CST8503 Assignment 1 – Frank

Author: Frank
Date: 2025-09-14

Load the Knowledge Base

```
?- [family_tree].
```

1.1 What will be Prolog's answers to the following Prolog queries? (0.5 mark each)

a)

```
?- parent(brad, X).
false.
```

b)

```
?- parent(X, brad).
X = joe.
```

c)

```
?- parent(henry, X), parent(X, jeff).
X = susan.
```

d)

```
?- parent(henry, X), parent(X, Y), parent(Y, brad).
false.
```

1.2 Provide Prolog queries that correspond to the following questions about the parent relation: (0.5 mark each)

a) Who is brad's parent?

```
?- parent(X, brad).
X = joe.
```

b) Does josh have a child?

```
?- parent(josh, _).
false
```

c) Who is brad's grandparent? (limit to parent/2)

```
?- parent(X, Y), parent(Y, brad).
X = henry,
Y = joe
```

1.3 Translate the following statements into Prolog rules and add the resulting Prolog code to your program (0.5 mark each)

a) Everybody who has a child is educated.

```
educated(X) :- parent(X, _).
```

b) If X has a child who has a sibling, then X is poor.

```
poor(X) :- parent(X, C), sibling(C, S).
```

1.4 Define the relation grandchild using the parent relation. Hint: It will be similar to the grandparent relation below. Add the code for both grandparent and grandchild to your Prolog program. (0.5 mark)

```
grandparent(X, Z) :- parent(X, Y), parent(Y, Z).
grandchild(Z, X)  :- parent(X, Y), parent(Y, Z).
```

1.5 For each person named in our program, use male/1 and female/1 to assert that they are male or female. Then define a sister(X,Y) relation in terms of parent and female (X is the sister of Y if Z is parent of X, and Z is parent of Y, and X is female, and X is not the same as Y. There is a built-in predicate you can use, dif(X,Y) that means X is not the same as Y. Define the relation aunt(X,Y) in terms of the relations parent and sister. Add the prolog code to your program. (1 mark)

```
male(henry).
male(joe).
male(brad).
male(jeff).

female(susan).
female(julie).

sister(X, Y) :- parent(Z, X), parent(Z, Y), female(X), dif(X, Y).
aunt(X, Y)   :- sister(X, Z), parent(Z, Y).
```

Example checks:

```
?- sister(susan, joe).
true.

?- aunt(susan, brad).
true.
```

1.6 Ancestor – two alternative definitions

Prolog code (both are correct)

```
% Definition 1: top-down recursion (expand from X toward Z)
ancestor(X, Z) :- parent(X, Z).
ancestor(X, Z) :- parent(X, Y), ancestor(Y, Z).

% Definition 2: bottom-up recursion (work back from Z toward X)
ancestor(X, Z) :- parent(X, Z).
ancestor(X, Z) :- ancestor(X, Y), parent(Y, Z).
```

Why both are proper

Both definitions say “there is a path of one or more parent/2 links from X to Z.”
They only differ in **recursion direction**:

- Def. 1 grows the path forward: X → Y … → Z.
- Def. 2 grows the path backward: X … → Y → Z.
  With the base case `parent(X,Z)` present, each definition is logically complete.

Diagram A (Definition 1 – top-down)

```
X
|  parent
▼
Y
|  parent
▼
… (zero or more intermediates)
|  parent
▼
Z
======================== ancestor (X ⇒ Z)
```

Diagram B (Definition 2 – bottom-up)

```
X
|  ancestor (via zero or more intermediates)
▼
…
|
▼
Y
|  parent
▼
Z
======================== ancestor (X ⇒ Z)
```

