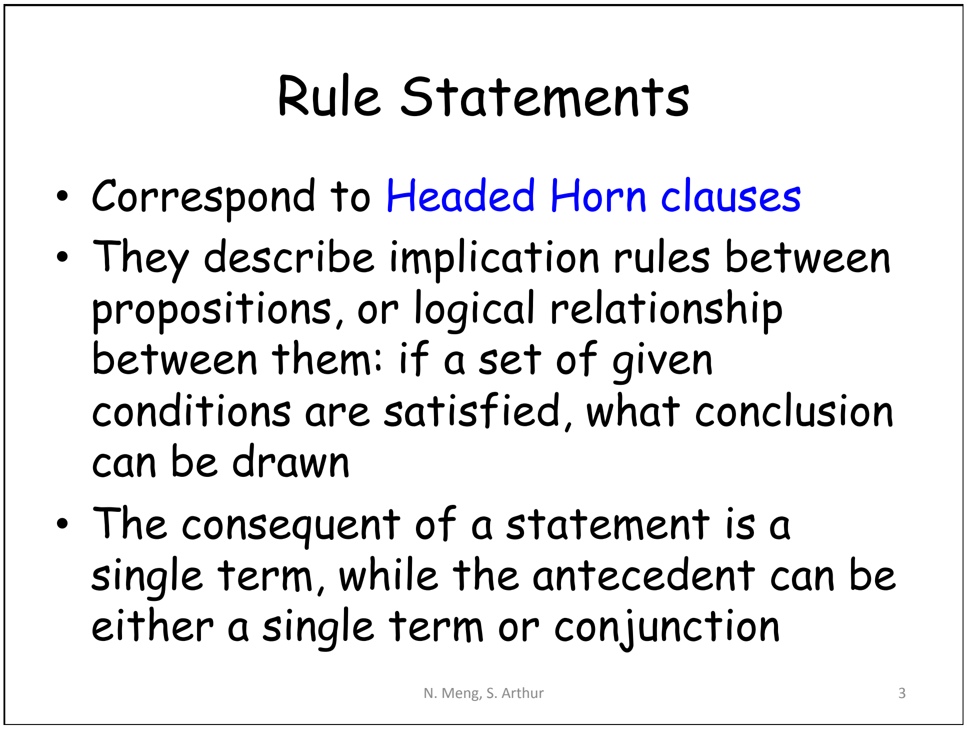
Prolog

- A logic programming language
- Prolog programs consist of collections of statements
- There are only a few kinds of statements in Prolog, but they can be complex
  - Fact statements, rule statements, and goal statements
- All prolog statements are constructed from terms

