Prolog

- Declarative/logic paradigm

Prolog

- Declarative/logic paradigm
- Functional paradigm – No assignment statement

Prolog

- Declarative/logic paradigm
- Functional paradigm – No assignment statement
- Declarative paradigm – No program! Specification without implementation.

Using Prolog

- Two shells
- `vi` to edit and save the database, or `more` to view it
- Prolog to query the database

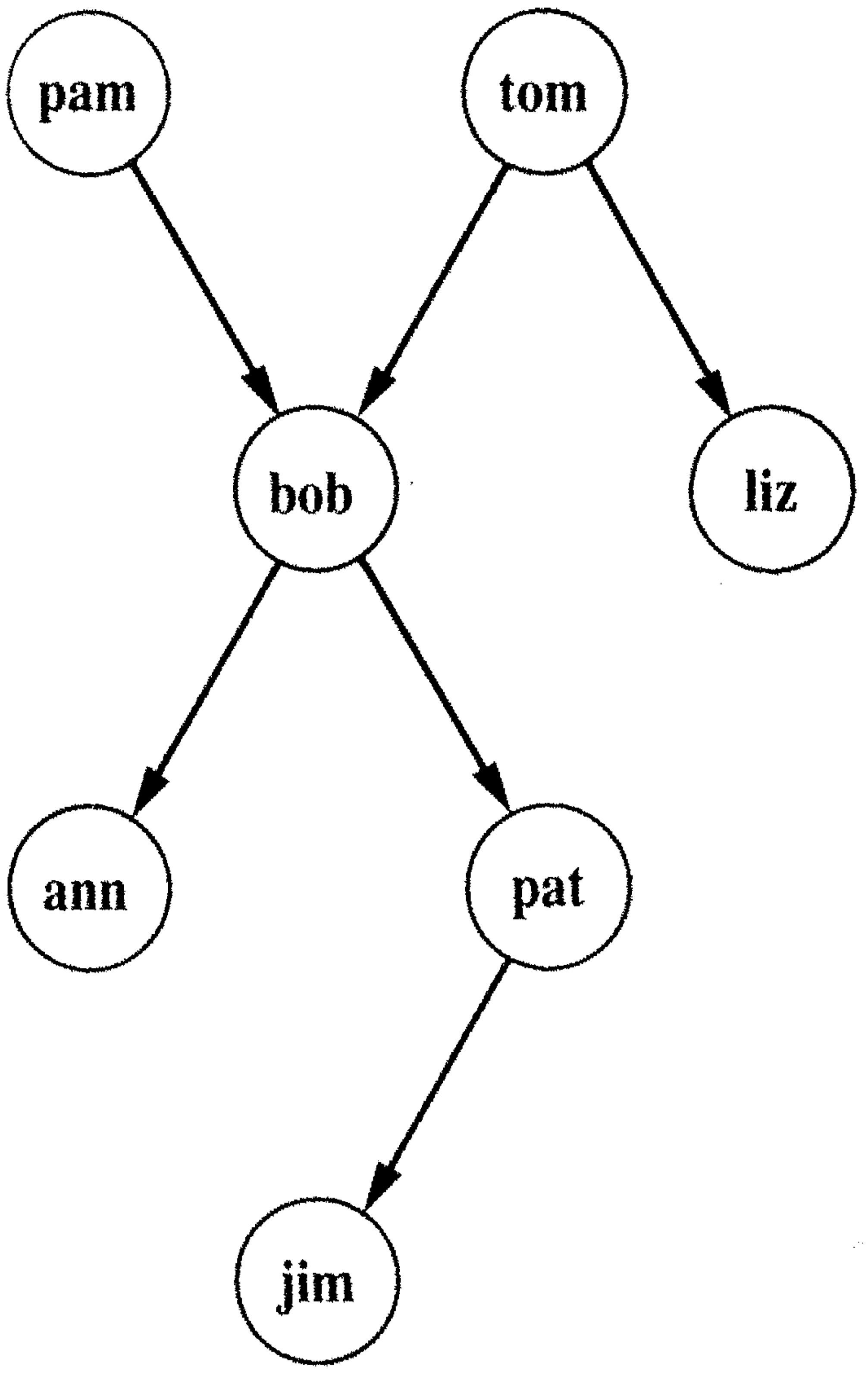


Figure 1.1 A family tree.

Defining relations by facts

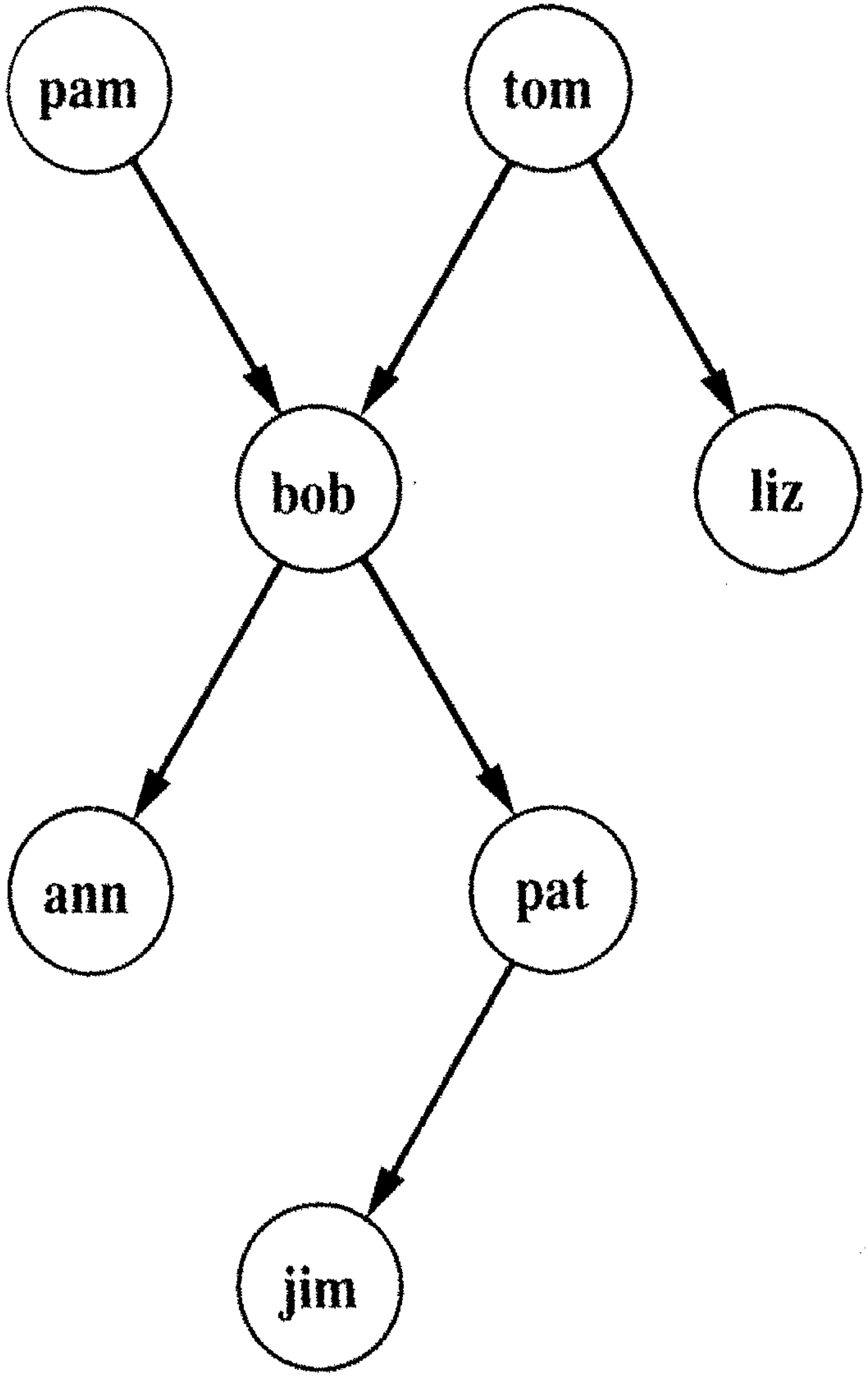


Figure 1.1 A family tree.

```
parent( pam, bob).
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
```

