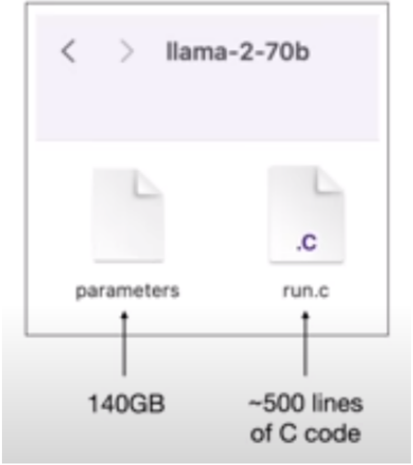
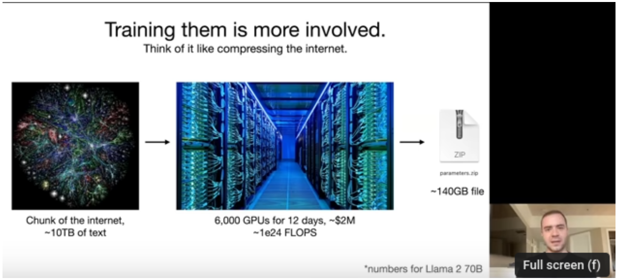
An open source large language model

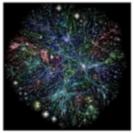


From A.Karpathy



Training them is more involved.

Think of it like compressing the internet.

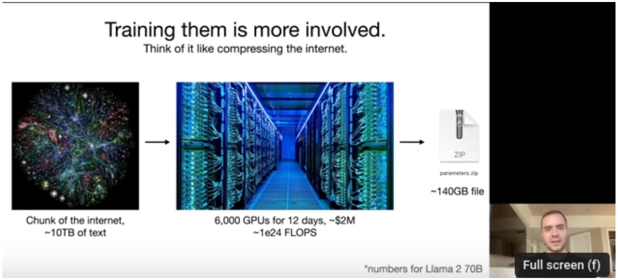


Chunk of the internet, ~10TB of text

6,000 GPUs for 12 days, ~$2M ~1e24 FLOPS

~140GB file

*numbers for Llama 2 70B



Training them is more involved.

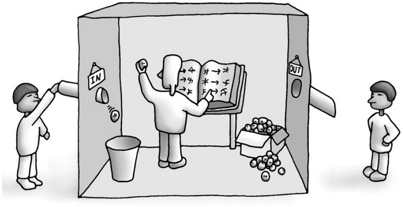
Think of it like compressing the internet.

Chunk of the internet, ~10TB of text

6,000 GPUs for 12 days, ~$2M ~1e24 FLOPS

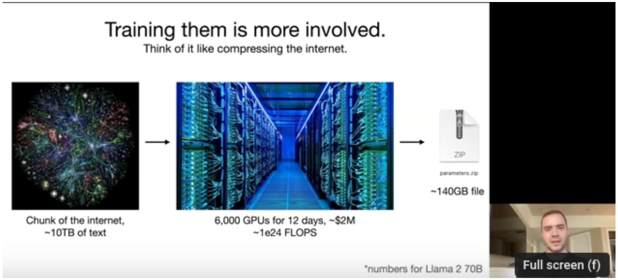
parameters.zip ~140GB file

*numbers for Llama 2 70B



Mindless obedience

From http://america.pink/images/9/6/3/2/5/4/en/2-chinese-room.jpg



Training them is more involved.

Think of it like compressing the internet.

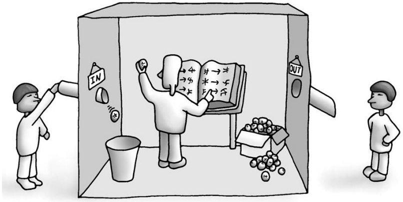
Chunk of the internet, ~10TB of text

6,000 GPUs for 12 days, ~$2M ~1e24 FLOPS

parameters.zip

~140GB file

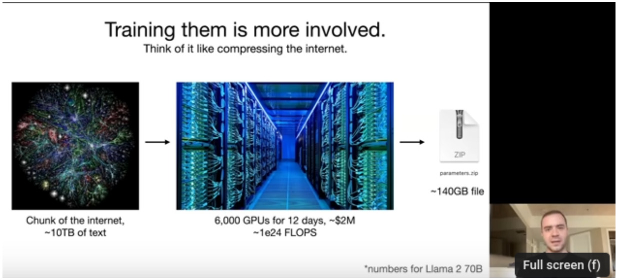
*numbers for Llama 2 70B



Mindless obedience

HP: does P halt on D?