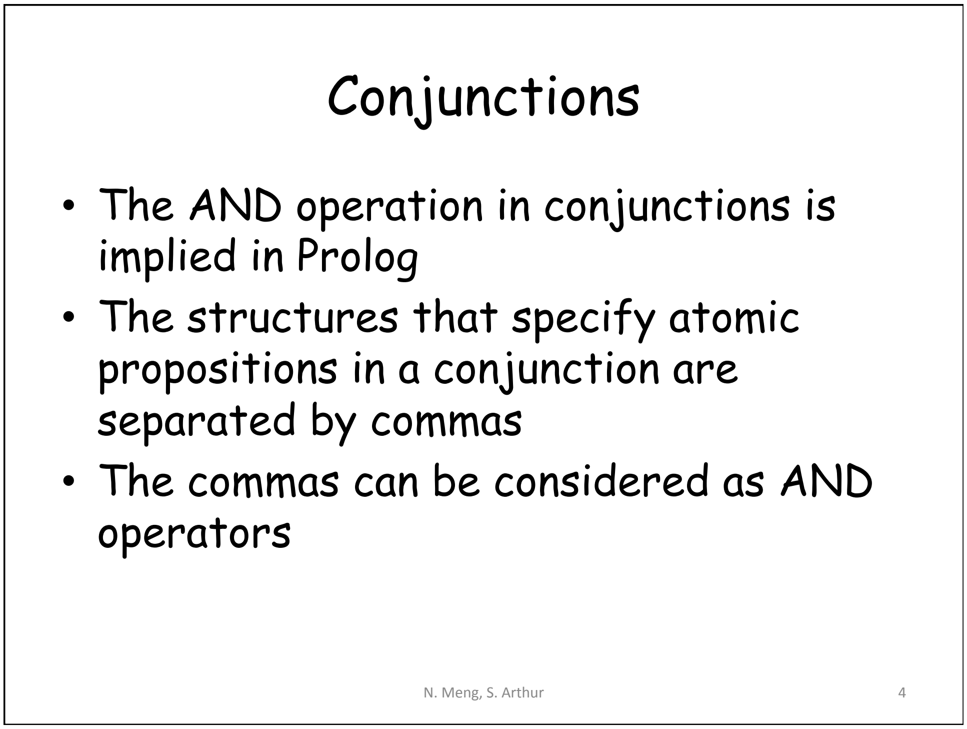
Fact Statements

- Correspond to Headless Horn clauses
- Fact statements are propositions that are assumed to be true, and from which new information can be inferred
- E.g., female(shelley).
  female(mary).
  mother(mary, shelley).



Rule Statements

- Correspond to Headed Horn clauses
- They describe implication rules between propositions, or logical relationship between them: if a set of given conditions are satisfied, what conclusion can be drawn
- The consequent of a statement is a single term, while the antecedent can be either a single term or conjunction



Conjunctions

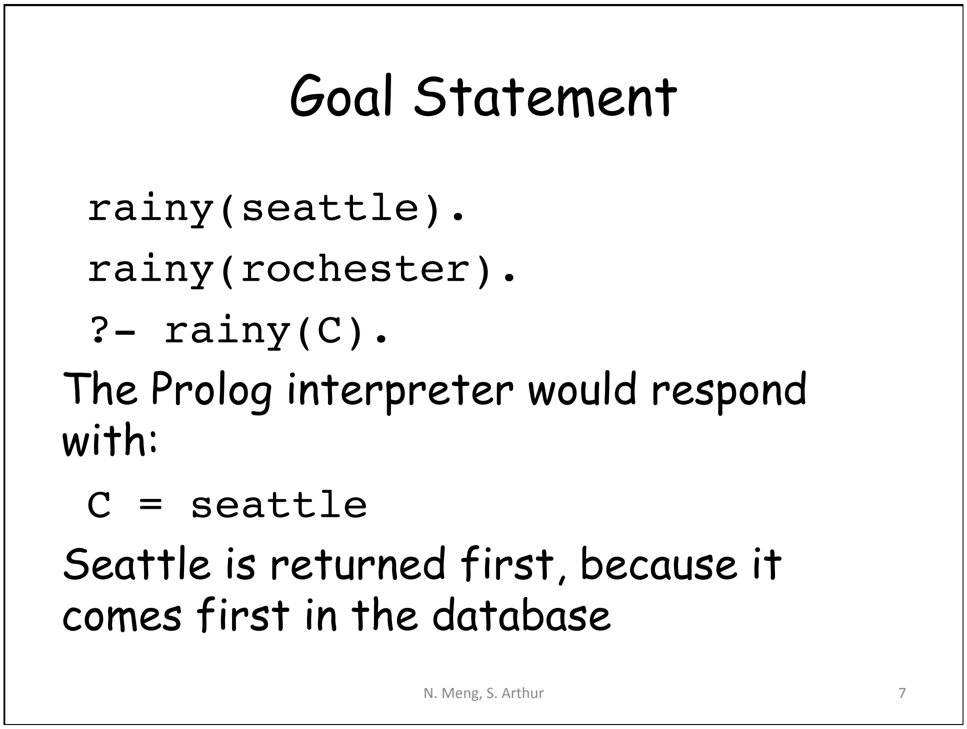
- The AND operation in conjunctions is implied in Prolog
- The structures that specify atomic propositions in a conjunction are separated by commas
- The commas can be considered as AND operators

Rule Statements

- E.g., grandparent(X, Z) :- parent(X, Y), parent(Y, Z),
  where X, Y, and Z are universal objects
  - It states that if there are instantiations of X, Y, and Z such that parent (X, Y) is true, and parent (Y, Z) is true, then for those same instantiations of X, Y, and Z, grandparent(X, Z) is true

Goal Statements

- Also correspond to Headless Horn clauses
- **Goal statements** are propositions describing the theorem that we want the system to either prove or disprove
  - E.g., man(fred).
- Because goal statements and some nongoal statements have the same form, a Prolog implementation must have some means to distinguish between the two