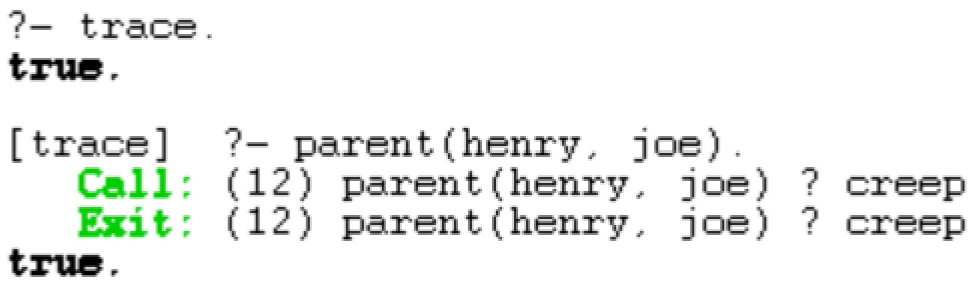
Prolog Execution Trace

```
Facts:
parent(a,b).
parent(b,c).
parent(c,d).

Rules:
Rule 1 (base case): ancestor(X,Z) :- parent(X,Z).
Rule 2 (recusive case): ancestor(X,Z) :- ancestor(X,Y), parent(Y,Z).

Query:
?- ancestor(a,d).

Level 0: Call: ancestor(a,d), {X=a, Z=d}
Level 1: ├─ Rule 1: ancestor(X,Z) :- parent(X,Z)
Level 1: |     Call: parent(a,d)
Level 1: |     ✗ Fail
Level 1: |
Level 1: └─ Rule 2: ancestor(X,Z) :- ancestor(X,Y), parent(Y,Z)
Level 1:       Head unify with goal ancestor(a,d) ⇒ Assign: X := a, Z := d
Level 1:       Standardize-apart: Y → C1
Level 1:       [Level 1 Rule 2 Left ]  Call: ancestor(a,C1)
Level 1:       [Level 1 Rule 2 Right] Call: parent(C1,d)        ; Check after C1
is bound

Level 2:       ├─ Rule 1: ancestor(X,Z) :- parent(X,Z)
Level 2:       |     Call: parent(a,C1)
Level 2:       |          → unify with fact parent(a,b) ✓
Level 2:       |          ⇒ Assign: C1 := b
Level 2:       |          ⇒ Env: {X=a, Z=d, C1=b}
Level 2:       |     [Level 1 Rule 2 Right] Call: parent(C1,d)
Level 2:       |          Substitute C1=b ⇒ Call: parent(b,d)
Level 2:       |          ✗ Fail → backtrack
Level 2:       |          ⇒ Unbind on backtrack: C1
Level 2:       |          ⇒ Env: {X=a, Z=d}
Level 2:       |
Level 2:       └─ Rule 2: ancestor(X,Z) :- ancestor(X,Y), parent(Y,Z)
Level 2:             Head unify with goal ancestor(a,C1) ⇒ Assign: X := a, Z :=
C1
Level 2:             Standardize-apart: Y → C2
Level 2:             [Level 2 Rule 2 Left ]  Call: ancestor(a,C2)
Level 2:             [Level 2 Rule 2 Right] Call: parent(C2,C1)  ; Check after C2
is bound

Level 3:             ├─ Rule 1: ancestor(X,Z) :- parent(X,Z)
Level 3:             |     Call: parent(a,C2)
Level 3:             |          → unify with fact parent(a,b) ✓
Level 3:             |          ⇒ Assign: C2 := b
Level 3:             |          ⇒ Env: {X=a, Z=d, C2=b}
Level 3:             |     [Level 2 Rule 2 Right] Call: parent(C2,C1)
Level 3:             |          Substitute C2=b ⇒ Call: parent(b,C1)
Level 3:             |          → unify with fact parent(b,c) ✓
Level 3:             |          ⇒ Assign: C1 := c
Level 3:             |          ⇒ Env: {X=a, Z=d, C2=b, C1=c}
Level 3:             |     → ancestor(a,C1) ✓ with {C1=c, C2=b}
Level 3:             |
Level 3:             Return to [Level 1 Rule 2 Right]
Level 3:             Call: parent(C1,d)
Level 3:                  Substitute C1=c ⇒ Call: parent(c,d)
Level 3:                  ✓ Success
Level 3:                  ⇒ Final Env: {X=a, Z=d, C1=c, C2=b}
Level 3:                  → Final Success: ancestor(a,d), path a→b→c→d
```

1.7 Try to understand how Prolog derives answers to the following queries, using your Prolog code from this assignment. Open your Prolog program file in a window so that you can view your program and this question at the same time. Trace through, in your mind, what the Prolog algorithm will do with your program when the following queries are issued. Show your work similar to the example, showing the intermediate goals indented, and draw an X where Prolog will backtrack to the most recent choice point. (3 marks)

a)

