LECTURE 14 OF 42

First-Order Logic: Unification, Inference
Discussion: PS3, Constraint Logic

Monday, 25 September 2006

William H. Hsu
Department of Computing and Information Sciences, KSU

KSOL course page: **http://snipurl.com/v9v3**
Course web site: **http://www.kddresearch.org/Courses/Fall-2006/CIS730**
Instructor home page: **http://www.cis.ksu.edu/~bhsu**

Reading for Next Class:
Section 9.2 – 9.4, p. 275 – 295, Russell & Norvig 2nd edition

---

LECTURE OUTLINE

- **Reading for Next Class: Section 9.2 – 9.4, R&N 2e**
- **Recommended : Nilsson and Genesereth (Chapter 5 online)**
- **Today**
  - **Generalized Modus Ponens**
  - **Unification**
  - **Constraint logic**
- **Wednesday**
  - **Resolution theorem proving**
  - **Prolog as related to resolution**
  - **MP4 & 5 preparations**
- **Friday**
  - **Logic programming in real life**
  - **Industrial-strength Prolog**
  - **Lead-in to MP4**
- **Week of 04 Oct 2006: KR and Ontologies**

LOGICAL AGENTS: REVIEW

Logical agents apply inference to a knowledge base
to derive new information and make decisions

Basic concepts of logic:

- syntax: formal structure of sentences
- semantics: truth of sentences wrt models
- entailment: necessary truth of one sentence given another
- inference: deriving sentences from other sentences
- soundess: derivations produce only entailed sentences
- completeness: derivations can produce all entailed sentences

Wumpus world requires the ability to represent partial and negated information, reason by cases, etc.

Propositional logic suffices for some of these tasks

Adapted from slides by
S. Russell, UC Berkeley

EXAMPLE PROOF: REVIEW

Bob is a buffalo	1. $Buffalo(Bob)$
Pat is a pig	2. $Pig(Pat)$
Buffaloes outrun pigs	3. $\forall x, y \;\; Buffalo(x) \land Pig(y) \Rightarrow Faster(x,y)$
Bob outruns Pat	

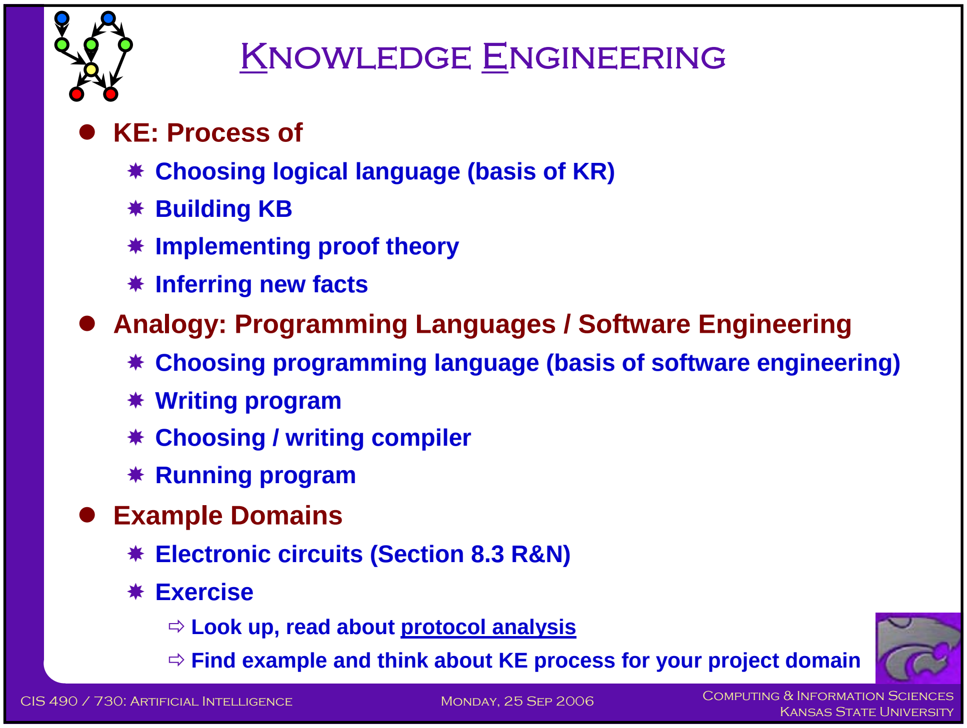
- Apply Sequent Rules to Generate New Assertions