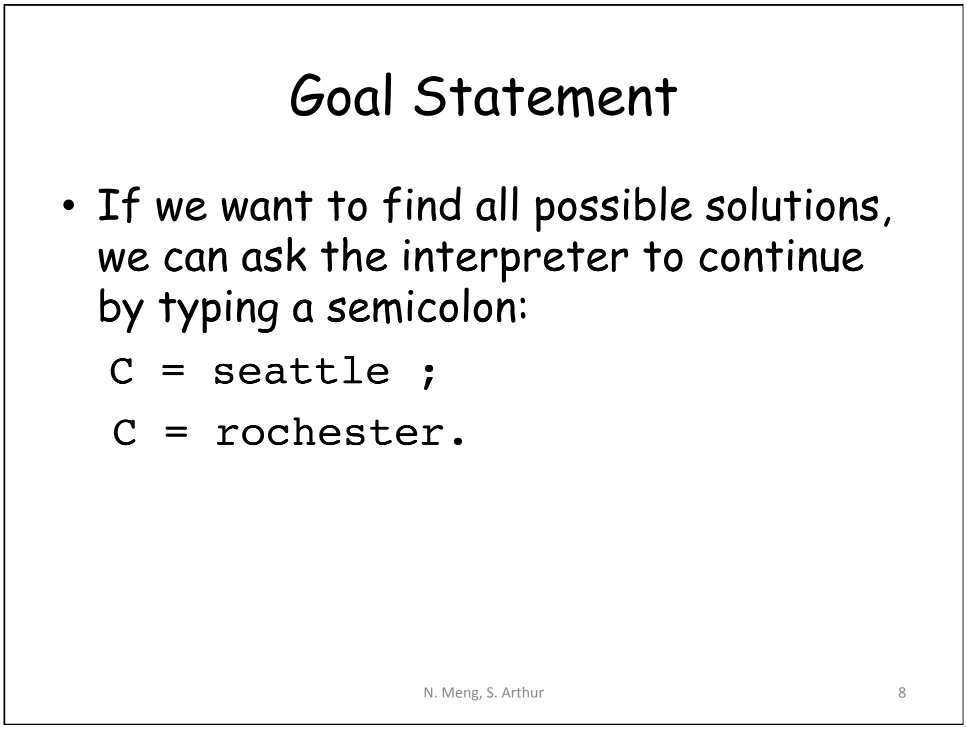
Goal Statement

```
rainy(seattle).
rainy(rochester).
?- rainy(C).
```

The Prolog interpreter would respond with:

```
C = seattle
```

Seattle is returned first, because it comes first in the database



Goal Statement

- If we want to find all possible solutions, we can ask the interpreter to continue by typing a semicolon:

```
C = seattle ;
C = rochester.
```

Another Example

```
 takes(jane_doe, his201).
 takes(jane_doe, cs254).
 takes(ajit_chandra, art302).
 takes(ajit_chandra, cs254).
 classmates(X, Y) :- takes(X, Z),
takes(Y, Z).
```

What does the following query return?

```
 ?- classmates(jane_doe, X).
```

```
 X = jane_doe ;
 X = jane_doe;
 X = ajit_chandra.
```

How should we modify the rule so that the student is not considered as a classmate of himself or herself?

```
classmates(X, Y) :- takes(X, Z),
takes(Y, Z), X\=Y.
```

- Can we define propositions in the following way?

