).
D = north
```

```
?- walk2(north,east,town_hall,Y).
Y = town_hall ;
Y = queens_st ;
Y = park ;
Y = university

?- walk2(D1,D2,cinema,market).
D1 = south
D2 = east

?- walk2(north,east,clock_tower,X), near(X,church).
X = market ;
X = post_office
```

To follow a list of directions which are given:

```
?- route(ash_road,Y,[north,north,east,south]).
Y = clock_tower


route(X,X,[]).
route(X,Y,[D|Ds]) :- go(D,X,Z), route(Z,Y,Ds).
```

To find the directions from one place to another:

```
?- steps2(park, clock_tower, Ds).
Ds = [south, south, west]


steps(_,X,X,[]).
steps(D,X,Y,[D|Ds]) :- go(D,X,Z), steps(D,Z,Y,Ds).
steps2(X,Y,Ds) :- perp(D1,D2), steps(D1,X,Z,Bs),
                  steps(D2,Z,Y,Cs), append(Bs,Cs,Ds).
```