```
% search(Agenda,Goal) <- Goal is a goal node, and a
%                        descendant of one of the nodes
%                        on the Agenda
search(Agenda,Goal):-
  next(Agenda,Goal,Rest),
  goal(Goal).
search(Agenda,Goal):-
  next(Agenda,Current,Rest),
  children(Current,Children),
  add(Children,Rest,NewAgenda),
  search(NewAgenda,Goal).
```

Agenda-based search


```
search_df([Goal|Rest],Goal):-
  goal(Goal).
search_df([Current|Rest],Goal):-
  children(Current,Children),
  append(Children,Rest,NewAgenda),
  search_df(NewAgenda,Goal).

search_bf([Goal|Rest],Goal):-
  goal(Goal).
search_bf([Current|Rest],Goal):-
  children(Current,Children),
  append(Rest,Children,NewAgenda),
  search_bf(NewAgenda,Goal).

children(Node,Children):-
  findall(C,arc(Node,C),Children).
```

Depth-first vs. breadth-first search

- ☞ Breadth-first search
  - ✓ agenda = queue (first-in first-out)
  - ✓ complete: guaranteed to find all solutions
  - ✓ first solution founds along shortest path
  - ✓ requires $O(B^n)$ memory
- ☞ Depth-first search
  - ✓ agenda = stack (last-in first-out)
  - ✓ incomplete: may get trapped in infinite branch
  - ✓ no shortest-path property
  - ✓ requires $O(B \times n)$ memory

Depth-first vs. breadth-first search


```
% depth-first search with loop detection
search_df_loop([Goal|Rest],Visited,Goal):-
  goal(Goal).
search_df_loop([Current|Rest],Visited,Goal):-
  children(Current,Children),
  add_df(Children,Rest,Visited,NewAgenda),
  search_df_loop(NewAgenda,[Current|Visited],Goal).

add_df([],Agenda,Visited,Agenda).
add_df([Child|Rest],OldAgenda,Visited,[Child|NewAgenda]):-
  not element(Child,OldAgenda),
  not element(Child,Visited),
  add_df(Rest,OldAgenda,Visited,NewAgenda).
add_df([Child|Rest],OldAgenda,Visited,NewAgenda):-
  element(Child,OldAgenda),
  add_df(Rest,OldAgenda,Visited,NewAgenda).
add_df([Child|Rest],OldAgenda,Visited,NewAgenda):-
  element(Child,Visited),
  add_df(Rest,OldAgenda,Visited,NewAgenda).
```

Loop detection

```
% depth-first search by means of backtracking
search_bt(Goal,Goal):-
  goal(Goal).
search_bt(Current,Goal):-
  arc(Current,Child),
  search_bt(Child,Goal).


% backtracking depth-first search with depth bound
search_d(D,Goal,Goal):-
  goal(Goal).
search_d(D,Current,Goal):-
  D>0, D1 is D-1,
  arc(Current,Child),
  search_d(D1,Child,Goal).
```

Backtracking search

```
search_id(First,Goal):-
  search_id(1,First,Goal).      % start with depth 1
search_id(D,Current,Goal):-
  search_d(D,Current,Goal).
search_id(D,Current,Goal):-
  D1 is D+1,                     % increase depth
  search_id(D1,Current,Goal).
```

☞ combines advantages of
breadth-first search (complete, shortest path)
with those of depth-first search (memory-efficient)

Iterative deepening

```
prove(true):-!.
prove((A,B)):-!,
  clause(A,C),
  conj_append(C,B,D),
  prove(D).
prove(A):-
  clause(A,B),
  prove(B).
```

```
prove_df_a(Goal):-
  prove_df_a([Goal]).

prove_df_a([true|Agenda]).
prove_df_a([(A,B)|Agenda]):-!,
  findall(D,(clause(A,C),conj_append(C,B,D)),Children),
  append(Children,Agenda,NewAgenda),
  prove_df_a(NewAgenda).
prove_df_a([A|Agenda]):-
  findall(B,clause(A,B),Children),
  append(Children,Agenda,NewAgenda),
  prove_df_a(NewAgenda).
```

Agenda-based SLD-prover


```
refute((false:-true)).
refute((A,C)):-
  cl(Cl),
  resolve(A,Cl,R),
  refute(R).
```

```
% refute_bf(Clause) <- Clause is refuted by clauses
%                      defined by cl/1
%                      (breadth-first search strategy)
refute_bf_a(Clause):-
 refute_bf_a([a(Clause,Clause)],Clause).

refute_bf_a([a((false:-true),Clause)|Rest],Clause).
refute_bf_a([a(A,C)|Rest],Clause):-
 findall(a(R,C),(cl(Cl),resolve(A,Cl,R)),Children),
 append(Rest,Children,NewAgenda),% breadth-first
 refute_bf_a(NewAgenda,Clause).
```

