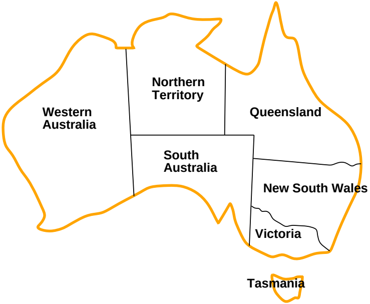
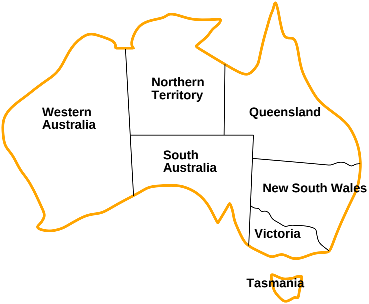
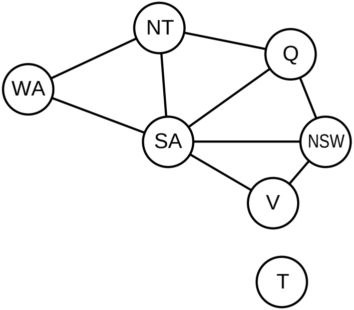
Graph modeling



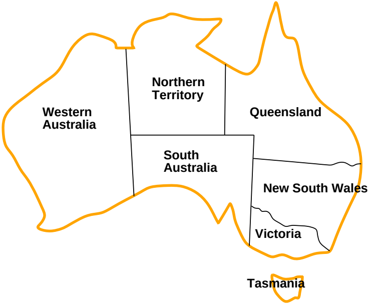
Graph modeling

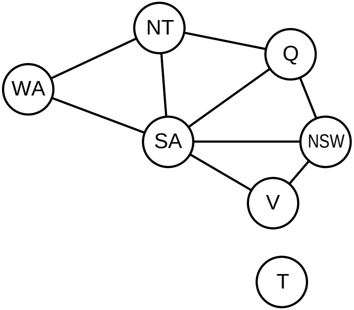




Russell & Norvig

Graph modeling





Russell & Norvig

```
arc(wa,nt).    arc(nt,q).    arc(q,nsw).
arc(wa,sa).    arc(nt,sa).   arc(sa,q).
arc(sa,nsw).   arc(sa,v).    arc(v,nsw).

arc2(X,Y) :- arc(X,Y) ; arc(Y,X).
```

Non-termination (due to poor choices)

```
i :- p,q.                    [i]

i :- r.

p :- i.

r.
```

```
| ?- i.
```

```
prove([],_).
prove([H|T],KB) :- member([H|B],KB), append(B,T,Next),
                   prove(Next,KB).
| ?- prove([i],[[i,p,q],[i,r],[p,i],[r]]).
```

Non-termination (due to poor choices)

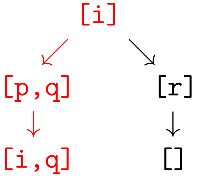
```
i :- p,q.

i :- r.

p :- i.

r.
_________
| ?- i.
```

```
      [i]
     ↙   ↘
[p,q]     [r]
```

```
prove([],_).

prove([H|T],KB) :- member([H|B],KB), append(B,T,Next),
                   prove(Next,KB).

| ?- prove([i],[[i,p,q],[i,r],[p,i],[r]]).
```

Non-termination (due to poor choices)

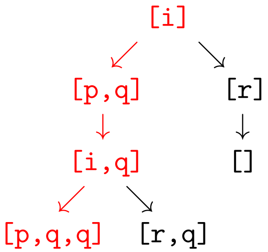
```
i :- p,q.

i :- r.

p :- i.

r.
_________
| ?- i.
```




```
prove([],_).

prove([H|T],KB) :- member([H|B],KB), append(B,T,Next),
                   prove(Next,KB).

| ?- prove([i],[[i,p,q],[i,r],[p,i],[r]]).
```

Non-termination (due to poor choices)

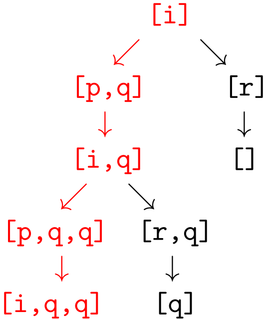
```
i :- p,q.

i :- r.

p :- i.

r.
_________

| ?- i.
```




```
prove([],_).

prove([H|T],KB) :- member([H|B],KB), append(B,T,Next),
                   prove(Next,KB).

| ?- prove([i],[[i,p,q],[i,r],[p,i],[r]]).
```

