Abstraction
Reachability revisited
Prolog? `true`

February 6–7, 2012
CS 60: Principles of Computer Science

Assignment 2 due: Higher-Order Functions, Trees, Graphs, and Java(!)
Assignment 3 due Monday: Prolog

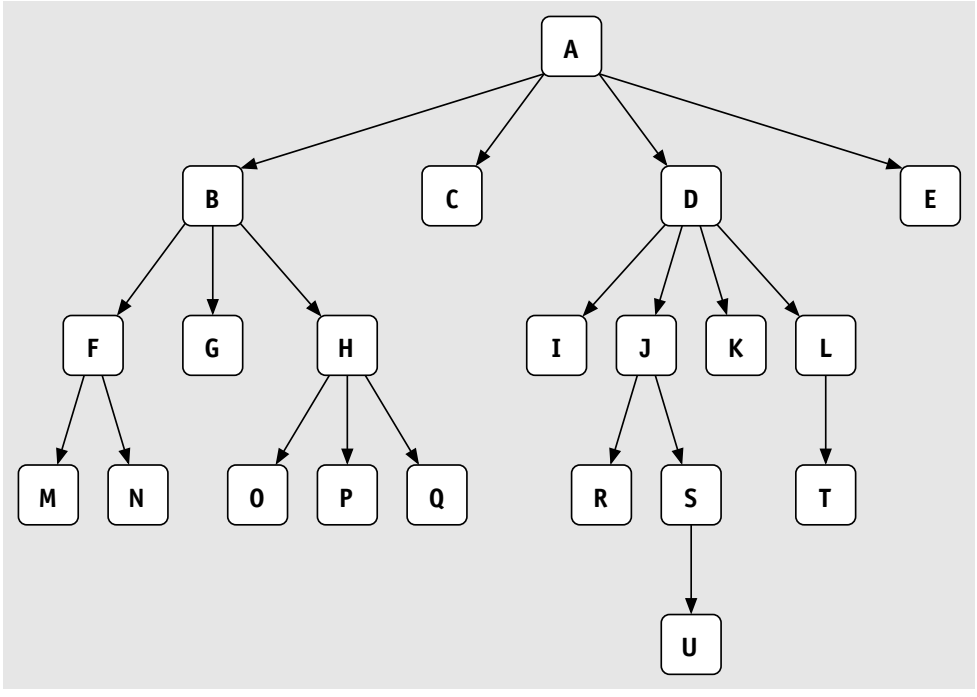
Interface vs. Implementation

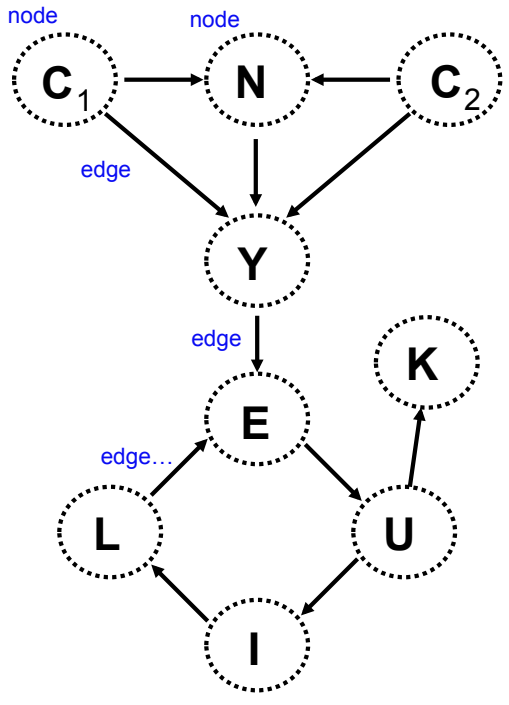
1. What operations should a *set* support?
2. What operations should a *tree* support?
3. What operations should a *list* support?