Demo

- `?- consult( ‘ch1.pl’).`
- `?- halt.                   % to quit`
- `;                          % next solution`
- `a                          % all solutions`
- `<ret>                      % stop`

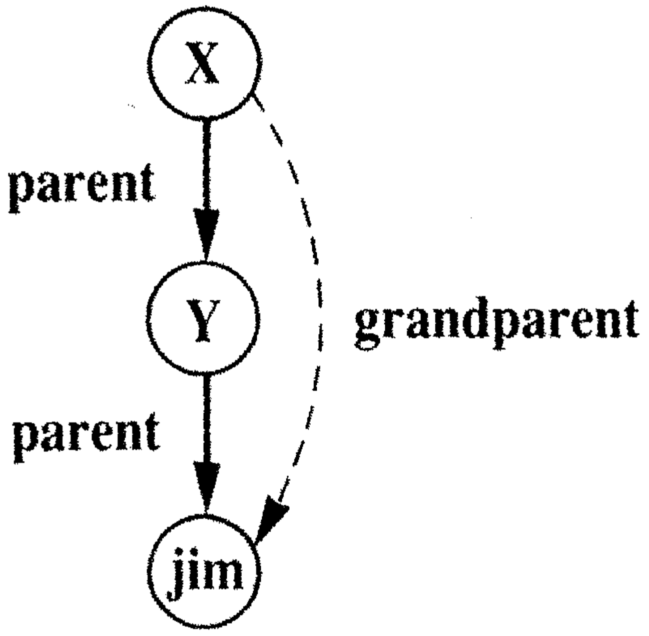


Figure 1.2 The **grandparent** relation expressed as a composition of two **parent** relations.

Who is a grandparent of jim?

1. Who is a parent of jim? `Y`
2. Who is a parent of `Y`? `X`

Query:

```
?- parent( Y, jim), parent( X, Y).
```

Who are tom’s grandchildren?

Who are tom’s grandchildren?

```
?- parent( tom, X), parent( X, Y).
```

Demo trace

```
| ?- parent( tom, X), parent( X, Y).
      1    1  Call: parent(tom,_273) ?
      1    1  Exit: parent(tom,bob) ?
```

Demo trace

```
| ?- parent( tom, X), parent( X, Y).
        1      1  Call: parent(tom,_273) ?
        1      1  Exit: parent(tom,bob) ?
```

Exits one goal, and calls the next goal.
Exit means "success".

Demo trace

```
| ?- parent( tom, X), parent( X, Y).
        1      1  Call: parent(tom,_273) ?
        1      1  Exit: parent(tom,bob) ?
```

The invocation number.

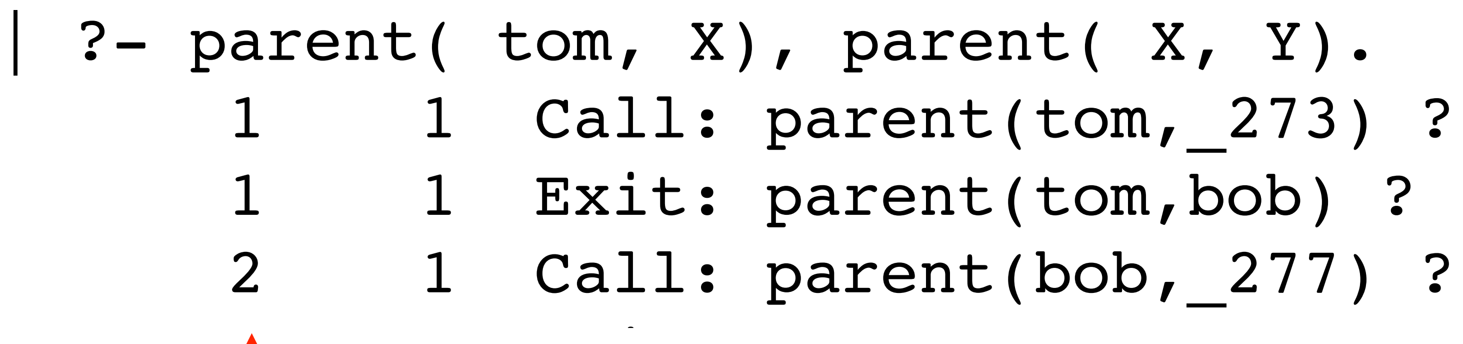
Unique for every invocation.

Demo trace

```
| ?- parent( tom, X), parent( X, Y).
        1     1  Call: parent(tom,_273) ?
        1     1  Exit: parent(tom,bob) ?
```

The index number.
The number of direct ancestors of the goal,
i.e., the current depth of the goal.

Demo trace

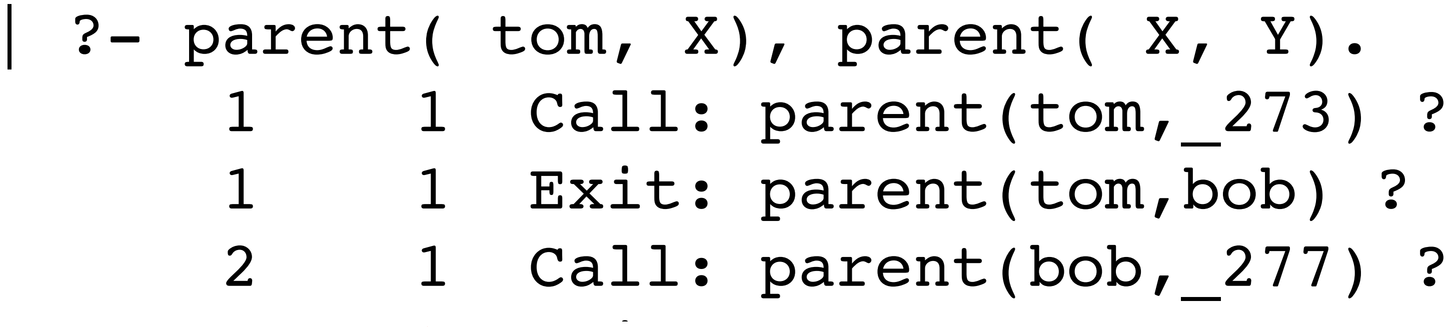


```
| ?- parent( tom, X), parent( X, Y).
        1      1  Call: parent(tom,_273) ?
        1      1  Exit: parent(tom,bob) ?
        2      1  Call: parent(bob,_277) ?
```

The invocation number increases.

Now working off of invocation 1.

Demo trace



```
| ?- parent( tom, X), parent( X, Y).
      1      1  Call: parent(tom,_273) ?
      1      1  Exit: parent(tom,bob) ?
      2      1  Call: parent(bob,_277) ?
```

The index number remains `1`.

No direct ancestors of the goal,

i.e., the current depth of the goal is `1`.