Non-termination (due to poor choices)

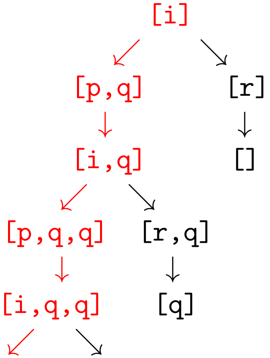
```
i :- p,q.

i :- r.

p :- i.

r.
________

| ?- i.
```




```
prove([],_).

prove([H|T],KB) :- member([H|B],KB), append(B,T,Next),
                   prove(Next,KB).

| ?- prove([i],[[i,p,q],[i,r],[p,i],[r]]).
```

Non-termination (due to poor choices)

```
i :- p,q.

i :- r.

p :- i.

r.
________
| ?- i.
```




```
prove([],_).

prove([H|T],KB) :- member([H|B],KB), append(B,T,Next),
                   prove(Next,KB).

| ?- prove([i],[[i,p,q],[i,r],[p,i],[r]]).
```

Determinization (eliminate choice)

A fsm `[Trans, Final, Q0]` such that

for all `[Q,X,Qn]` *and* `[Q,X,Qn']` *in* `Trans`, $\texttt{Qn} = \texttt{Qn}'$

is a *deterministic finite automaton* (DFA).

Determinization (eliminate choice)

A fsm [Trans, Final, Q0] such that

for all [Q,X,Qn] *and* [Q,X,Qn′] *in* Trans, $\quad \texttt{Qn} = \texttt{Qn}'$

is a *deterministic finite automaton* (DFA).

Fact. Every fsm has a DFA accepting the same language.

Determinization (eliminate choice)

A fsm `[Trans, Final, Q0]` such that

for all `[Q,X,Qn]` *and* `[Q,X,Qn′]` *in* `Trans`, $\texttt{Qn} = \texttt{Qn}'$

is a *deterministic finite automaton* (DFA).

Fact. Every fsm has a DFA accepting the same language.

Proof: Subset (powerset) construction

Apply to `arc`,`goal`, contra `Trans`,`Final`:

```
arcD(NodeList,NextList) :-
      setof(Next, arcLN(NodeList,Next), NextList).
arcLN(NodeList,Next) :- member(Node,NodeList),
                        arc(Node,Next).
```

Determinization (eliminate choice)

A fsm `[Trans, Final, Q0]` such that

for all `[Q,X,Qn]` *and* `[Q,X,Qn']` *in* `Trans`, $\texttt{Qn} = \texttt{Qn}'$

is a *deterministic finite automaton* (DFA).

Fact. Every fsm has a DFA accepting the same language.

Proof: Subset (powerset) construction

Apply to `arc`,`goal`, contra `Trans`,`Final`:

```
arcD(NodeList,NextList) :-
      setof(Next, arcLN(NodeList,Next), NextList).

arcLN(NodeList,Next) :- member(Node,NodeList),
                        arc(Node,Next).

goalD(NodeList):-member(Node,NodeList),goal(Node).
```

Determinization (eliminate choice)

A fsm `[Trans, Final, Q0]` such that

for all `[Q,X,Qn]` *and* `[Q,X,Qn']` *in* `Trans`, $\texttt{Qn} = \texttt{Qn}'$

is a *deterministic finite automaton* (DFA).

Fact. Every fsm has a DFA accepting the same language.

Proof: Subset (powerset) construction

Apply to `arc`,`goal`, contra `Trans`,`Final`:

```
arcD(NodeList,NextList) :-
        setof(Next, arcLN(NodeList,Next), NextList).

arcLN(NodeList,Next) :- member(Node,NodeList),
                        arc(Node,Next).

goalD(NodeList):-member(Node,NodeList),goal(Node).
```

```
searchD(NL) :- goalD(NL);
               (arcD(NL,NL2), searchD(NL2)).
```

Determinization (eliminate choice)

A fsm [Trans, Final, Q0] such that

for all [Q,X,Qn] *and* [Q,X,Qn′] *in* Trans, $\mathtt{Qn} = \mathtt{Qn}'$

is a *deterministic finite automaton* (DFA).

