More on Prolog

COMP360

“Ambition is the path to success, persistence is the vehicle you arrive in.”

William Eardley IV

Remaining Schedule

	Wednesday, April 12 Prolog	Friday, April 14 *Good Friday* *(no classes)*
Monday, April 17 Types read sections 7.1 & 7.2	Wednesday, April 19 Drag & drop programming	Friday, April 21 Drag & drop programming
Monday, April 24 Concurrent Programming read chapter 13	Wednesday, April 26 Concurrent Programming	Friday, April 28 Concurrent Programming
Monday, May 1 **Exam 3**	Wednesday, May 3 final review	
Tuesday, May 9 **Final Exam** 10:30am – 12:30pm		

Drag & Drop Programming Reading

http://appinventor.mit.edu/explore/ai2/tutorials.html

What to do with Prolog?

- Genealogy is the very favorite Prolog example
- Find the shortest path
- AI game playing
- Theorem Proving

Reading a Prolog Rule

- loves(chuck, X) :- female(X), rich(X).
- Declarative reading: Chuck loves X if X is female and rich.
- Approximate procedural reading: To find an X that Chuck loves, first find a female X, then check that X is rich.
- Declarative readings are almost always preferred.

slide from Tim Finin at UMBC

The Notion of Unification

- Unification is when two things “become one”
- Unification is kind of like assignment
- Unification is kind of like equality in algebra
- Unification is mostly like pattern matching
- Example:
  - loves(john, X) can unify with loves(john, mary)
  - and in the process, X gets unified with mary

slide from Tim Finin at UMBC

Unification I

- Any value can be unified with itself.
  - weather(sunny) = weather(sunny)
- A variable can be unified with another variable.
  - X = Y
- A variable can be unified with ("instantiated to") any Prolog value.
  - Topic = weather(sunny)

slide from Tim Finin at UMBC

Unification II

- Two different structures can be unified if their constituents can be unified.
  - mother(mary, X) = mother(Y, father(Z))
- A variable can be unified with a structure containing that same variable. This is usually a Bad Idea.
  - X = f(X)

slide from Tim Finin at UMBC

Unification III

- The explicit unification operator is =
- Unification is symmetric:

  Steve = father(isaac)

  means the same as

  father(isaac) = Steve
- Most unification happens implicitly, as a result of parameter transmission.

slide from Tim Finin at UMBC

Ordering

- Clauses are always tried in order
- buy(X) :- good(X).
  buy(X) :- cheap(X).

  cheap('Java 2 Complete').
  good('Thinking in Java').

- What will buy(X) choose first?

slide from Tim Finin at UMBC

Ordering II

- Try to handle more specific cases (those having more variables instantiated) first.

```
dislikes(john, bill).

dislikes(john, X) :- rich(X).

dislikes(X, Y) :- loves(X, Z), loves(Z, Y).
```

slide from Tim Finin at UMBC

Ordering III

- Some "actions" cannot be undone by backtracking over them:
  - write, nl, assert, retract, consult
- Do tests before you do undoable actions:
  - take(A) :-
    at(A, in_hand),
    write('You\'re already holding it!'),
    nl.

How does matching happen?