Demo trace

```
| ?- parent( tom, X), parent( X, Y).
        1    1  Call: parent(tom,_273) ?
        1    1  Exit: parent(tom,bob) ?
        2    1  Call: parent(bob,_277) ?
        2    1  Exit: parent(bob,ann) ?

X = bob
Y = ann ? ;
```

Demo trace

```
| ?- parent( tom, X), parent( X, Y).
        1     1  Call: parent(tom,_273) ?
        1     1  Exit: parent(tom,bob) ?
        2     1  Call: parent(bob,_277) ?
        2     1  Exit: parent(bob,ann) ?

X = bob
Y = ann ? ;
        2     1  Redo: parent(bob,ann) ?
```

Redo indicates backtracking.

Demo trace

```
| ?- parent( tom, X), parent( X, Y).
      1    1  Call: parent(tom,_273) ?
      1    1  Exit: parent(tom,bob) ?
      2    1  Call: parent(bob,_277) ?
      2    1  Exit: parent(bob,ann) ?

X = bob
Y = ann ? ;
      2    1  Redo: parent(bob,ann) ?
      2    1  Exit: parent(bob,pat) ?

X = bob
Y = pat ? ;
      1    1  Redo: parent(tom,bob) ?
      1    1  Exit: parent(tom,liz) ?
      2    1  Call: parent(liz,_277) ?
      2    1  Fail: parent(liz,_277) ?

(1 ms) no
```

Do ann and pat have a common parent?

Do ann and pat have a common parent?

```
?- parent( X, ann), parent( X, pat).
```

Bratko vs. `gprolog`

In `gprolog`, identical functors must be contiguous.

Bratko

```
female( pam).
male( tom).
male( bob).
female( liz).
female( ann).
female( pat).
male( jim).
```

`gprolog`

```
female( pam).
female( liz).
female( ann).
female( pat).
male( tom).
male( bob).
male( jim).
```

Defining relations by rules

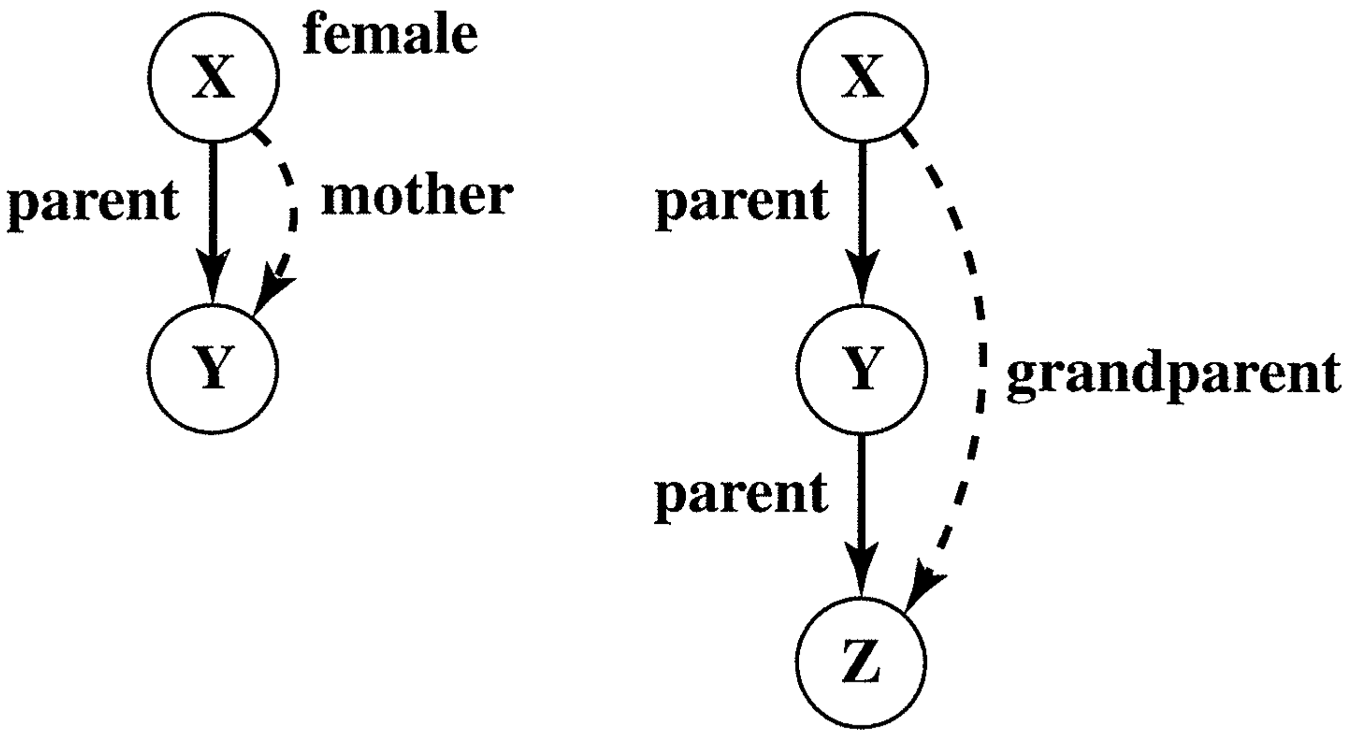


Figure 1.3 Definition graphs for the relations **mother** and **grandparent** in terms of relations **parent** and female.

Defining relations by rules

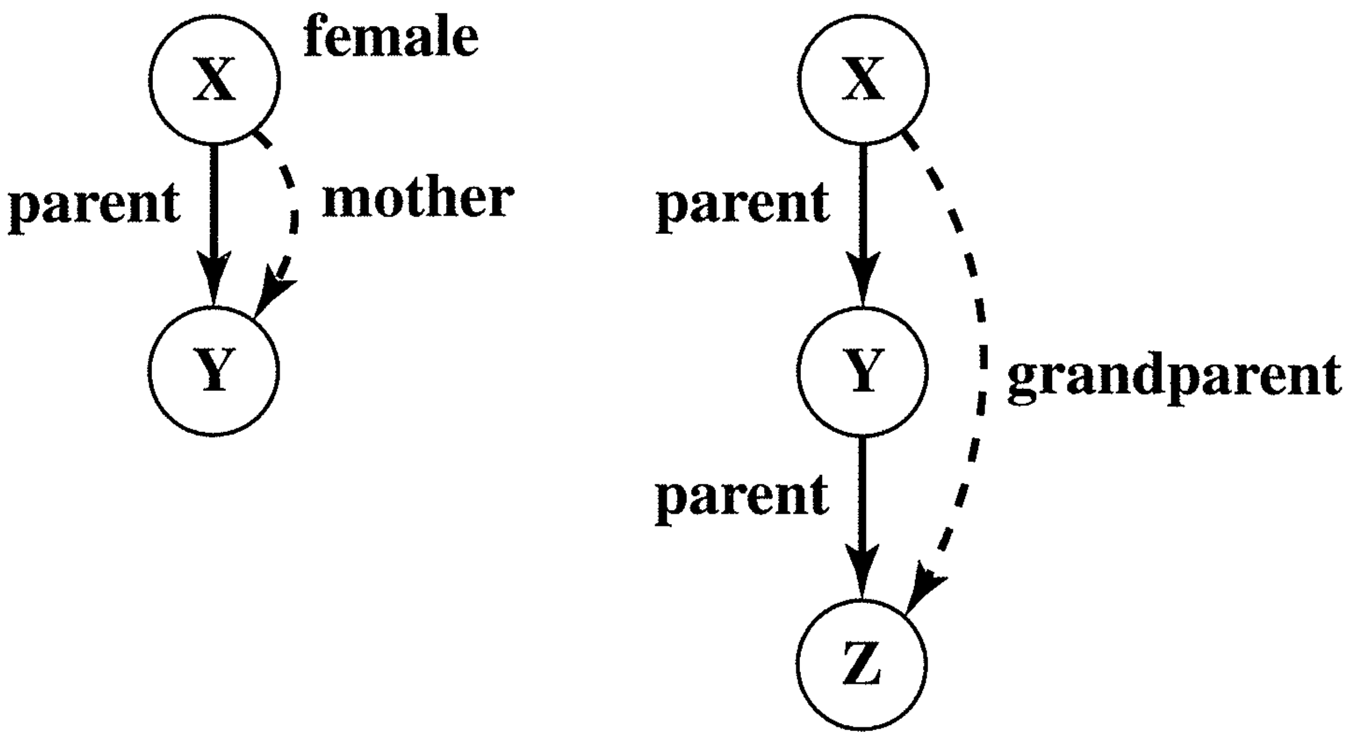


Figure 1.3 Definition graphs for the relations **mother** and **grandparent** in terms of relations **parent** and female.

```
mother( X, Y) :-      % X is the mother of Y if
  parent( X, Y),      % X is a parent of Y and
  female( X).         % X is female
```