Fact. Every fsm has a DFA accepting the same language.

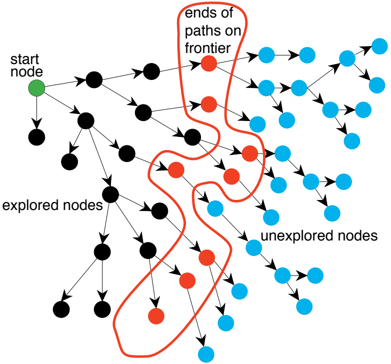
Proof: Subset (powerset) construction

Apply to arc,goal, contra Trans,Final:

```
arcD(NodeList,NextList) :-
        setof(Next, arcLN(NodeList,Next), NextList).

arcLN(NodeList,Next) :- member(Node,NodeList),
                        arc(Node,Next).

goalD(NodeList):-member(Node,NodeList),goal(Node).
```

```
searchD(NL) :- goalD(NL);
               (arcD(NL,NL2), searchD(NL2)).

search(Node) :- searchD([Node]).
```

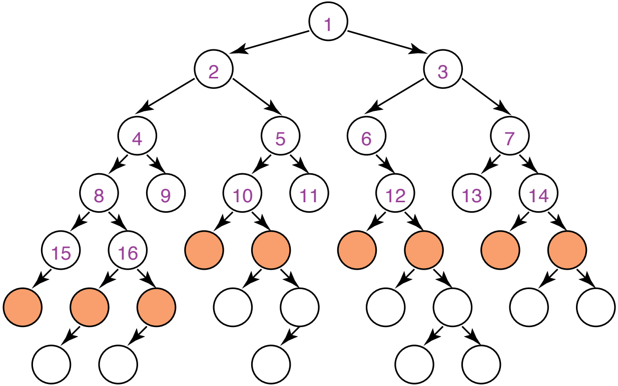
Frontier search



Poole & Mackworth

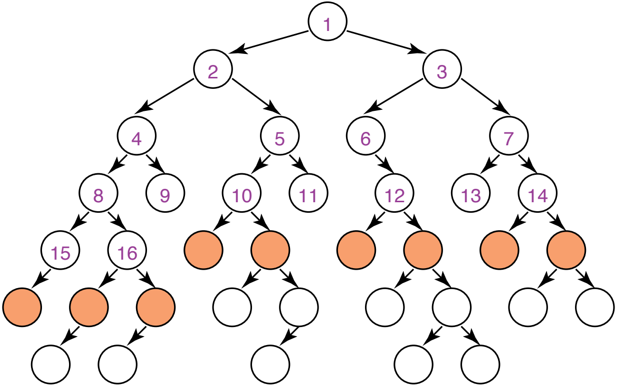
```
search(Node) :- frontierSearch([Node]).

frontierSearch([Node|_]) :- goal(Node).

frontierSearch([Node|Rest]) :-
            findall(Next, arc(Node,Next), Children),
            add2frontier(Children,Rest,NewFrontier),
            frontierSearch(NewFrontier).
```

Breadth-first: queue (FIFO)



`[1]` $\rightsquigarrow$ `[2,3]` $\rightsquigarrow$ `[3,4,5]` $\rightsquigarrow$ `[4,5,6,7]` $\rightsquigarrow \cdots$

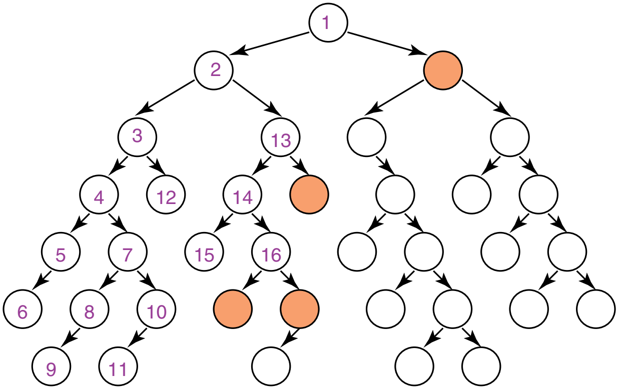
Breadth-first: queue (FIFO)