From http://america.pink/images/9/6/3/2/5/4/en/2-chinese-room.jpg

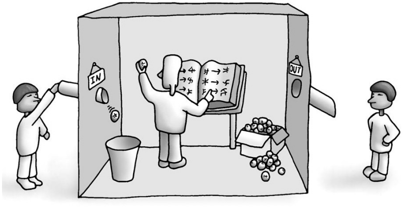


Training them is more involved.

Think of it like compressing the internet.

Chunk of the internet, ~10TB of text → 6,000 GPUs for 12 days, ~$2M ~1e24 FLOPS → parameters.zip ~140GB file

*numbers for Llama 2 70B

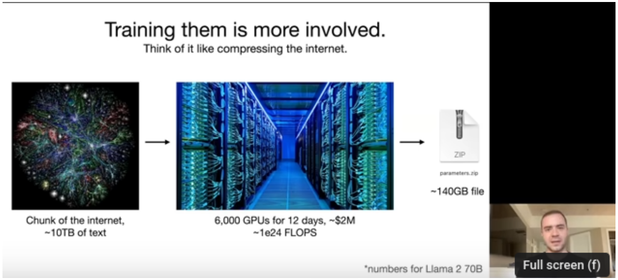


From http://america.pink/images/9/6/3/2/5/4/en/2-chinese-room.jpg

Mindless obedience

HP: does P halt on D?
Semi-solvable:

```
    run $P$ on $D$;
    return 1.
```



Training them is more involved.

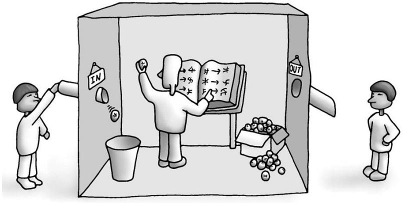
Think of it like compressing the internet.

Chunk of the internet, ~10TB of text

6,000 GPUs for 12 days, ~$2M ~1e24 FLOPS

~140GB file

*numbers for Llama 2 70B



From http://america.pink/images/9/6/3/2/5/4/en/2-chinese-room.jpg

Mindless obedience

HP: does P halt on D?

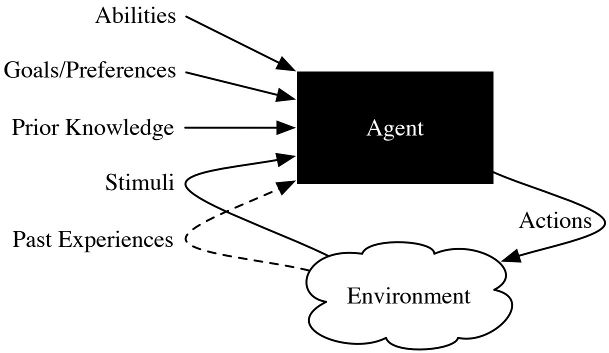
Semi-solvable:

```
    run $P$ on $D$;
    return 1.
```

$$\begin{array}{rcll} \text{run} & \approx & \text{run.c}, \quad U, & \text{accept} \\ P & \approx & \text{parameters, TM}, & \text{fsm} \end{array}$$

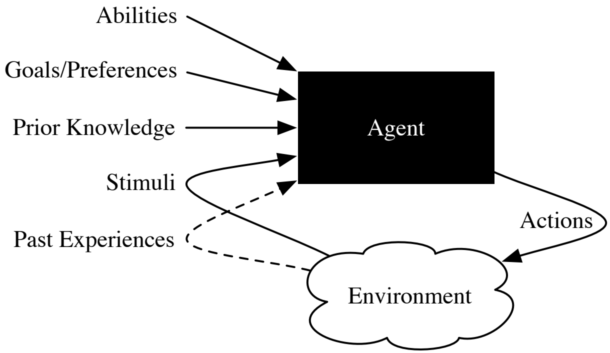
Fsm exercise: solution

```
accept(_,Final,Q,[]) :-

accept(Trans,Final,Q,[H|T]) :-
```