Defining relations by rules

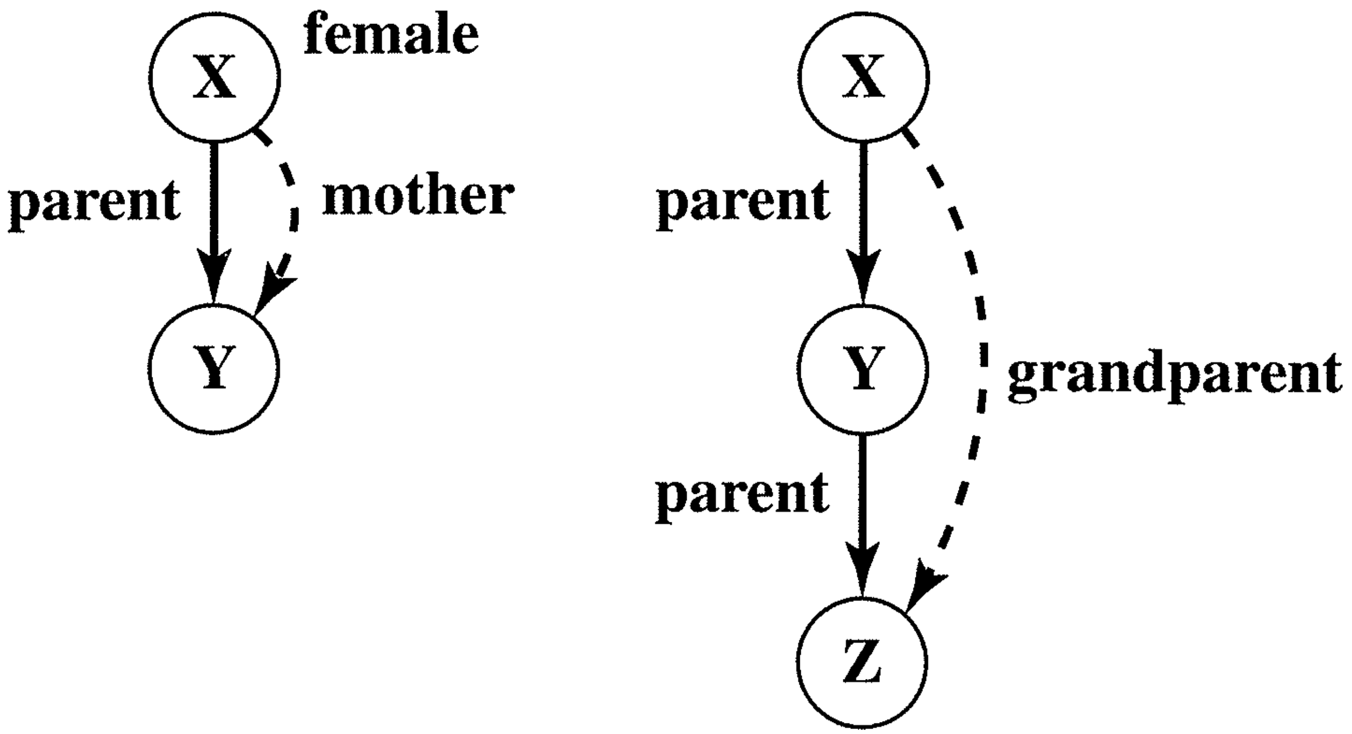
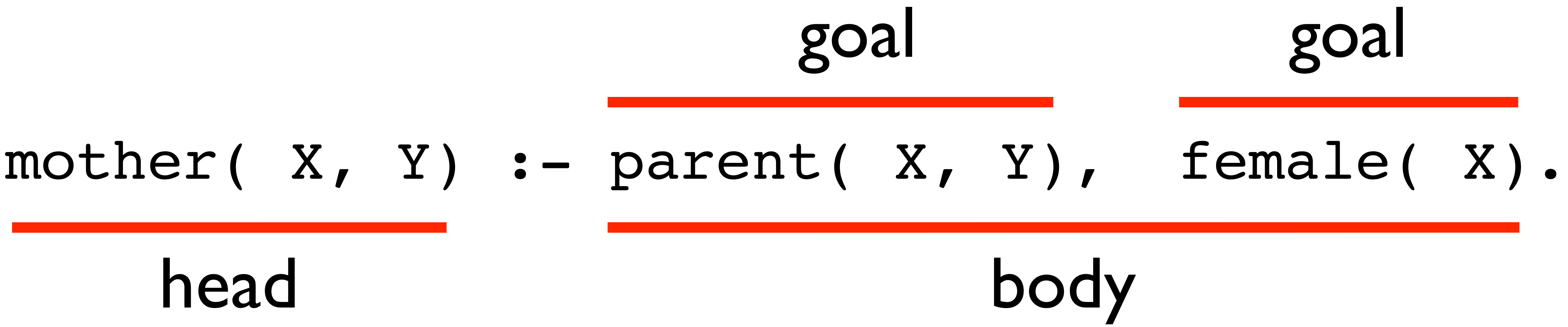


Figure 1.3 Definition graphs for the relations **mother** and **grandparent** in terms of relations **parent** and female.

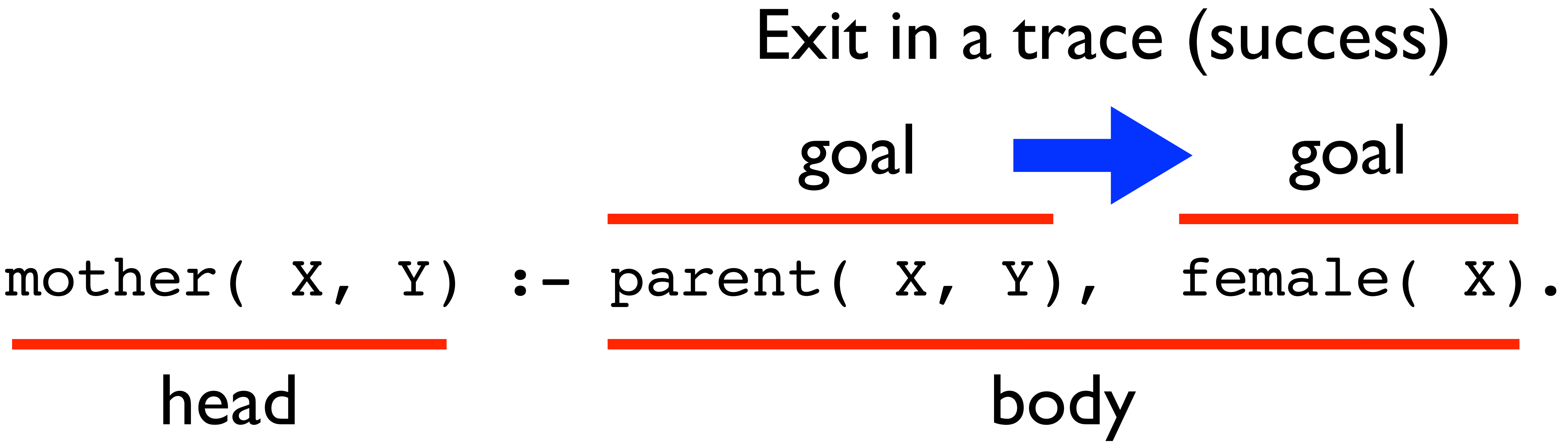
```
grandparent( X, Z) :-    % X is a grandparent of Z if
  parent( X, Y),         % X is a parent of Y and
  parent( Y, Z).         % Y is a parent of Z
```