$[1] \rightsquigarrow [2,3] \rightsquigarrow [3,4,5] \rightsquigarrow [4,5,6,7] \rightsquigarrow \cdots$

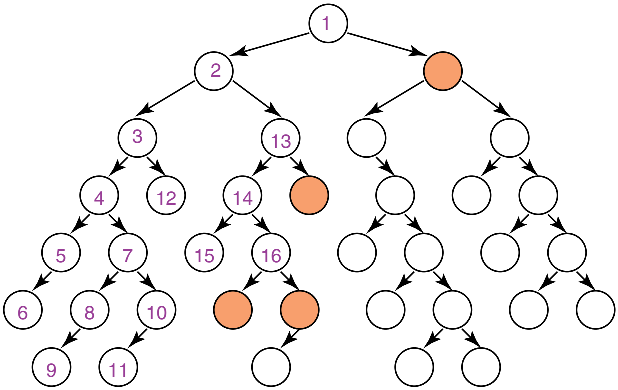
```
add2frontier(Children,[],Children).

add2frontier(Children,[H|T],[H|More]) :-
                              add2frontier(Children,T,More).
```

Depth-first: stack (LIFO)



$[1] \rightsquigarrow [2,\bullet] \rightsquigarrow [3,13,\bullet] \rightsquigarrow [4,12,13,\bullet] \rightsquigarrow \cdots$

Depth-first: stack (LIFO)



[1] $\rightsquigarrow$ [2,•] $\rightsquigarrow$ [3,13,•] $\rightsquigarrow$ [4,12,13,•] $\rightsquigarrow \cdots$

```
add2frontier([],Rest,Rest).

add2frontier([H|T],Rest,[H|TRest]) :-
                        add2frontier(T,Rest,TRest).
```

If-then-else and cut !

```
i :- p,!,q.

i :- r.

p.

r.
```

```
| ?- i.
```

If-then-else and cut !

```
i :- p,!,q.                    [i]

i :- r.

p.

r.
```

```
| ?- i.
```

If-then-else and cut !

```
i :- p,!,q.

i :- r.

p.

r.
```

```
| ?- i.
```

$$
\begin{tikzcd}
 & \texttt{[i]} \arrow[dl] \arrow[dr] & \\
\texttt{[p,!,q]} & & \texttt{[r]}
\end{tikzcd}
$$

If-then-else and cut !

```
i :- p,!,q.

i :- r.

p.

r.
```

```
| ?- i.
```

$$\begin{tikzcd}
& {[i]} \arrow[dl] \arrow[dr] & \\
{[p,!,q]} \arrow[d] & & {[r]} \\
{[!,q]} & &
\end{tikzcd}$$

Cut ! is true but destroys backtracking.

If-then-else and cut !

```
i :- p,!,q.

i :- r.

p.

r.
```

```
| ?- i.
```

$$
\begin{tikzcd}
 & {[i]} \arrow[dl] \\
{[p,!,q]} \arrow[d] & \\
{[!,q]} \arrow[d] & \\
{[q]} &
\end{tikzcd}
$$

Cut ! is true but destroys backtracking.

If-then-else and cut !

```
i :- p,!,q.

i :- r.

p.

r.
```

$$\begin{array}{c} \texttt{[i]} \\ \swarrow \\ \texttt{[p,!,q]} \\ \downarrow \\ \texttt{[!,q]} \\ \downarrow \\ \texttt{[q]} \end{array}$$

```
| ?- i.

no
```

Cut ! is true but destroys backtracking.

Review: Depth-first as frontier search

```
prove([],_).        % goal([]).
prove(Node,KB) :- arc(Node,Next,KB), prove(Next,KB).
```

Review: Depth-first as frontier search

```
prove([],_).      % goal([]).
prove(Node,KB) :- arc(Node,Next,KB), prove(Next,KB).
```

```
fs([[]|_],_).

fs([Node|More],KB) :- findall(X,arc(Node,X,KB),L),
                      append(L,More,NewFrontier),
                      fs(NewFrontier,KB).
```

Review: Depth-first as frontier search

```
prove([],_).      % goal([]).
prove(Node,KB) :- arc(Node,Next,KB), prove(Next,KB).


fs([[]|_],_).

fs([Node|More],KB) :- findall(X,arc(Node,X,KB),L),
                      append(L,More,NewFrontier),
                      fs(NewFrontier,KB).
```

Cut?