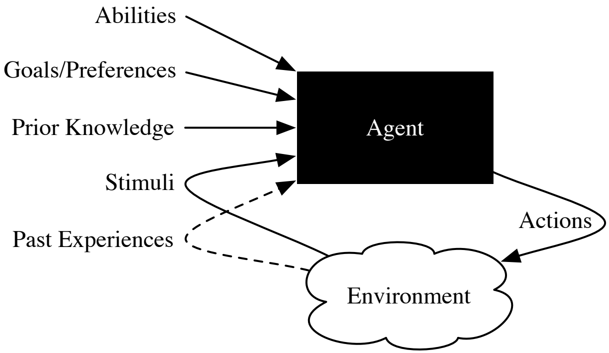
Fsm exercise: solution

```
accept(_,Final,Q,[]) :-  member(Q,Final).

accept(Trans,Final,Q,[H|T]) :-




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

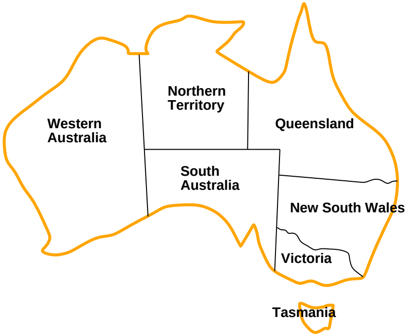


Fsm exercise: solution

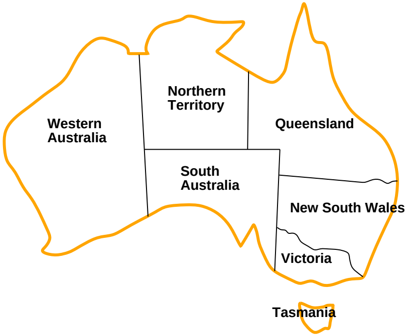
```
accept(_,Final,Q,[]) :-  member(Q,Final).

accept(Trans,Final,Q,[H|T]) :-
                          member([Q,H,Qn],Trans),
                          accept(Trans,Final,Qn,T).


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

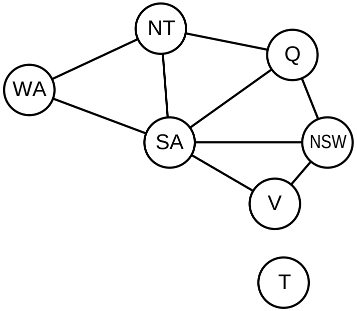


Graph modeling



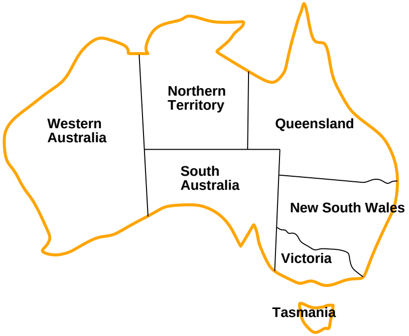
Graph modeling

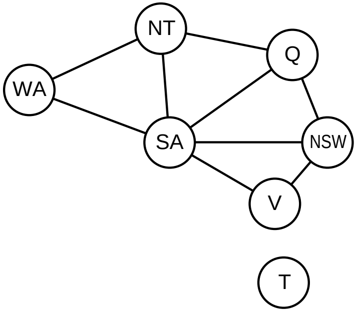




Russell & Norvig

Graph modeling





Russell & Norvig

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

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

Search (in Prolog)

Given `goal`, `arc`

```
search(Node) :- goal(Node).

search(Node) :- arc(Node,Next), search(Next).
```

Search (in Prolog)

Given `goal`, `arc`

```
search(Node) :- goal(Node).

search(Node) :- arc(Node,Next), search(Next).
```

Example: `accept(Trans,Final,Q0,String)`
`Node` as `[Q,UnseenString]`

Search (in Prolog)

Given `goal`, `arc`

```
search(Node) :- goal(Node).

search(Node) :- arc(Node,Next), search(Next).
```