Refutation prover for clausal logic

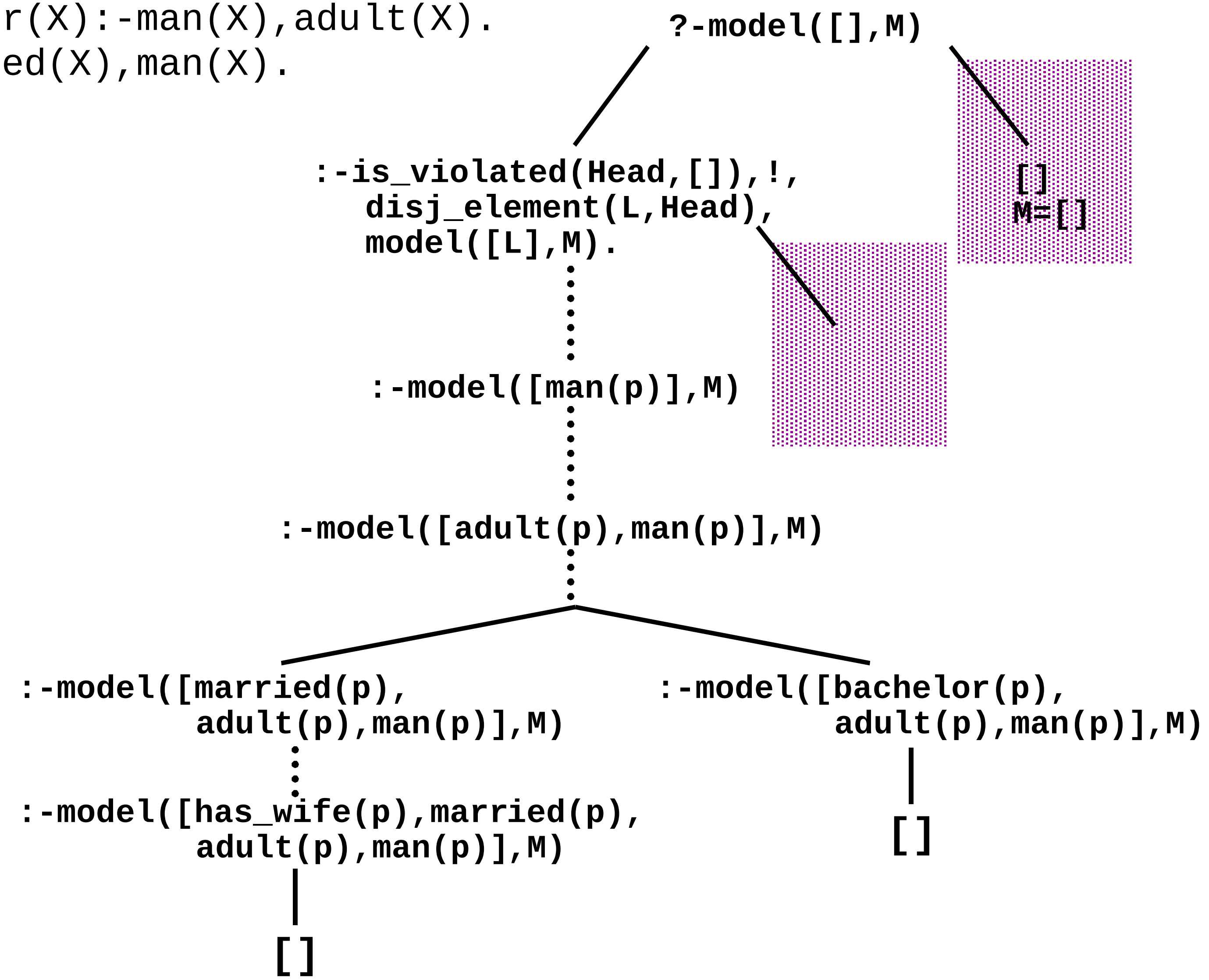
```
% model(M) <- M is a model of the clauses defined by cl/1
model(M):-
 model([],M).

model(M0,M):-
 is_violated(Head,M0),!,     % instance of violated clause
 disj_element(L,Head),       % L: ground literal from head
 model([L|M0],M).            % add L to the model
model(M,M).                  % no more violated clauses

is_violated(H,M):-
 cl((H:-B)),
 satisfied_body(B,M),        % grounds the variables
 not satisfied_head(H,M).
```

Forward chaining

```
married(X);bachelor(X):-man(X),adult(X).
has_wife(X):-married(X),man(X).
man(paul).
adult(paul).
```



Forward chaining: example

```
% model_d(D,M) <- M is a submodel of the clauses
%                 defined by cl/1
model_d(D,M):-
  model_d(D,[],M).

model_d(0,M,M).
model_d(D,M0,M):-
  D>0,D1 is D-1,
  findall(H,is_violated(H,M0),Heads),
  satisfy_clauses(Heads,M0,M1),
  model_d(D1,M1,M).

satisfy_clauses([],M,M).
satisfy_clauses([H|Hs],M0,M):-
  disj_element(L,H),
  satisfy_clauses(Hs,[L|M0],M).
```

Forward chaining with depth-bound