Tracking the frontier

[[i]]

```
i :- p,!,q.                    [i]

i :- r.

p.

r.
```

```
| ?- i.
```

Tracking the frontier

$\texttt{[[i]]} \rightsquigarrow \texttt{[[p,!,q],[r]]}$

```
i :- p,!,q.

i :- r.

p.

r.
```

```
| ?- i.
```

$$\begin{tikzcd} & \texttt{[i]} \arrow[dl] \arrow[dr] & \\ \texttt{[p,!,q]} & & \texttt{[r]} \end{tikzcd}$$

Tracking the frontier

`[[i]]` $\rightsquigarrow$ `[[p,!,q],[r]]` $\rightsquigarrow$ `[[!,q],[r]]`

```
i :- p,!,q.

i :- r.

p.

r.
```

```
| ?- i.
```

$$
\begin{tikzcd}
 & \texttt{[i]} \arrow[dl] \arrow[dr] & \\
\texttt{[p,!,q]} \arrow[d] & & \texttt{[r]} \\
\texttt{[!,q]} & &
\end{tikzcd}
$$

Tracking the frontier

`[[i]]` $\rightsquigarrow$ `[[p,!,q],[r]]` $\rightsquigarrow$ `[[!,q],[r]]` $\rightsquigarrow$ `[[q]]`

```
i :- p,!,q.

i :- r.

p.

r.
```

```
      [i]
     ↙
[p,!,q]
   ↓
 [!,q]
   ↓
  [q]
```

```
| ?- i.
```

Tracking the frontier

`[[i]]` $\rightsquigarrow$ `[[p,!,q],[r]]` $\rightsquigarrow$ `[[!,q],[r]]` $\rightsquigarrow$ `[[q]]` $\rightsquigarrow$ `[]`

```
i :- p,!,q.

i :- r.

p.

r.
```

```
| ?- i.
```

`[i]` $\swarrow$ `[p,!,q]` $\downarrow$ `[!,q]` $\downarrow$ `[q]`

Tracking the frontier

`[[i]]` $\rightsquigarrow$ `[[p,!,q],[r]]` $\rightsquigarrow$ `[[!,q],[r]]` $\rightsquigarrow$ `[[q]]` $\rightsquigarrow$ `[]`

```
i :- p,!,q.

i :- r.

p.

r.
```

```
| ?- i.

no
```

`[i]`

$\swarrow$

`[p,!,q]`

$\downarrow$

`[!,q]`

$\downarrow$

`[q]`

Cut via frontier depth-first search

```
fs([[]|_],_).



fs([Node|More],KB) :-
                        findall(X,arc(Node,X,KB),L),
                        append(L,More,NewFrontier),
                        fs(NewFrontier,KB).
```

Cut via frontier depth-first search

```
fs([[]|_],_).

fs([[cut|T]|_],KB)) :- fs([T],KB).

fs([Node|More],KB) :-
                     findall(X,arc(Node,X,KB),L),
                     append(L,More,NewFrontier),
                     fs(NewFrontier,KB).
```

Cut via frontier depth-first search

```
fs([[]|_],_).

fs([[cut|T]|_],KB)) :- fs([T],KB).

fs([Node|More],KB) :-  Node = [H|_], H\== cut,
                       findall(X,arc(Node,X,KB),L),
                       append(L,More,NewFrontier),
                       fs(NewFrontier,KB).
```

Cut via frontier depth-first search

```
fs([[]|_],_).

fs([[cut|T]|_],KB)) :- fs([T],KB).

fs([Node|More],KB) :-  Node = [H|_], H\== cut,
                       findall(X,arc(Node,X,KB),L),
                       append(L,More,NewFrontier),
                       fs(NewFrontier,KB).


if(p,q,r) :- (p,!,q); r.      % contra (p,q);r
```

Cut via frontier depth-first search

```
fs([[]|_],_).

fs([[cut|T]|_],KB)) :- fs([T],KB).

fs([Node|More],KB) :-  Node = [H|_], H\== cut,
                       findall(X,arc(Node,X,KB),L),
                       append(L,More,NewFrontier),
                       fs(NewFrontier,KB).


if(p,q,r) :- (p,!,q); r.       % contra (p,q);r

negation-as-failure(p) :- (p,!,fail); true.
```