AI 1 & 2	4. $Buffalo(Bob) \land Pig(Pat)$
UE 3, $\{x/Bob, y/Pat\}$	5. $Buffalo(Bob) \land Pig(Pat) \Rightarrow Faster(Bob, Pat)$
MP 6 & 7	6. $Faster(Bob, Pat)$

$$\frac{\alpha, \quad \alpha \Rightarrow \beta}{\beta} \qquad \frac{\alpha \quad \beta}{\alpha \land \beta} \qquad \frac{\forall x \;\; \alpha}{\alpha\{x/\tau\}}$$

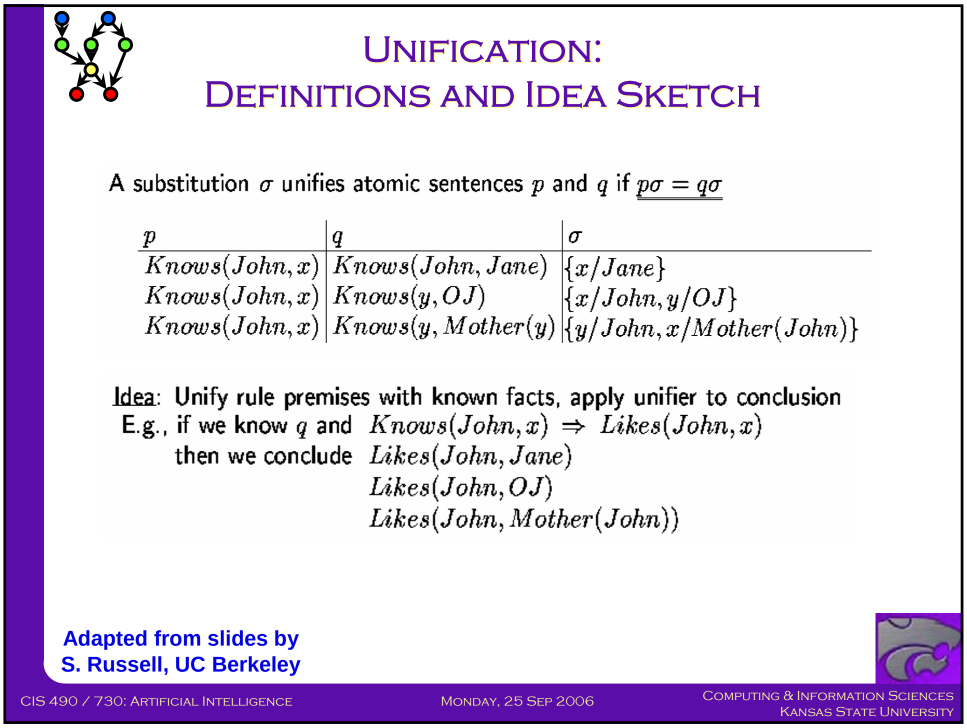
- Modus Ponens — And Introduction — Universal Elimination

Adapted from slides by
S. Russell, UC Berkeley



Knowledge Engineering

- **KE: Process of**
  - Choosing logical language (basis of KR)
  - Building KB
  - Implementing proof theory
  - Inferring new facts
- **Analogy: Programming Languages / Software Engineering**
  - Choosing programming language (basis of software engineering)
  - Writing program
  - Choosing / writing compiler
  - Running program
- **Example Domains**
  - Electronic circuits (Section 8.3 R&N)
  - Exercise
    - ⇨ Look up, read about protocol analysis
    - ⇨ Find example and think about KE process for your project domain