Example: `accept(Trans,Final,Q0,String)`
`Node as [Q,UnseenString]`

```
goal([Q,[]],Final) :- member(Q,Final).
```

Search (in Prolog)

Given `goal`, `arc`

```
search(Node) :- goal(Node).

search(Node) :- arc(Node,Next), search(Next).
```

Example: `accept(Trans,Final,Q0,String)`
Node as `[Q,UnseenString]`

```
goal([Q,[]],Final) :- member(Q,Final).

arc([Q,[H|T]],[Qn,T],Trans) :-
                              member([Q,H,Qn],Trans).
```

Search (in Prolog)

Given `goal`, `arc`

```
search(Node) :- goal(Node).

search(Node) :- arc(Node,Next), search(Next).
```

Example: `accept(Trans,Final,Q0,String)`
Node as `[Q,UnseenString]`

```
goal([Q,[]],Final) :- member(Q,Final).

arc([Q,[H|T]],[Qn,T],Trans) :-
                              member([Q,H,Qn],Trans).


search(Node,Final,_) :- goal(Node,Final).

search(Node,Fi,Tr) :- arc(Node,Next,Tr),
                      search(Next,Fi,Tr).
```

Search (in Prolog)

Given `goal`, `arc`

```
search(Node) :- goal(Node).

search(Node) :- arc(Node,Next), search(Next).
```

Example: `accept(Trans,Final,Q0,String)`
Node as `[Q,UnseenString]`

```
goal([Q,[]],Final) :- member(Q,Final).

arc([Q,[H|T]],[Qn,T],Trans) :-
                          member([Q,H,Qn],Trans).


search(Node,Final,_) :- goal(Node,Final).

search(Node,Fi,Tr) :- arc(Node,Next,Tr),
                      search(Next,Fi,Tr).

accept(Tr,Fi,Q0,S) :- search([Q0,S],Fi,Tr).
```

Prolog as search

```
i :- p,q.

i :- r.

p.

r.
```

```
| ?- i.
```

Prolog as search

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

i :- r.

p.

r.

______________

| ?- i.               StartNode = [i]
```

Prolog as search

```
i :- p,q.

i :- r.

p.

r.
```

```
| ?- i.
```

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

```
StartNode = [i]
```

Prolog as search

```
i :- p,q.

i :- r.

p.

r.
```

```
| ?- i.
```

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

```
StartNode = [i]
```

Prolog as search

```
i :- p,q.

i :- r.

p.

r.
```

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

```
| ?- i.
```

```
StartNode = [i]
```

Prolog as search

```
i :- p,q.

i :- r.

p.

r.
```

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

```
| ?- i.

yes
```

```
StartNode = [i]

goal([]).
```

Prolog as search

```
i :- p,q.

i :- r.

p.

r.
```

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

```
| ?- i.

yes
```

```
StartNode = [i]

goal([]).
```

```
prove(Node) :- goal(Node) .
prove(Node) :- arc(Node,Next), prove(Next).
```

KB and arc

```
i :- p,q.

i :- r.

p.

r.
```

KB and arc

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

KB and arc

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

```
KB = [[i,p,q],[i,r],[p],[r]]
```

KB and arc

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

i :- r.                                 [i,r]

p.                                       [p]

r.                                       [r]

                          KB = [[i,p,q],[i,r],[p],[r]]


arc(Node1,Node2,KB) :- ??
```

KB and arc

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

i :- r.                      [i,r]

p.                           [p]

r.                           [r]

             KB = [[i,p,q],[i,r],[p],[r]]
```

```
arc([H|T],N,KB) :- member([H|B],KB), append(B,T,N).
```

KB and arc

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

i :- r.                           [i,r]

p.                                 [p]

r.                                 [r]

                     KB = [[i,p,q],[i,r],[p],[r]]
```

```
arc([H|T],N,KB) :- member([H|B],KB), append(B,T,N).

prove(Node,KB) :- goal(Node) ;
                  arc(Node,Next,KB), prove(Next,KB).
```

Non-termination (due to poor choice)

```
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 choice)

```
i :- p,q.

i :- r.

p :- i.

r.
________

| ?- i.
```

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

```
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 choice)

```
i :- p,q.

i :- r.

p :- i.

r.
```

```
| ?- i.
```

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

```
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 choice)

```
i :- p,q.

i :- r.

p :- i.

r.
────────
| ?- i.
```

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

```
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 choice)

```
i :- p,q.

i :- r.

p :- i.

r.
________

| ?- i.
```

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

```
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 choice)

```
i :- p,q.

i :- r.

p :- i.

r.
________

| ?- i.
```

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

```
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 (do all)

A fsm [Trans, Final, Q0] such that

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

is a *deterministic finite automaton* (DFA).