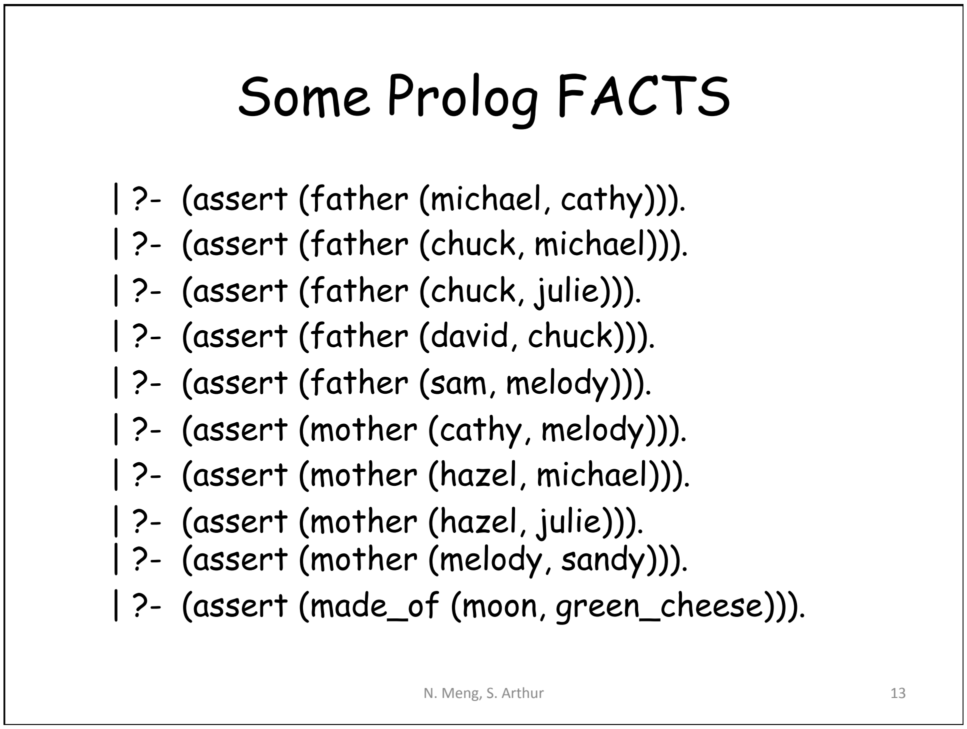
```
takes(jane doe, his201).
```

- No. The prolog interpreter will complain. Instead, we can define the proposition as below:

```
takes('jane doe', his201).
```

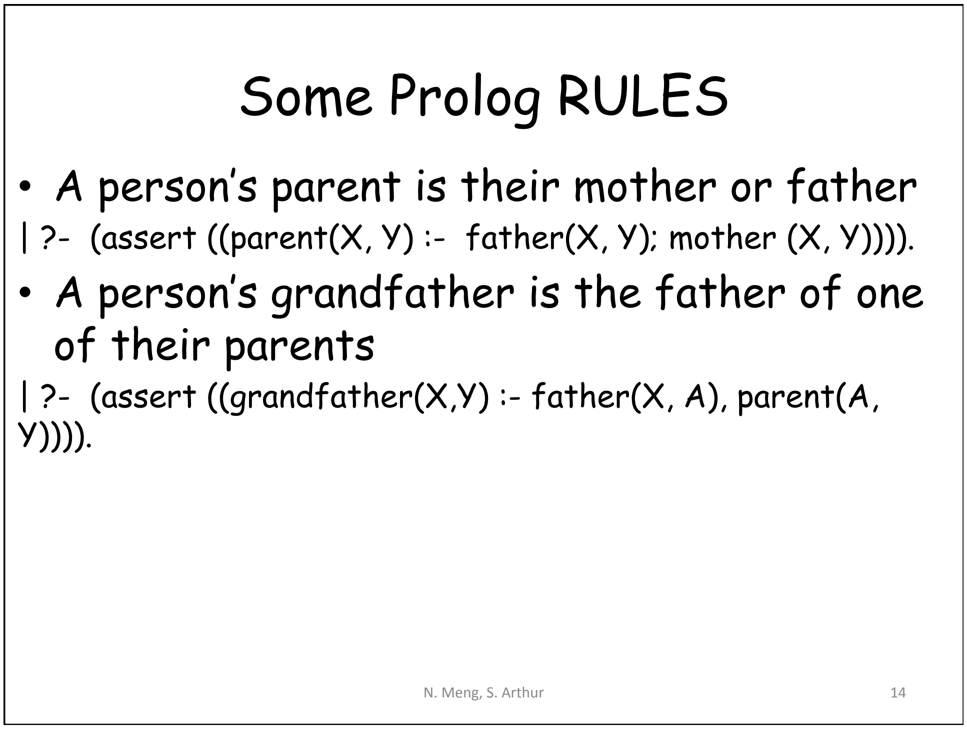
Prolog Programs

- ASSERT (define)
  - <u>FACTS</u> about <u>OBJECTS</u>
  - <u>RULES</u>("CLAUSES") that inter-relate facts
- Ask <u>QUESTION</u>S about objects and their relationship
  - <u>GOALS</u>