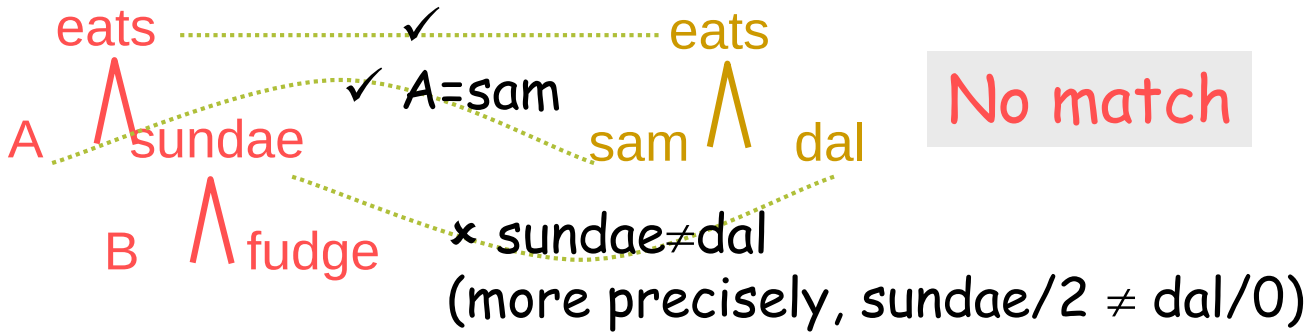
- eats(sam, dal).
- eats(josie, sundae(vanilla, caramel)).
- eats(rajiv, sundae(mintchip, fudge)).
- eats(robot('C-3PO'), Anything). % variable in a fact

- Query: eats(A, sundae(B,fudge)).
- Answer: A=rajiv, B=mintchip

How does matching happen?

- **eats(sam, dal).**
- eats(josie, sundae(vanilla, caramel)).
- eats(rajiv, sundae(mintchip, fudge)).
- eats(robot('C-3PO'), Anything). % variable in a fact

- Query: eats(A, sundae(B,fudge)).
- What happens when we try to match this against facts?



How does matching happen?

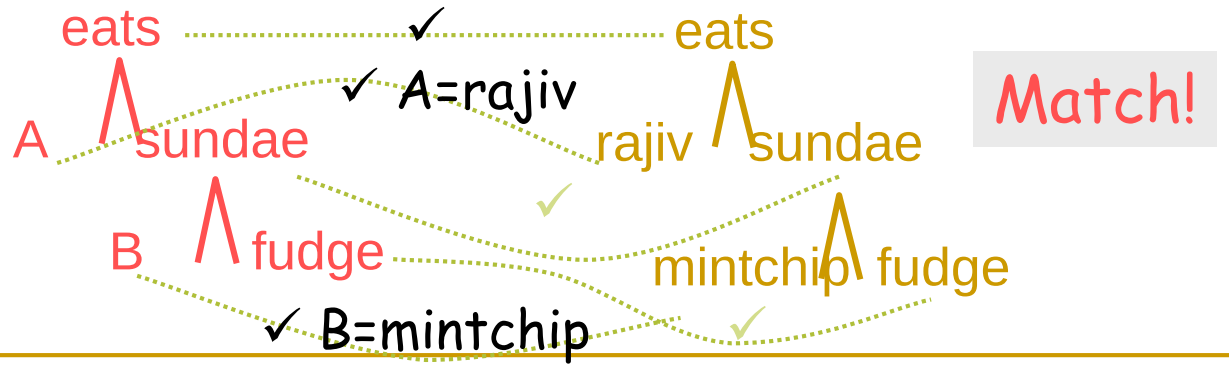
- eats(sam, dal).
- **eats(josie, sundae(vanilla, caramel)).**
- eats(rajiv, sundae(mintchip, fudge)).
- eats(robot('C-3PO'), Anything). % variable in a fact

- Query: eats(A, sundae(B,fudge)).
- What happens when we try to match this against facts?

How does matching happen?

- eats(sam, dal).
- eats(josie, sundae(vanilla, caramel)).
- **eats(rajiv, sundae(mintchip, fudge)).**
- eats(robot('C-3PO'), Anything). % variable in a fact