Determinization (do all)

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.

Determinization (do all)

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 (do all)

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 (do all)

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 (do all)

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

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

is a *deterministic finite automaton* (DFA).

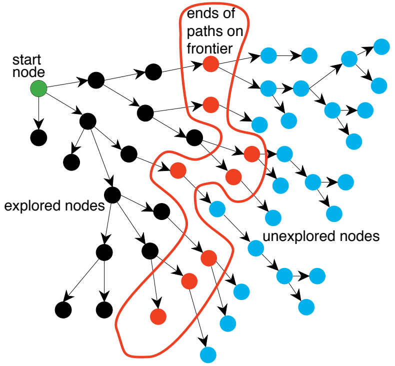
Fact. Every fsm has a DFA accepting the same language.

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]).
```

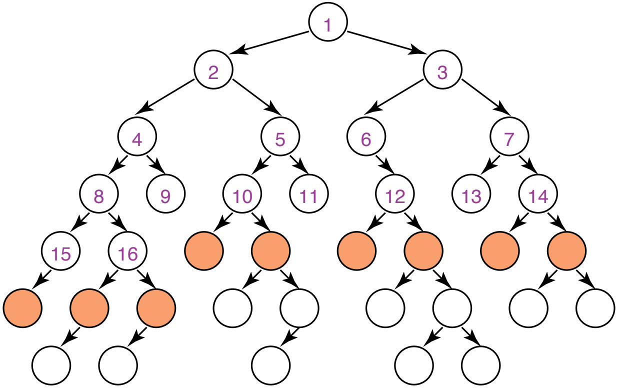
Frontier search (manage choices)



Poole & Mackworth

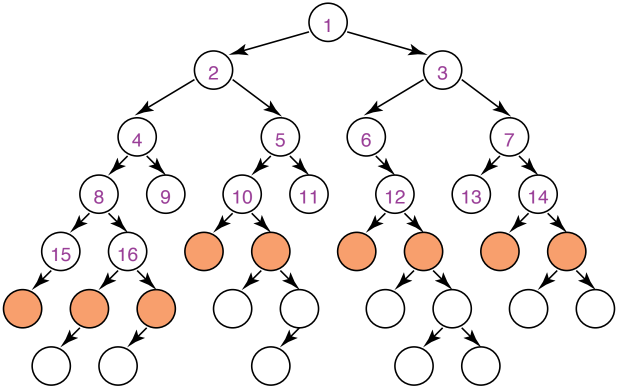
```
search(Node) :- frontierSearch([Node]).

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

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

Breadth-first: queue (FIFO)



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

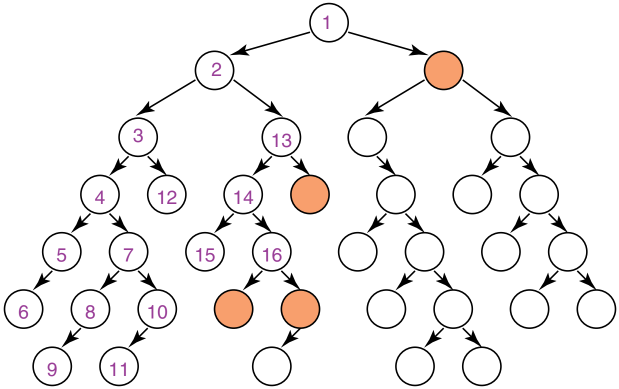
Breadth-first: queue (FIFO)