---

1. How could a *set* be implemented?
2. How could a *tree* be implemented?
3. How could a *list* be implemented?

Reachability Revisited

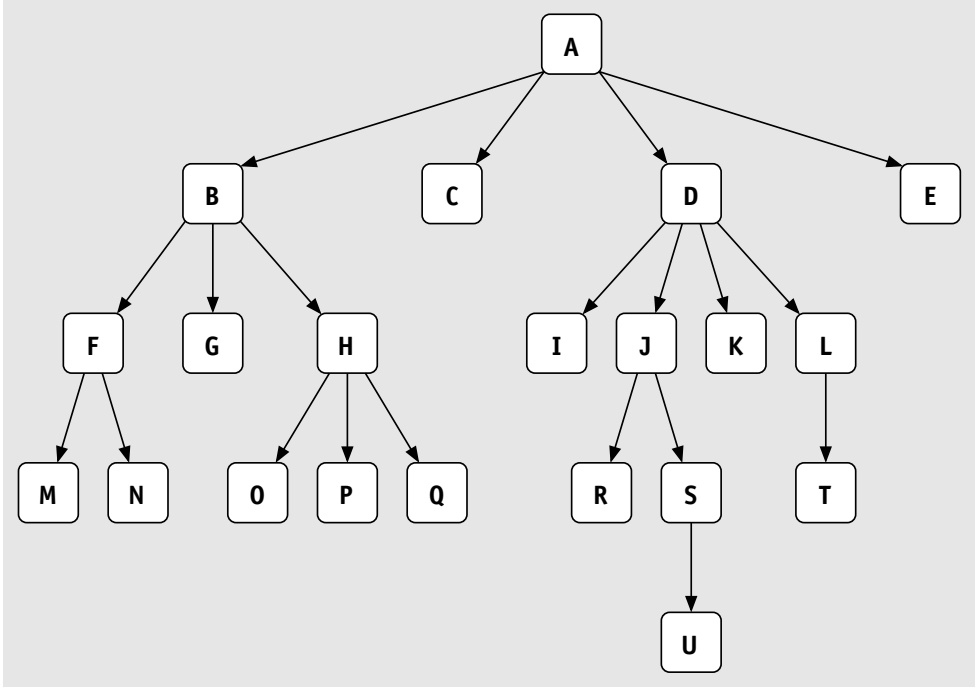


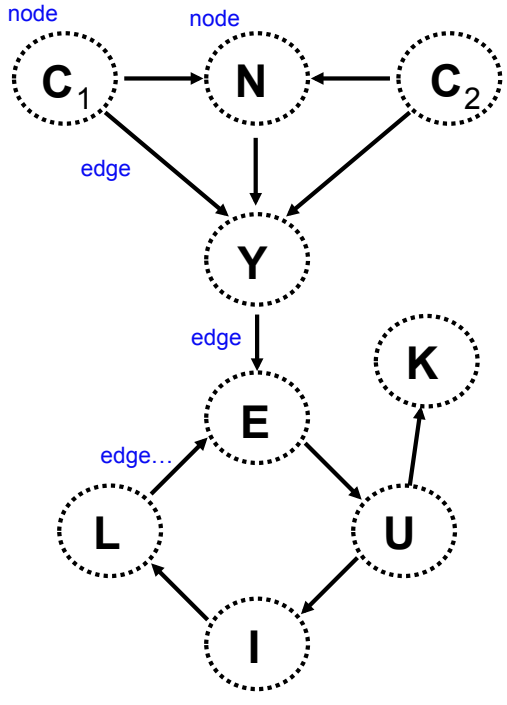


The use-it-or-lose-it algorithm is simple and it works, but it's not very intuitive (or efficient).