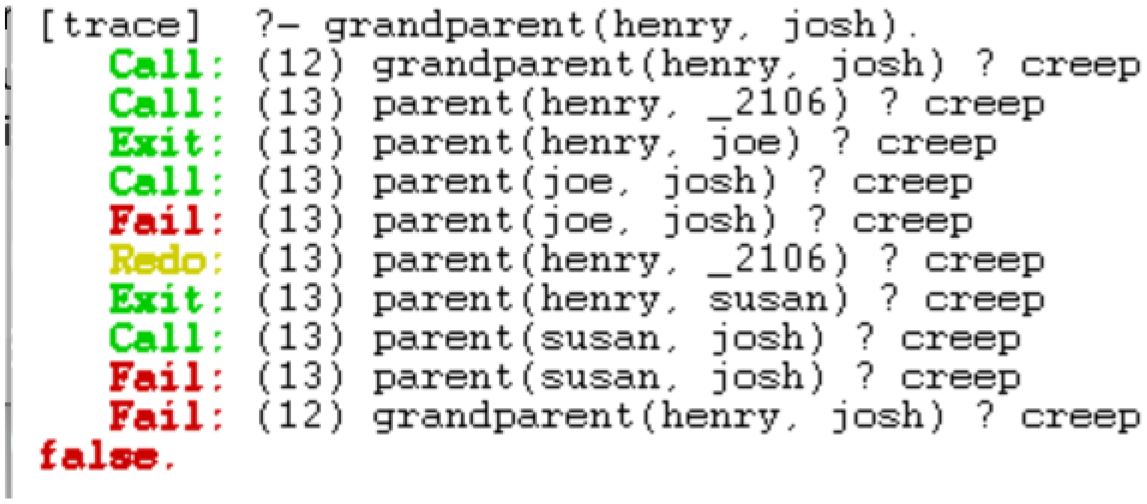
```
?- parent(henry, joe).
    ✓ fact parent(henry, joe) succeeds
```



```
?- trace.
true.

[trace]  ?- parent(henry, joe).
   Call: (12) parent(henry, joe) ? creep
   Exit: (12) parent(henry, joe) ? creep
true.
```

b)

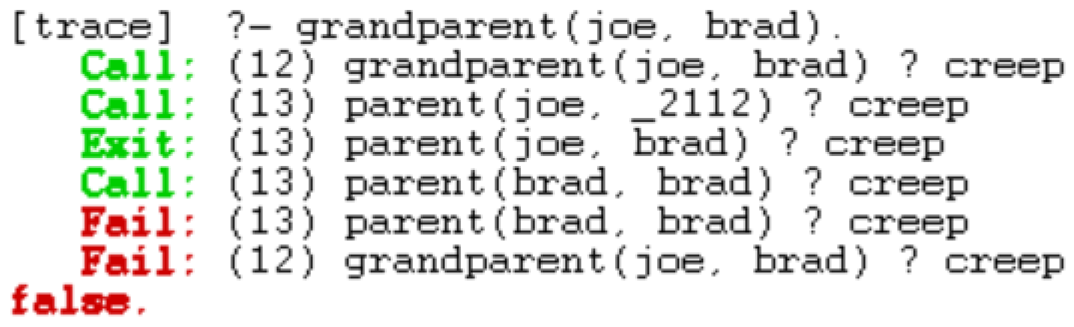
```
?- grandparent(henry, josh).
        parent(henry,Y).
        parent(henry,joe).
        parent(joe,josh).
        X backtrack
        parent(henry,Y).
        parent(henry,susan).
        parent(susan,josh).
        X backtrack
        X no more alternatives, fail.
```



```
[trace]  ?- grandparent(henry, josh).
   Call: (12) grandparent(henry, josh) ? creep
   Call: (13) parent(henry, _2106) ? creep
   Exit: (13) parent(henry, joe) ? creep
   Call: (13) parent(joe, josh) ? creep
   Fail: (13) parent(joe, josh) ? creep
   Redo: (13) parent(henry, _2106) ? creep
   Exit: (13) parent(henry, susan) ? creep
   Call: (13) parent(susan, josh) ? creep
   Fail: (13) parent(susan, josh) ? creep
   Fail: (12) grandparent(henry, josh) ? creep
false.
```

c)

```
?- grandparent(joe, brad).
        parent(joe,Y).
        parent(joe,brad).
        parent(brad,brad).
        X backtrack
        X no more alternatives, fail.
```



```
[trace]  ?- grandparent(joe, brad).
   Call: (12) grandparent(joe, brad) ? creep
   Call: (13) parent(joe, _2112) ? creep
   Exit: (13) parent(joe, brad) ? creep
   Call: (13) parent(brad, brad) ? creep
   Fail: (13) parent(brad, brad) ? creep
   Fail: (12) grandparent(joe, brad) ? creep
false.
```