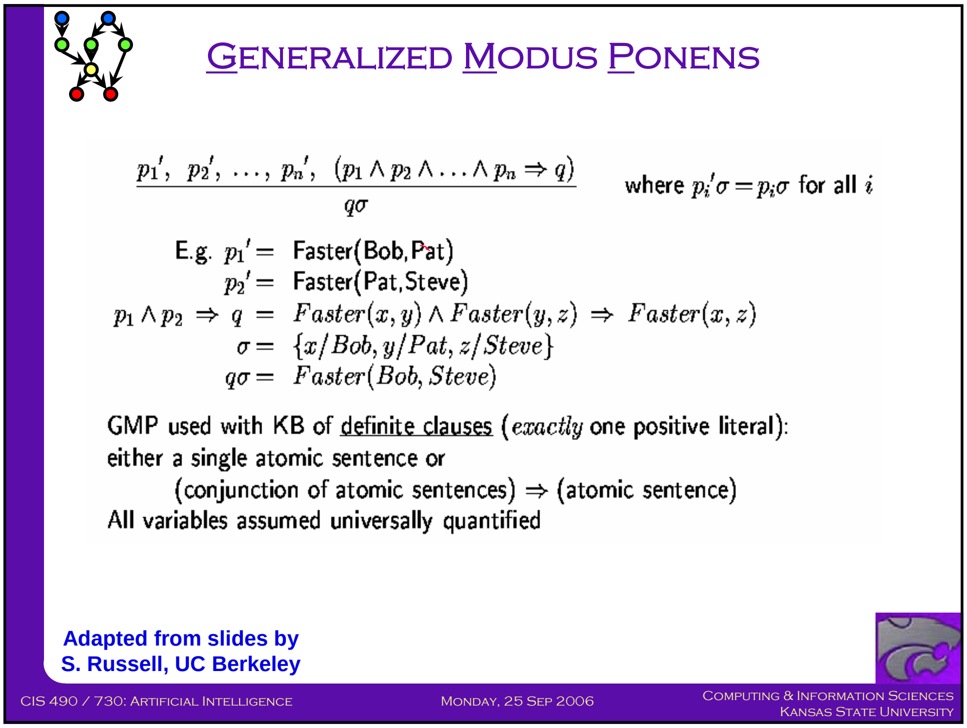
Unification:
Definitions and Idea Sketch

A substitution σ unifies atomic sentences p and q if $\underline{\underline{p\sigma = q\sigma}}$

p	q	σ
$Knows(John, x)$	$Knows(John, Jane)$	$\{x/Jane\}$
$Knows(John, x)$	$Knows(y, OJ)$	$\{x/John, y/OJ\}$
$Knows(John, x)$	$Knows(y, Mother(y))$	$\{y/John, x/Mother(John)\}$

Idea: Unify rule premises with known facts, apply unifier to conclusion
E.g., if we know q and $Knows(John, x) \Rightarrow Likes(John, x)$
then we conclude $Likes(John, Jane)$
$Likes(John, OJ)$
$Likes(John, Mother(John))$