Algorithm: Depth-First Search

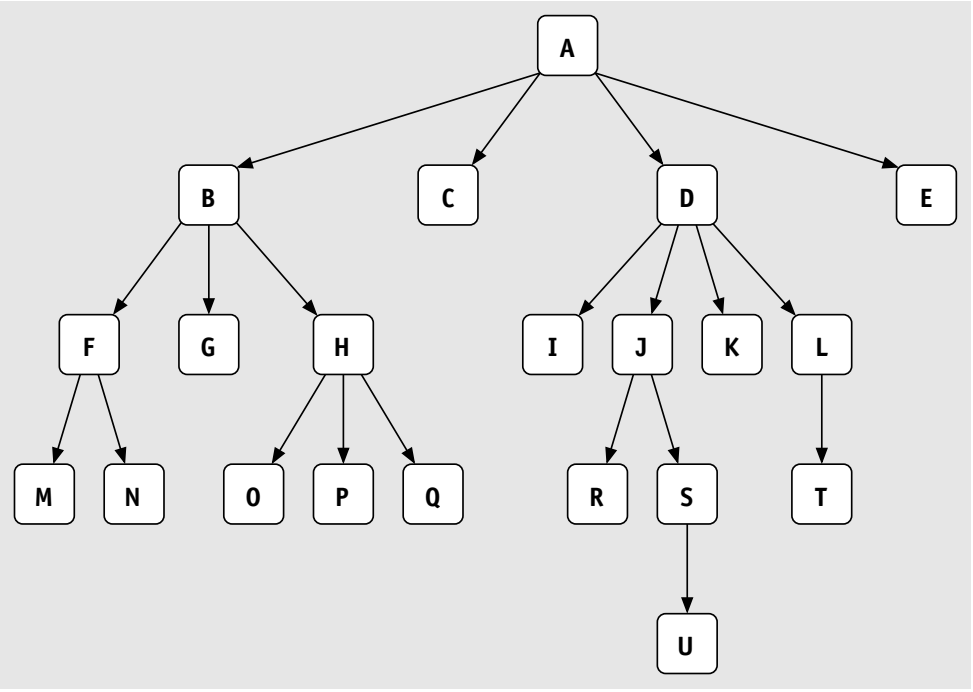
- ✓ Check the current node(Optional: Return if you've already searched it)
- ✓ (Recursively) search everything reachable from the first child
- ✓ (Recursively) search everything reachable from the second child
- ✓ ...
- ✓ (Recursively) search everything reachable from the last child

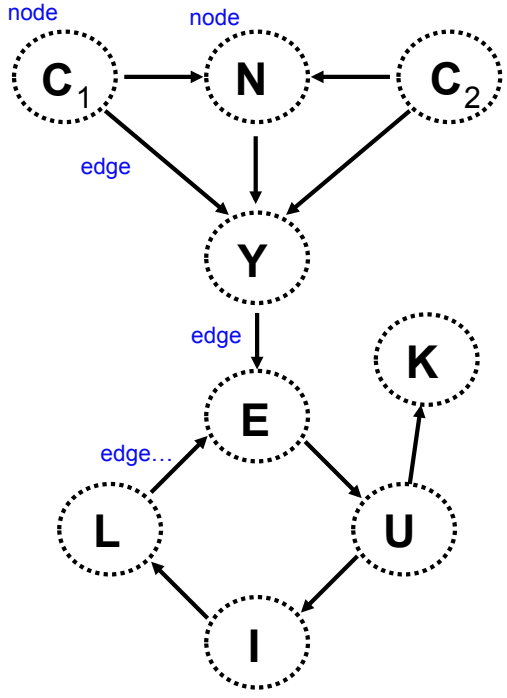




Algorithm: Breadth-First Search

- ✓ Check the current node
- ✓ Check the children
- ✓ Check the children's children
- ✓ Check the children's children's children
- ✓ ...





Coming Attractions

We will return to these ideas in a few weeks, when you'll need to implement one or both algorithms!

Prolog

CS 60: Principles of Computer Science

Prolog: The Beautiful Dream

We feed in a bunch of facts relevant to our problem:

```
pairs(apple, walnut).
pairs(apple, honey).
pairs(walnut, avocado).
pairs(walnut, banana).
pairs(apple, banana).
pairs(banana, ginger).
pairs(banana, cloves).
pairs(banana, strawberry).
pairs(banana, coriander).
pairs(strawberry, honey).
pairs(strawberry, ginger).
pairs(strawberry, tea).
pairs(tea, walnut).
pairs(tea, tomato).
pairs(tea, milk).

pairs(X, X).
pairs(X, coconut).
```

We describe how to recognize a solution:

```
yummy_triple(X,Y,Z) :- pairs(X,Y),  X \= Y,
                       pairs(Y,Z),  Y \= Z,
                       pairs(X,Z),  X \= Z.
```

Prolog then finds solution(s) for us.