A Prolog clause



```
mother( X, Y) :- parent( X, Y),  female( X).
```

A Prolog clause



Exit in a trace (success)

goal → goal

```
mother( X, Y) :- parent( X, Y),  female( X).
```

head

body

Recursive rules

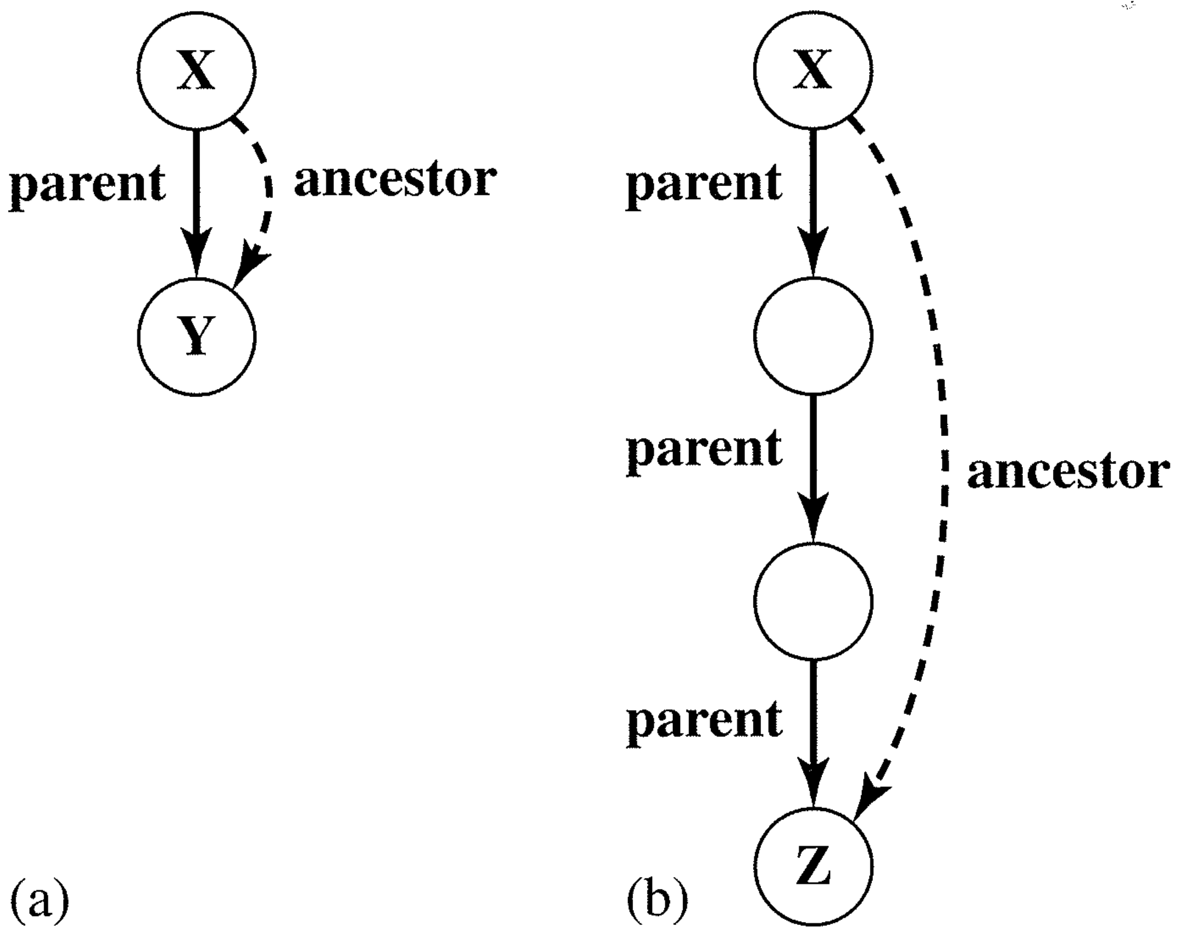


Figure 1.5 Examples of the **ancestor** relation: (a) X is a direct ancestor of Z; (b) X is an indirect ancestor of Z.

Recursive rules

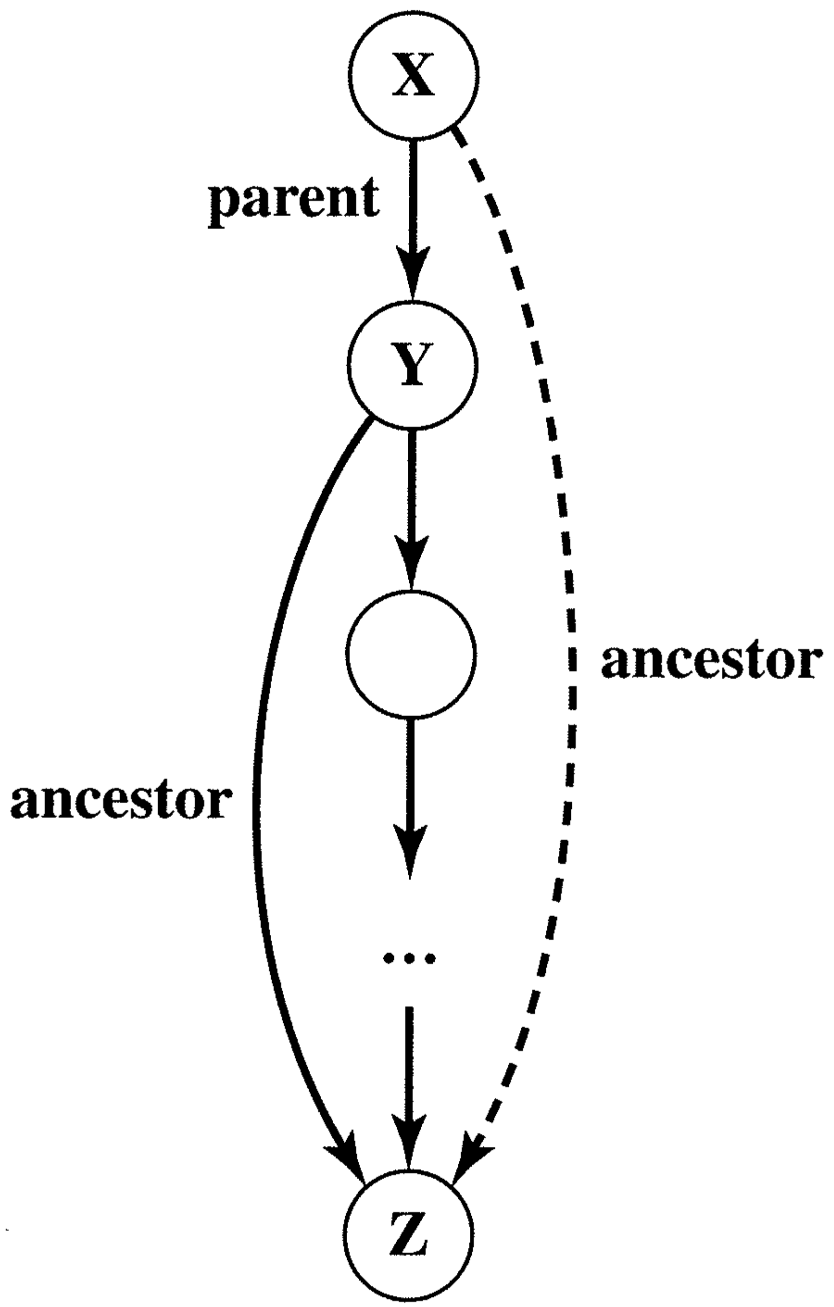


Figure 1.7 Recursive formulation of the **ancestor** relation.