Some Prolog FACTS

```
| ?-  (assert (father (michael, cathy))).
| ?-  (assert (father (chuck, michael))).
| ?-  (assert (father (chuck, julie))).
| ?-  (assert (father (david, chuck))).
| ?-  (assert (father (sam, melody))).
| ?-  (assert (mother (cathy, melody))).
| ?-  (assert (mother (hazel, michael))).
| ?-  (assert (mother (hazel, julie))).
| ?-  (assert (mother (melody, sandy))).
| ?-  (assert (made_of (moon, green_cheese))).
```



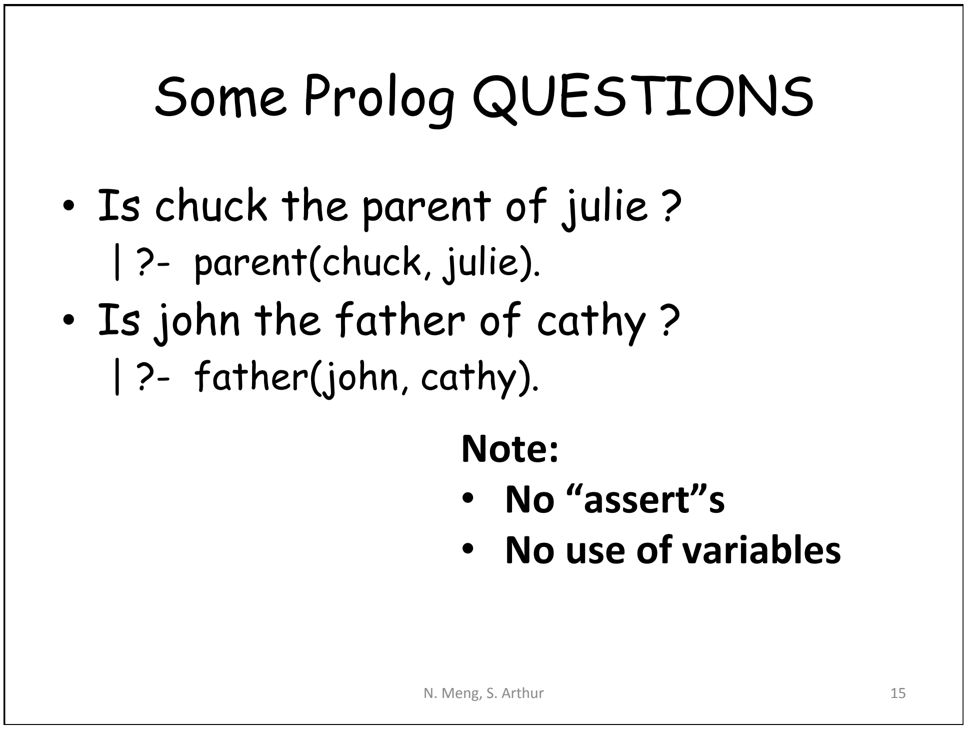
Some Prolog RULES

- A person's parent is their mother or father

```
| ?-  (assert ((parent(X, Y) :-  father(X, Y); mother (X, Y)))).
```

- A person's grandfather is the father of one of their parents

```
| ?-  (assert ((grandfather(X,Y) :- father(X, A), parent(A,
Y)))).
```