So get it now! `http://www.swi-prolog.org/`

Applications of Prolog

Natural Language Processing With Prolog in the IBM Watson System

March 31, 2011

By Adam Lally (1) and Paul Fodor (2)

(1) IBM Thomas J. Watson Research Center
(2) Stony Brook University

On February 14-16, 2011, the IBM Watson question answering system won the JeopardyMan vs. Machine Challenge by defeating two former grand champions, Ken Jennings and Brad Rutter. To compete successfully at Jeopardy!, Watson had to answer complex natural language questions over an extremely broad domain of knowledge. Moreover, it had to compute an accurate confidence in its answers and to complete its processing in a very short amount of time.

POETS & POETRY: He was a bank clerk in the Yukon before he published "Songs of a Sourdough" in 1907

The output of the parser includes, among many other things, that "published" is a verb with base form (or lemma) "publish", subject "he", and object "Songs of a Sourdough".

Next, Watson applies numerous detection rules that match patterns in the parse. These rules detect elements such as the focus of the question (the words that refer to the answer, in this case "he"), the lexical answer types (terms in the question or category that indicate what type of entity is being asked for, in this case "poet"), and the relationships between entities in either a question or a potential supporting passage.

We required a language in which we could conveniently express pattern matching rules over the parse trees and other annotations (such as named entity recognition results), and a technology that could execute these rules very efficiently. We found that Prolog was the ideal choice for the language due to its simplicity and expressiveness. The information in the parse is easily converted into Prolog facts, such as (the numbers representing unique identifiers for parse nodes):

```
lemma(1, "he").
partOfSpeech(1,pronoun).
lemma(2, "publish").
partOfSpeech(2,verb).
lemma(3, "Songs of a Sourdough").
partOfSpeech(3,noun).
subject(2,1).
object(2,3).
```

Such facts were consulted into a Prolog system and several rule sets were executed to detect the focus of the question, the lexical answer type and several relations between the elements of the parse. A simplified rule for detecting the authorOf relation can be written in Prolog as follows:

Prolog Data is like Racket Data

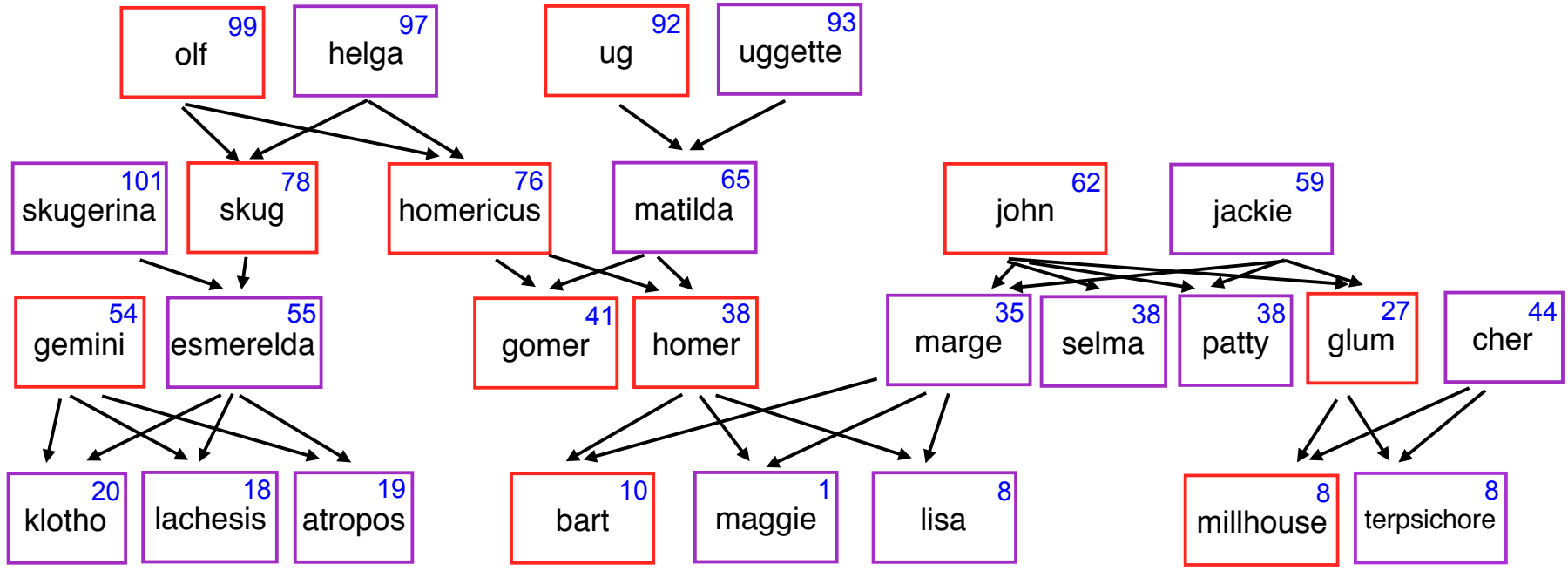
Integers:	`42` `60`
Floating Point:	`3.14` `2.17`
Symbols	`spam` `x` ← always lowercase!
Strings:	`'hi'`
Lists:	`[3.14, 42, spam, a, 'hi']`