Recursive rules

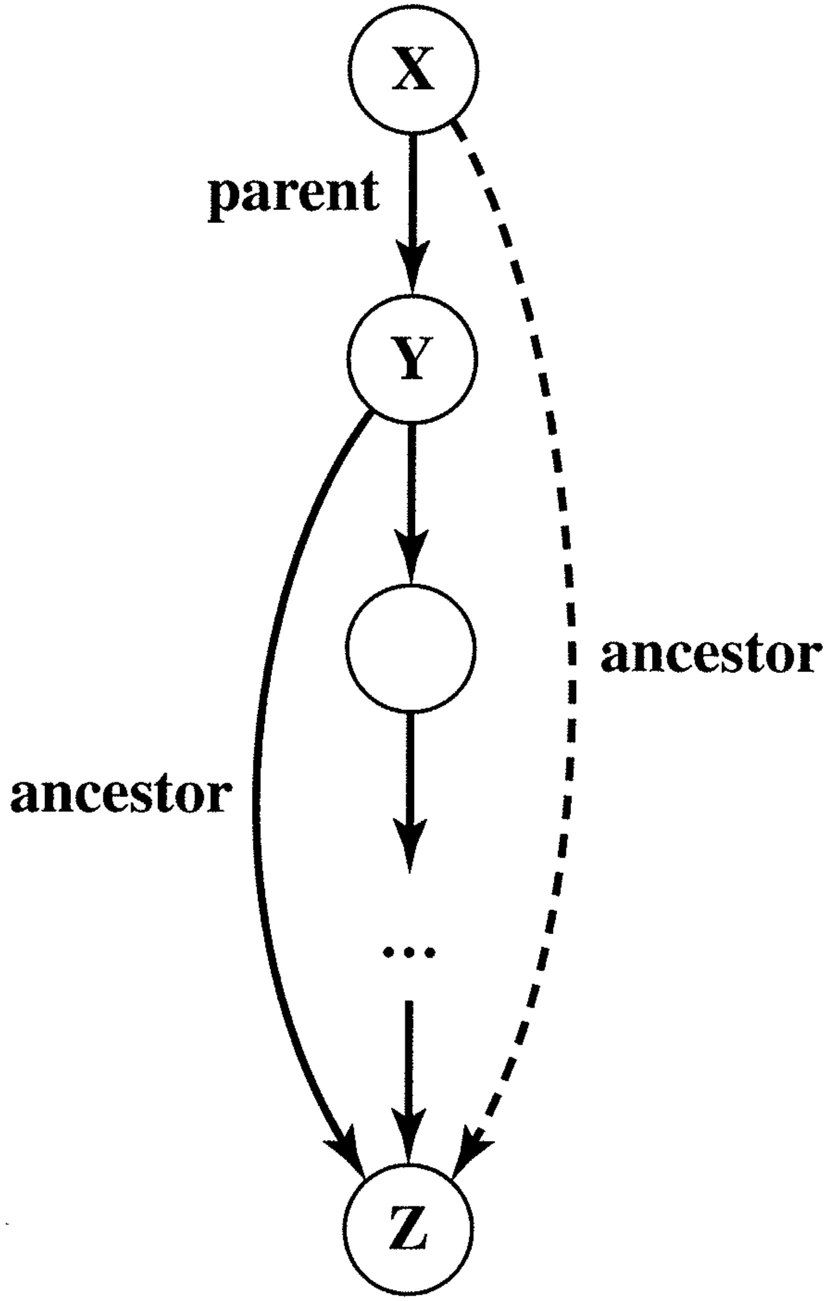
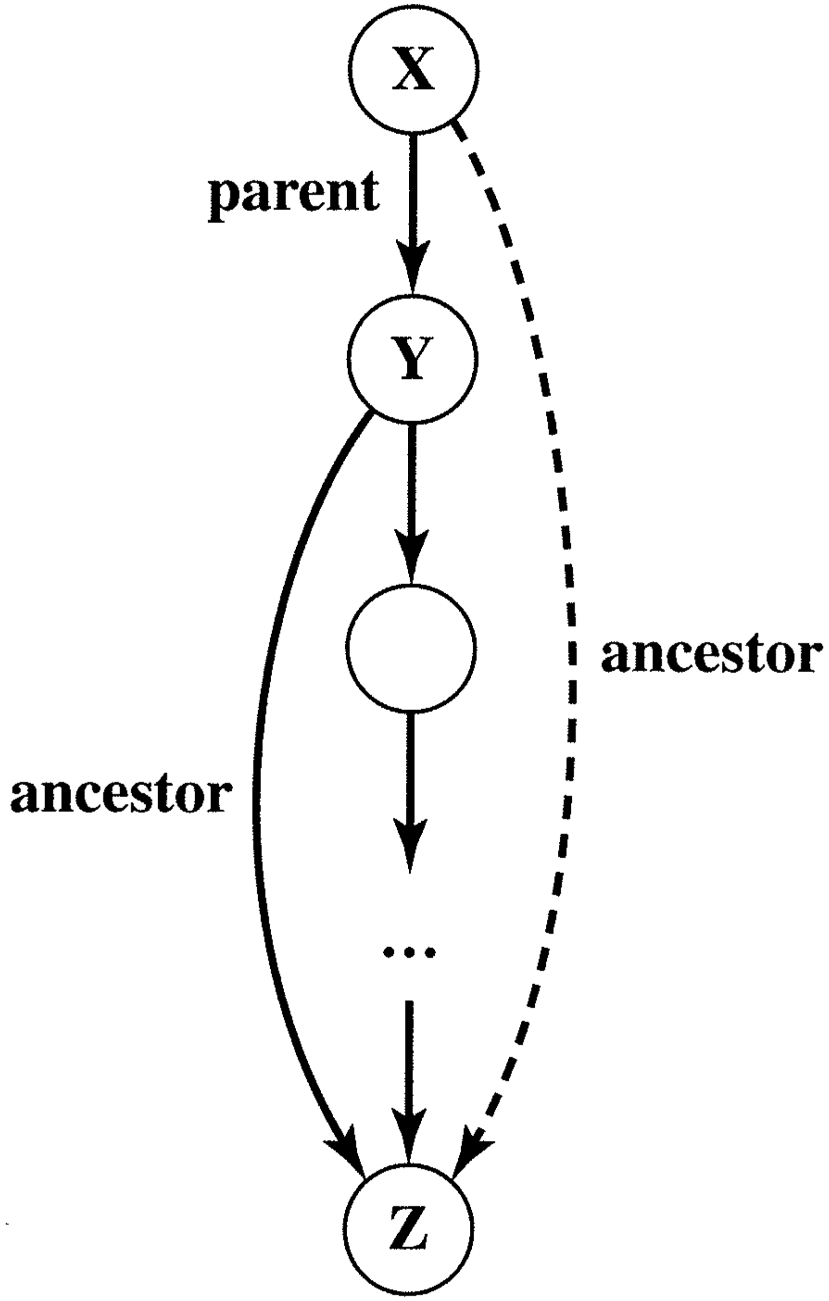