[1] ⇝ [2,3] ⇝ [3,4,5] ⇝ [4,5,6,7] ⇝ · · ·

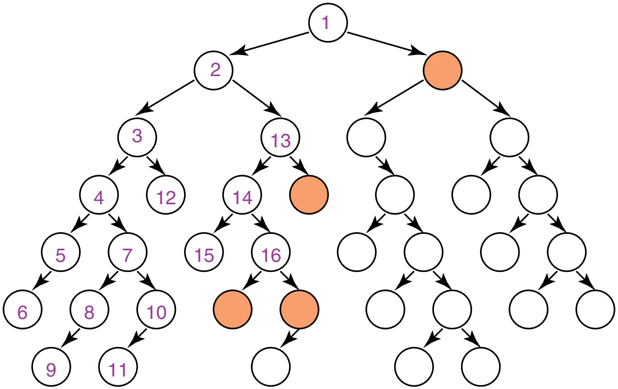
```
add2frontier(Children,[],Children).

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

Depth-first: stack (LIFO)



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

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.
```

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

If-then-else and cut !

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

i :- r.

p.

r.
```

```
| ?- i.
```

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

Cut ! is true but destroys backtracking.

If-then-else and cut !

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

i :- r.

p.

r.
```

$$\begin{tikzcd}[column sep=small] & \texttt{[i]} \arrow[dl] \\ \texttt{[p,!,q]} \arrow[d] & \\ \texttt{[!,q]} \arrow[d] & \\ \texttt{[q]} & \end{tikzcd}$$

```
| ?- i.
```

Cut ! is true but destroys backtracking.

If-then-else and cut !

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

i :- r.

p.

r.
```

```
| ?- i.

no
```

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

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|Rest],KB) :-
          findall(X,arc(Node,X,KB),Children),
          append(Children,Rest,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|Rest],KB) :-
          findall(X,arc(Node,X,KB),Children),
          append(Children,Rest,NewFrontier),
          fs(NewFrontier,KB).
```

Cut?

Tracking the frontier

```
[[i]]
```

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

i :- r.

p.

r.
```

```
| ?- i.
```

Tracking the frontier

`[[i]]` $\leadsto$ `[[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

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

```
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.
```

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

Tracking the frontier

```
[[i]] ⇝ [[p,!,q],[r]] ⇝ [[!,q],[r]] ⇝ [[q]] ⇝ []
```

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

i :- r.

p.

r.
```

```
| ?- i.
```

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

Tracking the frontier

```
[[i]] ⇝ [[p,!,q],[r]] ⇝ [[!,q],[r]] ⇝ [[q]] ⇝ []
```

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

i :- r.

p.

r.
```

```
| ?- i.

no
```

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

Cut via frontier depth-first search

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



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

Cut via frontier depth-first search

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

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

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

Cut via frontier depth-first search

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

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

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

Cut via frontier depth-first search

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

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

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


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

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

Exercise (Prolog)

Suppose a positive integer `Seed` links nodes `1,2,...` in two ways

```
arc(N,M,Seed) :- M is N*Seed.
arc(N,M,Seed) :- M is N*Seed +1.
```

e.g. `Seed`=3 gives arcs `(1,3), (1,4), (3,9), (3, 10) ...`

Exercise (Prolog)

Suppose a positive integer `Seed` links nodes `1,2,...` in two ways

```
arc(N,M,Seed) :- M is N*Seed.
arc(N,M,Seed) :- M is N*Seed +1.
```

e.g. `Seed`=3 gives arcs `(1,3), (1,4), (3,9), (3, 10) ...`

Goal nodes are multiples of a positive integer `Target`

```
goal(N,Target) :- 0 is N mod Target.
```

e.g. `Target`=13 gives goals `13, 26, 39 ...`

Exercise (Prolog)

Suppose a positive integer `Seed` links nodes `1,2,...` in two ways

```
arc(N,M,Seed) :- M is N*Seed.
arc(N,M,Seed) :- M is N*Seed +1.
```

e.g. `Seed`=3 gives arcs `(1,3), (1,4), (3,9), (3, 10) ...`

Goal nodes are multiples of a positive integer `Target`

```
goal(N,Target) :- 0 is N mod Target.
```

e.g. `Target`=13 gives goals `13, 26, 39 ...`

Modify frontier search to define predicates

```
breadth1st(+Start, ?Found, +Seed, +Target)
depth1st(+Start, ?Found, +Seed, +Target)
```

that search breadth-first and depth-first respectively for a `Target`-goal node `Found` linked to `Start` by `Seed`-arcs.