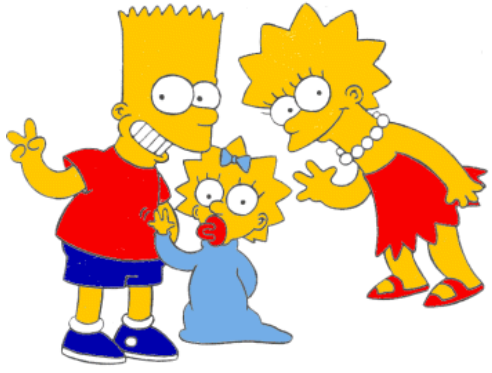
But Prolog *programs* feel completely different!

Prolog: Just the Facts

```
%% Here is a comment in Prolog
/* this is also a comment */

Chocolate_is_tasty.  %% A fact.  A good one too!

good(spam).  %% a debatable fact, but now prolog thinks so
good(42).    %% no one can deny this


better(tofu, spam).          %% tofu is better than spam
better(chocolate, tofu).     %% chocolate is better than tofu
better(money, chocolate).    %% money is better than chocolate


%% Does prolog know that money is better than spam?
%% Does prolog even know what spam is???
```

(Hypothetical) Simpsons Family Tree





Oh, **brother**!

Excerpts from simpsons.pl

```
%% Parent relation
parent(homer, bart).
parent(marge, bart).
parent(homer, lisa).
parent(marge, lisa).
parent(homer, maggie).
parent(marge, maggie).
```

```
%% Age relation
age(marge, 35).
age(homer, 38).
age(lisa, 8).
age(maggie, 1).
age(bart, 10).
age(gomer, 41).
```

```
%% Female predicate
female(marge).
female(jackie).
female(selma).
female(patty).
female(cher).
female(lisa).
```

```
%% Male predicate
male(homer).
male(gomer).
male(gemini).
male(glum).
male(bart).
male(millhouse).
```

```
%% Three rules about families

child(X, Y) :- parent(Y, X).

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

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

Using Prolog

- To run: `/opt/local/bin/swipl` **(macs)**
  `swipl` **(ssh to knuth.cs.hmc.edu server)**
- To exit: `halt.`
- To (re)load a file: `[file].` or `reconsult('file.pl').`

`[user].` lets you type facts

- The Prolog environment is for queries only!
  Use your favorite editor to change files…

Windows's `swipl` has an editor built-in.

Writing Queries in Prolog

Is Lisa a child of Marge?	`child(lisa,marge).`
Who has Marge as a parent?	`parent(marge,X).`
Is Ug a parent of anyone?	`parent(ug,_).`
Who are the ancestors of Lisa?	`anc(A,lisa).`
Who are Uggette's descendants?	`anc(uggette,D).`
Who is Maggie's mother?	`parent(X,maggie), female(X).`
Who is a person (you know about)?	`male(X); female(X).`
Who is Bart's age or younger?	`age(bart, N), age(P, M), M < N.`

Name: "Quiz" Two functions for BSTs

Define these three family-relationship predicates

1. `sib(X,Y)` — true when `X` and `Y` are siblings (or half-siblings)

```
sibs(X,Y) :- parent(P,X), parent(P,Y), X =\= Y.
```

2. `aunt(A,N)` — true when `A` is `N`'s aunt.

```
aunt(A,N) :- parent(P,N), sib(A,P), female(A).
```

3. `rel(X,Y)` — true when `X` and `Y` are related (by blood)

```
rel(X,Y) :- anc(Z, X), anc(Z, Y).
```