Figure 1.7 Recursive formulation of the **ancestor** relation.

```
ancestor( X, Z) :-   %rule a1
  parent( X, Z).
```

Recursive rules



```
ancestor( X, Z) :-    %rule a1
  parent( X, Z).


ancestor( X, Z) :-    %rule a2
  parent( X, Y),
  ancestor( Y, Z).
```

Figure 1.7 Recursive formulation of the **ancestor** relation.

```
ancestor( X, Z) :- parent( X, Z).
(1,1)
ancestor( X, Z) :- parent( X, Y), ancestor( Y, Z).
```

```
| ?- ancestor( tom, pat).
     1     1  Call: ancestor(tom,pat) ?
```

```
parent( pam, bob).
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
```

```
ancestor( X, Z) :- parent( X, Z).
(1,1)              (2,2)
ancestor( X, Z) :- parent( X, Y), ancestor( Y, Z).
```

```
| ?- ancestor( tom, pat).
        1      1  Call: ancestor(tom,pat) ?
        2      2  Call: parent(tom,pat) ?
```

```
parent( pam, bob).
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
```

```
ancestor( X, Z) :- parent( X, Z).
(1,1)              (2,2)
ancestor( X, Z) :- parent( X, Y), ancestor( Y, Z).
```

```
| ?- ancestor( tom, pat).
    1     1  Call: ancestor(tom,pat) ?
    2     2  Call: parent(tom,pat) ?
    2     2  Fail: parent(tom,pat) ?
```

```
parent( pam, bob).
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
```

```
ancestor( X, Z) :- parent( X, Z).

ancestor( X, Z) :- parent( X, Y), ancestor( Y, Z).
(1,1)              (2,2)
```

```
| ?- ancestor( tom, pat).
        1      1  Call: ancestor(tom,pat) ?
        2      2  Call: parent(tom,pat) ?
        2      2  Fail: parent(tom,pat) ?
        2      2  Call: parent(tom,_336) ?
```

```
parent( pam, bob).
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
```

```
ancestor( X, Z) :- parent( X, Z).

ancestor( X, Z) :- parent( X, Y), ancestor( Y, Z).
(1,1)                (2,2)
```

```
| ?- ancestor( tom, pat).
     1     1  Call: ancestor(tom,pat) ?
     2     2  Call: parent(tom,pat) ?
     2     2  Fail: parent(tom,pat) ?
     2     2  Call: parent(tom,_336) ?
     2     2  Exit: parent(tom,bob) ?
```

```
parent( pam, bob).
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
```

```
ancestor( X, Z) :- parent( X, Z).

ancestor( X, Z) :- parent( X, Y), ancestor( Y, Z).
(1,1)              (2,2)          (3,2)
```

```
| ?- ancestor( tom, pat).
      1     1  Call: ancestor(tom,pat) ?
      2     2  Call: parent(tom,pat) ?
      2     2  Fail: parent(tom,pat) ?
      2     2  Call: parent(tom,_336) ?
      2     2  Exit: parent(tom,bob) ?
      3     2  Call: ancestor(bob,pat) ?
```

```
parent( pam, bob).
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
```

```
ancestor( X, Z) :- parent( X, Z).
                   (4,3)
ancestor( X, Z) :- parent( X, Y), ancestor( Y, Z).
(1,1)              (2,2)          (3,2)
```

```
| ?- ancestor( tom, pat).
     1     1  Call: ancestor(tom,pat) ?
     2     2  Call: parent(tom,pat) ?
     2     2  Fail: parent(tom,pat) ?
     2     2  Call: parent(tom,_336) ?
     2     2  Exit: parent(tom,bob) ?
     3     2  Call: ancestor(bob,pat) ?
     4     3  Call: parent(bob,pat) ?
```

```
parent( pam, bob).
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
```

```
ancestor( X, Z) :- parent( X, Z).
                   (4,3)
ancestor( X, Z) :- parent( X, Y), ancestor( Y, Z).
(1,1)              (2,2)          (3,2)
```

```
| ?- ancestor( tom, pat).
     1     1  Call: ancestor(tom,pat) ?
     2     2  Call: parent(tom,pat) ?
     2     2  Fail: parent(tom,pat) ?
     2     2  Call: parent(tom,_336) ?
     2     2  Exit: parent(tom,bob) ?
     3     2  Call: ancestor(bob,pat) ?
     4     3  Call: parent(bob,pat) ?
     4     3  Exit: parent(bob,pat) ?
```

```
parent( pam, bob).
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
```

```
ancestor( X, Z) :- parent( X, Z).

ancestor( X, Z) :- parent( X, Y), ancestor( Y, Z).
(1,1)                (2,2)            (3,2)
```

```
| ?- ancestor( tom, pat).
      1     1  Call: ancestor(tom,pat) ?
      2     2  Call: parent(tom,pat) ?
      2     2  Fail: parent(tom,pat) ?
      2     2  Call: parent(tom,_336) ?
      2     2  Exit: parent(tom,bob) ?
      3     2  Call: ancestor(bob,pat) ?
      4     3  Call: parent(bob,pat) ?
      4     3  Exit: parent(bob,pat) ?
      3     2  Exit: ancestor(bob,pat) ?
```

```
parent( pam, bob).
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
```

```
ancestor( X, Z) :- parent( X, Z).

ancestor( X, Z) :- parent( X, Y), ancestor( Y, Z).
(1,1)
```

```
| ?- ancestor( tom, pat).
      1      1  Call: ancestor(tom,pat) ?
      2      2  Call: parent(tom,pat) ?
      2      2  Fail: parent(tom,pat) ?
      2      2  Call: parent(tom,_336) ?
      2      2  Exit: parent(tom,bob) ?
      3      2  Call: ancestor(bob,pat) ?
      4      3  Call: parent(bob,pat) ?
      4      3  Exit: parent(bob,pat) ?
      3      2  Exit: ancestor(bob,pat) ?
      1      1  Exit: ancestor(tom,pat) ?
```

```
parent( pam, bob).
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
```

```
ancestor( X, Z) :- parent( X, Z).

ancestor( X, Z) :- parent( X, Y), ancestor( Y, Z).
```

```
| ?- ancestor( tom, pat).
      1      1  Call: ancestor(tom,pat) ?
      2      2  Call: parent(tom,pat) ?
      2      2  Fail: parent(tom,pat) ?
      2      2  Call: parent(tom,_336) ?
      2      2  Exit: parent(tom,bob) ?
      3      2  Call: ancestor(bob,pat) ?
      4      3  Call: parent(bob,pat) ?
      4      3  Exit: parent(bob,pat) ?
      3      2  Exit: ancestor(bob,pat) ?
      1      1  Exit: ancestor(tom,pat) ?

true ? ;
```

```
parent( pam, bob).
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
```

```
ancestor( X, Z) :- parent( X, Z).

ancestor( X, Z) :- parent( X, Y), ancestor( Y, Z).
(1,1)
```

```
| ?- ancestor( tom, pat).
      1     1  Call: ancestor(tom,pat) ?
      2     2  Call: parent(tom,pat) ?
      2     2  Fail: parent(tom,pat) ?
      2     2  Call: parent(tom,_336) ?
      2     2  Exit: parent(tom,bob) ?
      3     2  Call: ancestor(bob,pat) ?
      4     3  Call: parent(bob,pat) ?
      4     3  Exit: parent(bob,pat) ?
      3     2  Exit: ancestor(bob,pat) ?
      1     1  Exit: ancestor(tom,pat) ?

true ? ;
      1     1  Redo: ancestor(tom,pat) ?
```

```
parent( pam, bob).
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
```

```
ancestor( X, Z) :- parent( X, Z).

ancestor( X, Z) :- parent( X, Y), ancestor( Y, Z).
(1,1)              (2,2)          (3,2)
```

```
| ?- ancestor( tom, pat).
      1    1  Call: ancestor(tom,pat) ?
      2    2  Call: parent(tom,pat) ?
      2    2  Fail: parent(tom,pat) ?
      2    2  Call: parent(tom,_336) ?
      2    2  Exit: parent(tom,bob) ?
      3    2  Call: ancestor(bob,pat) ?
      4    3  Call: parent(bob,pat) ?
      4    3  Exit: parent(bob,pat) ?
      3    2  Exit: ancestor(bob,pat) ?
      1    1  Exit: ancestor(tom,pat) ?

true ? ;
      1    1  Redo: ancestor(tom,pat) ?
      3    2  Redo: ancestor(bob,pat) ?
```

```
parent( pam, bob).
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
```

```
    ancestor( X, Z) :- parent( X, Z).
                       (4,3)
    ancestor( X, Z) :- parent( X, Y), ancestor( Y, Z).
    (1,1)              (2,2)          (3,2)
```

```
| ?- ancestor( tom, pat).
     1     1  Call: ancestor(tom,pat) ?
     2     2  Call: parent(tom,pat) ?
     2     2  Fail: parent(tom,pat) ?
     2     2  Call: parent(tom,_336) ?
     2     2  Exit: parent(tom,bob) ?
     3     2  Call: ancestor(bob,pat) ?
     4     3  Call: parent(bob,pat) ?
     4     3  Exit: parent(bob,pat) ?
     3     2  Exit: ancestor(bob,pat) ?
     1     1  Exit: ancestor(tom,pat) ?

true ? ;
     1     1  Redo: ancestor(tom,pat) ?
     3     2  Redo: ancestor(bob,pat) ?
     4     3  Call: parent(bob,_385) ?
```

```
parent( pam, bob).
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
```

```
ancestor( X, Z) :- parent( X, Z).
                   (4,3)
ancestor( X, Z) :- parent( X, Y), ancestor( Y, Z).
(1,1)              (2,2)          (3,2)
```

```
| ?- ancestor( tom, pat).
      1      1  Call: ancestor(tom,pat) ?
      2      2  Call: parent(tom,pat) ?
      2      2  Fail: parent(tom,pat) ?
      2      2  Call: parent(tom,_336) ?
      2      2  Exit: parent(tom,bob) ?
      3      2  Call: ancestor(bob,pat) ?
      4      3  Call: parent(bob,pat) ?
      4      3  Exit: parent(bob,pat) ?
      3      2  Exit: ancestor(bob,pat) ?
      1      1  Exit: ancestor(tom,pat) ?

true ? ;
      1      1  Redo: ancestor(tom,pat) ?
      3      2  Redo: ancestor(bob,pat) ?
      4      3  Call: parent(bob,_385) ?
```