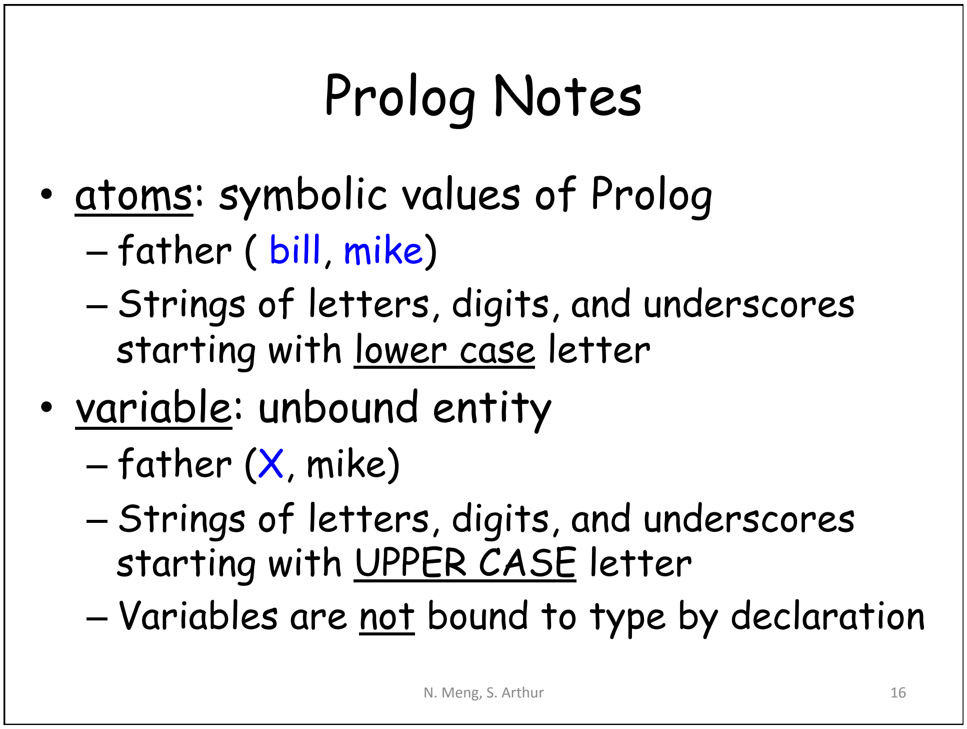
Some Prolog QUESTIONS

- Is chuck the parent of julie ?

  | ?- parent(chuck, julie).
- Is john the father of cathy ?

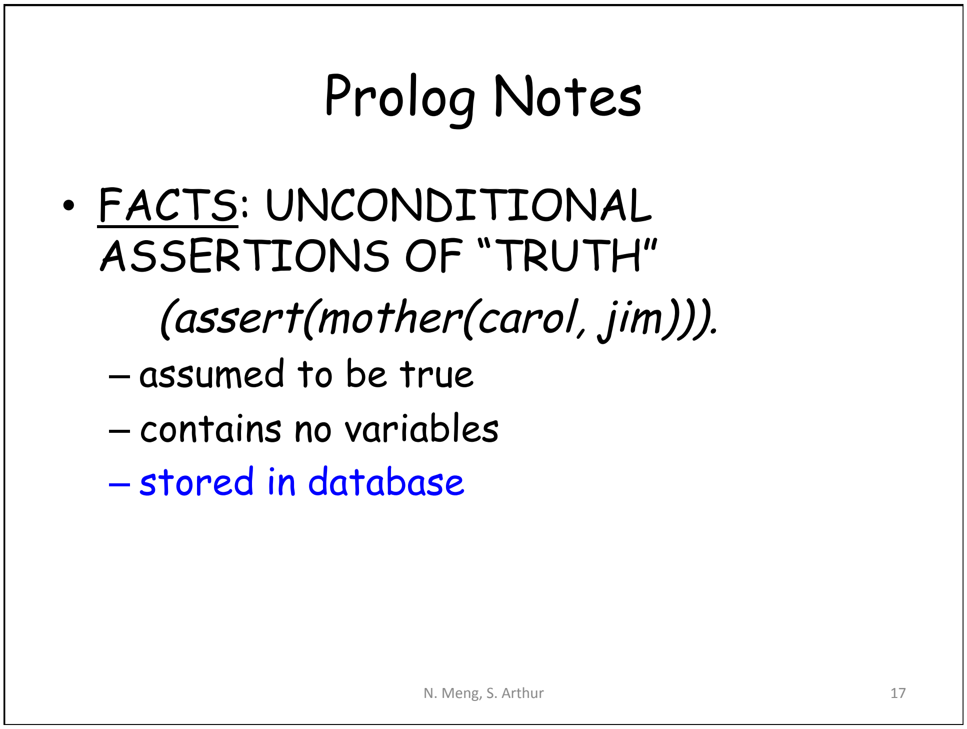
  | ?- father(john, cathy).