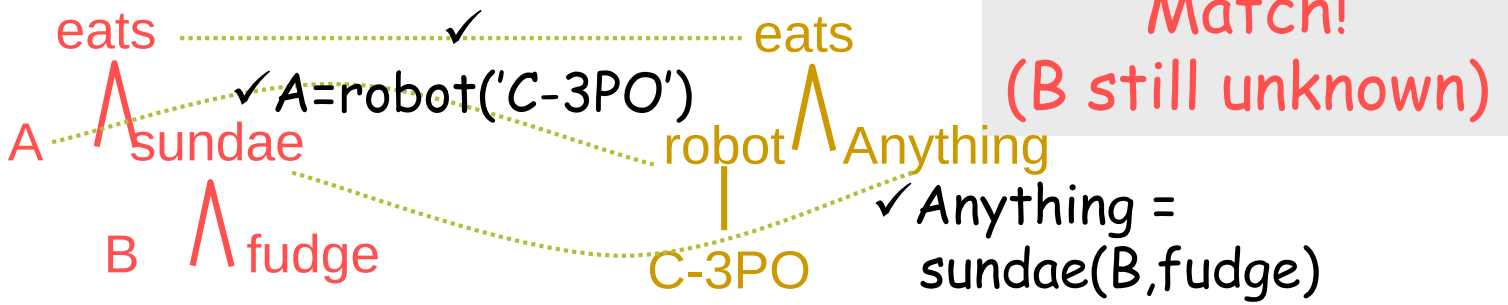
- Query: eats(A, sundae(B,fudge)).
- What happens when we try to match this against facts?



How does matching happen?

- eats(sam, dal).
- eats(josie, sundae(vanilla, caramel)).
- eats(rajiv, sundae(mintchip, fudge)).
- **eats(robot('C-3PO'), Anything).** % variable in a fact

- Query: eats(A, sundae(B,fudge)), icecream(B).
- What happens when we try to match this against facts?



How does matching happen?

- eats(sam, dal).
- eats(josie, sundae(vanilla, caramel)).
- eats(rajiv, sundae(mintchip, fudge)).
- **eats(robot('C-3PO'), Something) :- food(Something).**
- food(dal). icecream(vanilla).
- food(fudge). icecream(chocolate).
- **food(sundae(Base, Topping)) :- icecream(Base), food(Topping).**

- Query: eats(robot(A), sundae(B,fudge)).
- Answer: A='C-3PO', B can be any kind of ice cream

Recursion

- Handle the base cases first

```
ancestor(X, Y) :- child(Y, X).
```

(X is an ancestor of Y if Y is a child of X.)

- Recur only with a simpler case

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

(X is an ancestor of Y if Z is a child of X and Z is an ancestor of Y).

slide from Tim Finin at UMBC

Recursive Loops

- Prolog proofs must be tree structured, that is, they may not contain recursive loops.
  - child(X,Y) :- son(X,Y).
  - son(X,Y) :- child(X,Y), male(X).

  - ?- son(isaac, steven). <— *May loop!*
- Why? Neither child/2 nor son/2 is atomic

slide from Tim Finin at UMBC

Cut

- A cut, specified by an exclamation point (!), prevents backtracking from backing up before this point

```
dog :- animal, mammal, quadped, !, domestic, pet
```

- When a cut is encountered in a rule, Prolog becomes committed to all choices made since the parent goal was invoked
- An attempt to satisfy any goal between the parent goal and the cut will fail

Cut and Cut-fail

- The cut, !, is a commit point. It commits to:
  - the clause in which it occurs (can't try another)
  - everything up to that point in the clause
- Example:
  - `spouce(chuck, X) :- marriedTo(X), !, woman(X).`
  - Chuck is only married to one person. No use backing up to look for more
- Cut-fail, (!, `fail`), means give up *now* and don't even try for another solution

slide from Tim Finin at UMBC

Remaining Schedule

	Wednesday, April 12 Prolog	Friday, April 14 *Good Friday* *(no classes)*
Monday, April 17 Types read sections 7.1 & 7.2	Wednesday, April 19 Drag & drop programming	Friday, April 21 Drag & drop programming
Monday, April 24 Concurrent Programming read chapter 13	Wednesday, April 26 Concurrent Programming	Friday, April 28 Concurrent Programming
Monday, May 1 **Exam 3**	Wednesday, May 3 final review	
Tuesday, May 9 **Final Exam** 10:30am – 12:30pm		

Drag & Drop Programming Reading

http://appinventor.mit.edu/explore/ai2/tutorials.html