etc. ... eventually fails

```
parent( pam, bob).
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
```

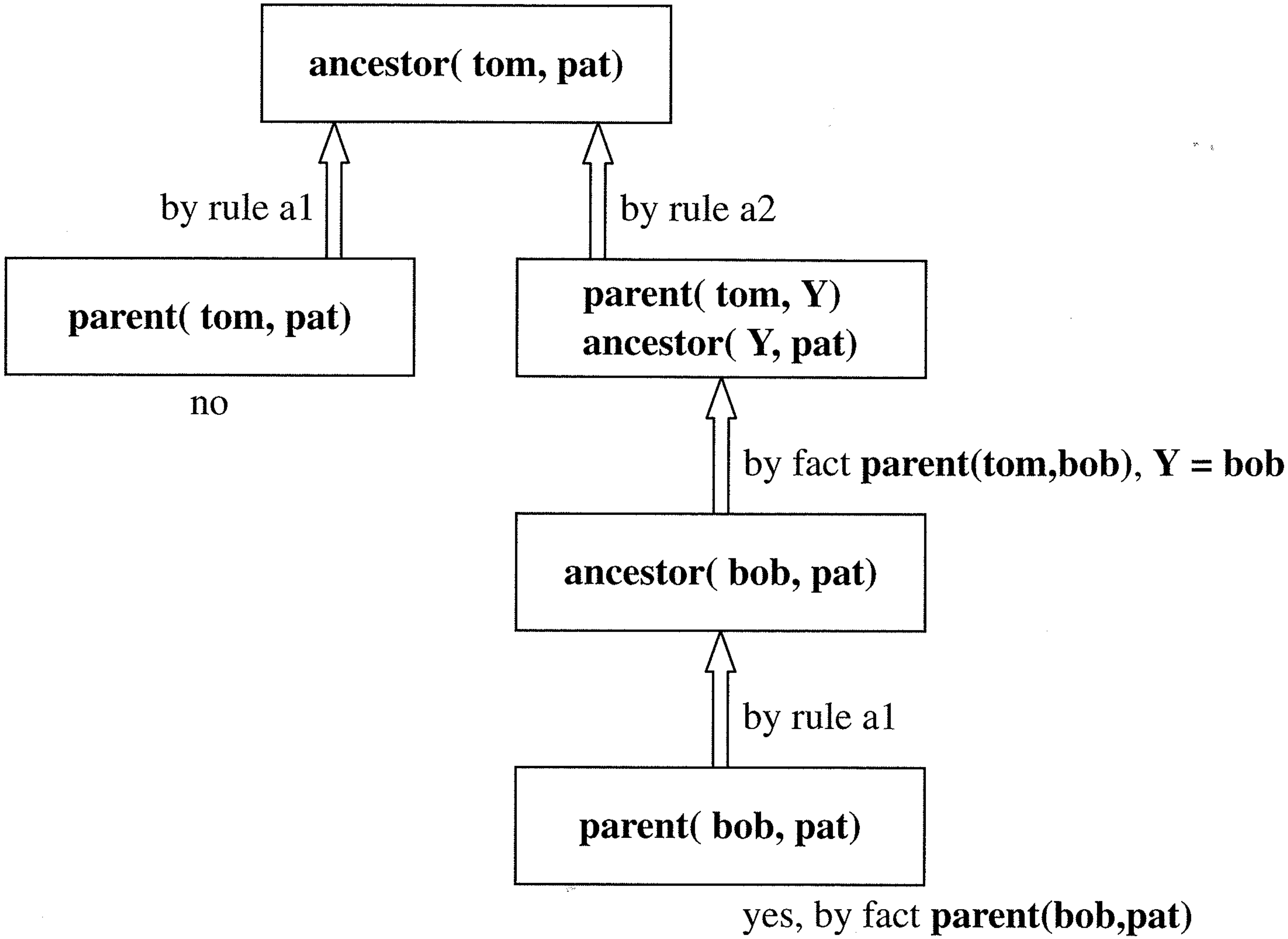


Figure 1.9 The complete execution trace to satisfy the goal **ancestor(tom, pat)**. The left-hand branch fails, but the right-hand branch proves the goal is satisfiable.