Note:
- **No “assert”s**
- **No use of variables**



Prolog Notes

- <u>atoms</u>: symbolic values of Prolog
  - father (bill, mike)
  - Strings of letters, digits, and underscores starting with <u>lower case</u> letter
- <u>variable</u>: unbound entity
  - father (X, mike)
  - Strings of letters, digits, and underscores starting with <u>UPPER CASE</u> letter
  - Variables are <u>not</u> bound to type by declaration

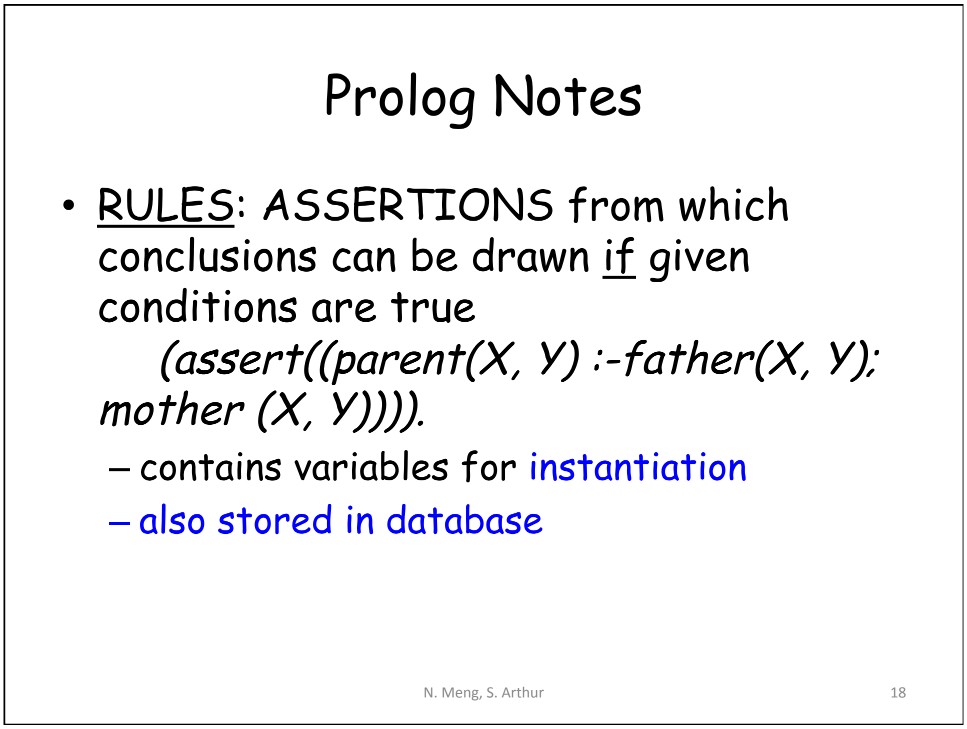


Prolog Notes

- FACTS: UNCONDITIONAL ASSERTIONS OF “TRUTH”

  (assert(mother(carol, jim))).

  - assumed to be true
  - contains no variables
  - stored in database



Prolog Notes

- RULES: ASSERTIONS from which conclusions can be drawn if given conditions are true

  (assert((parent(X, Y) :-father(X, Y); mother (X, Y)))).

  - contains variables for instantiation
  - also stored in database

An Example

FACTS

```
| ?-   (assert(color(banana, yellow))).
| ?-   (assert(color(squash, yellow))).
| ?-   (assert(color(apple, green))).
| ?-   (assert(color(peas, green))).

| ?-   (assert(fruit(banana))).
| ?-   (assert(fruit(apple))).
| ?-   (assert(vegetable(squash))).
| ?-   (assert(vegetable(peas))).
```

bob eats green colored vegetables

RULE

```
| ?-   (assert((eats(bob, X) :-  color(X,
 green), vegetable(X)))).
```

An Example

```
(assert ((eats(bob, X) :-
        color(X, green),
        vegetable(X)))).
```

Does bob eat apples ?

```
| ?- eats(bob, apple).
        color(apple, green) => match
        vegetable(apple)    => no
```

false

Does bob eat squash ?

```
| ?- eats(bob, squash).
        color(squash, green) => no
```

false

What does bob eat ?

```
| ?- eats(bob, X).
        color(banana, green) => no
        color(squash, green) => no
        color(apple, green)     => yes
            vegetable(apple)  => no
        color(peas, green)      => yes
            vegetable(peas)   => yes
```

therefore X = peas