```
# ============================================================
#  Backward Chaining
# ============================================================
#  Knowledge Base: list of tuples (premises..., conclusion)
rules = [
   ("has_feathers", "has_wings",  "is_bird"),
   ("is_bird",      "can_fly",    "is_flying_bird"),
   ("lays_eggs",    "has_feathers","is_bird"),
]
# Working Memory: known facts
working_memory = ["has_feathers", "has_wings", "can_fly"]
#  Goal Stack: goals to be proved (start with final goal)
goal_stack = ["is_flying_bird"]
#  Visited: prevent infinite loops
visited = []
#  Results tracker
proved = []
failed = []
print(f"Initial working memory : {working_memory}")
print(f"Final goal             : '{goal_stack[0]}'")
# ============================================================
#  Main Loop - process goals one by one from the stack
# ============================================================
while goal_stack:
   # Pick the current goal from the top of the stack
   current_goal = goal_stack[-1]
   # ── Step 1: Already proved? ──────────────────────────────
   if current_goal in working_memory:
      print(f" '{current_goal}' found in working memory")
      proved.append(current_goal)
      goal_stack.pop()
      continue
```

```
# — Step 2: Already tried and failed? ————————————————
if current_goal in failed:
    print(f" '{current_goal}' already failed — removing from stack")
    goal_stack.pop()
    continue
# — Step 3: Mark as visited to avoid re-processing ————
if current_goal not in visited:
    visited.append(current_goal)
    print(f"\n Trying to prove: '{current_goal}'")
# — Step 4: Find rules whose THEN matches current goal ——
matching_rules = [r for r in rules if r[-1] == current_goal]

if not matching_rules:
    print(f" No rules to prove '{current_goal}'")
    failed.append(current_goal)
    goal_stack.pop()
    continue
# — Step 5: Check premises of matching rules ——————————
goal_proved = False
for rule in matching_rules:
    premises  = list(rule[:-1])   # IF parts  → list
    conclusion = rule[-1]         # THEN part → string
    # Check which premises are missing from working memory
    missing = [p for p in premises if p not in working_memory]
    if not missing:
        # All premises already known → conclude!
        print(f" Rule matched: {premises} → '{conclusion}'")
        working_memory.append(conclusion)
        proved.append(conclusion)
        goal_stack.pop()
        print(f" Proved '{conclusion}'! Added to working memory.")
        goal_proved = True
```

```
            break
        else:
            # Push missing premises onto the stack as sub-goals
            for sub in missing:
                if sub not in goal_stack and sub not in proved and sub not in failed:
                    print(f"    New sub-goal: '{sub}'")
                    goal_stack.append(sub)
    # If no rule fired, check if new sub-goals were pushed
    if not goal_proved:
        new_subgoals_pushed = any(
            p for r in matching_rules for p in r[:-1]
            if p in goal_stack and p != current_goal
        )
        if not new_subgoals_pushed:
            # Nothing new to try — mark as failed
            failed.append(current_goal)
            goal_stack.pop()
# ============================================================
#  Final Report
# ============================================================
final_goal = "is_flying_bird"
print(f"\n Result: '{final_goal}' → "
      f"{'PROVED ' if final_goal in working_memory else 'FAILED '}")
print(f" Final working memory : {working_memory}")
print(f" Proved goals         : {proved}")
print(f" Failed goals         : {failed}")
```

Execution

Initial working memory : ['has_feathers', 'has_wings', 'can_fly['

Final goal : 'is_flying_bird'

Trying to prove: 'is_flying_bird'

New sub-goal: 'is_bird'

Trying to prove: 'is_bird'

Rule matched: ['has_feathers', 'has_wings'] → 'is_bird'

Proved 'is_bird'! Added to working memory.

Rule matched: ['is_bird', 'can_fly'] → 'is_flying_bird'

Proved 'is_flying_bird'! Added to working memory.

Result: 'is_flying_bird' → PROVED

Final working memory : ['has_feathers', 'has_wings', 'can_fly', 'is_bird', 'is_flying_bird['

Proved goals : ['is_bird', 'is_flying_bird['

Failed goals : []