Adapted from slides by
S. Russell, UC Berkeley



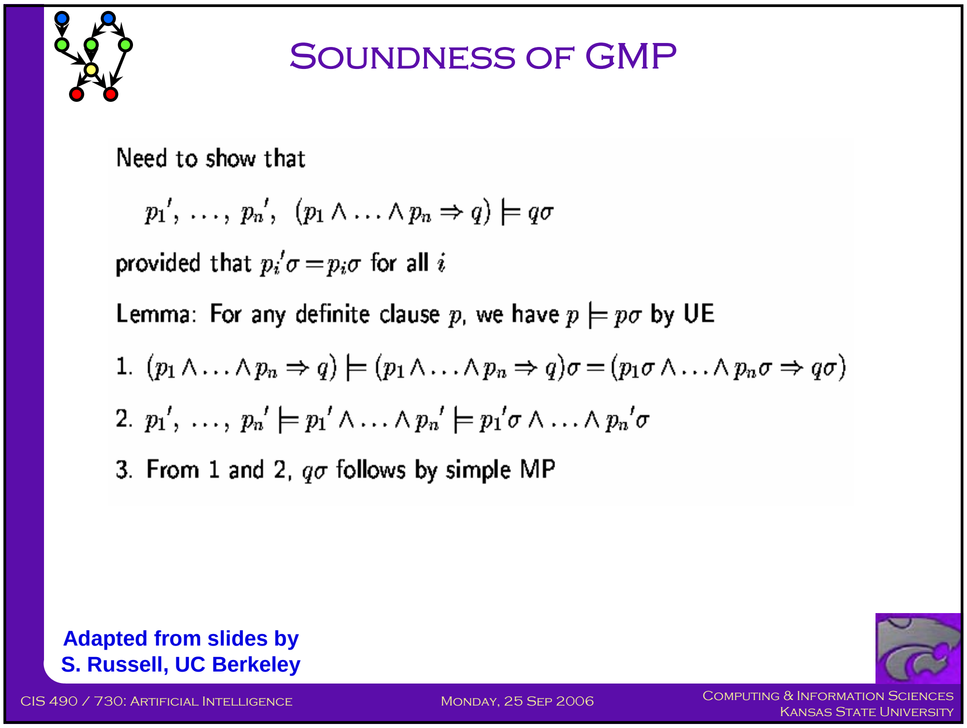
GENERALIZED MODUS PONENS

$$\frac{p_1',\ p_2',\ \ldots,\ p_n',\ (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{q\sigma} \qquad \text{where } p_i'\sigma = p_i\sigma \text{ for all } i$$

$$\begin{aligned} \text{E.g. } p_1' &= \text{Faster(Bob,Pat)} \\ p_2' &= \text{Faster(Pat,Steve)} \\ p_1 \wedge p_2 \Rightarrow q &= Faster(x,y) \wedge Faster(y,z) \Rightarrow Faster(x,z) \\ \sigma &= \{x/Bob, y/Pat, z/Steve\} \\ q\sigma &= Faster(Bob, Steve) \end{aligned}$$

GMP used with KB of definite clauses (*exactly* one positive literal):
either a single atomic sentence or
(conjunction of atomic sentences) ⇒ (atomic sentence)
All variables assumed universally quantified

Adapted from slides by
S. Russell, UC Berkeley



SOUNDNESS OF GMP

Need to show that

$$p_1',\ \ldots,\ p_n',\ (p_1 \wedge \ldots \wedge p_n \Rightarrow q) \models q\sigma$$

provided that $p_i'\sigma = p_i\sigma$ for all i

Lemma: For any definite clause p, we have $p \models p\sigma$ by UE

1. $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \models (p_1 \wedge \ldots \wedge p_n \Rightarrow q)\sigma = (p_1\sigma \wedge \ldots \wedge p_n\sigma \Rightarrow q\sigma)$
2. $p_1',\ \ldots,\ p_n' \models p_1' \wedge \ldots \wedge p_n' \models p_1'\sigma \wedge \ldots \wedge p_n'\sigma$
3. From 1 and 2, $q\sigma$ follows by simple MP

Adapted from slides by
S. Russell, UC Berkeley

FORWARD CHAINING

```
When a new fact $p$ is added to the KB
    for each rule such that $p$ unifies with a premise
        if the other premises are known
        then add the conclusion to the KB and continue chaining
```

Forward chaining is data-driven

e.g., inferring properties and categories from percepts

Adapted from slides by
S. Russell, UC Berkeley

EXAMPLE:
FORWARD CHAINING

Add facts 1, 2, 3, 4, 5, 7 in turn.

Number in [] = unification literal; $\surd$ indicates rule firing

1. $Buffalo(x) \wedge Pig(y) \Rightarrow Faster(x, y)$
2. $Pig(y) \wedge Slug(z) \Rightarrow Faster(y, z)$
3. $Faster(x, y) \wedge Faster(y, z) \Rightarrow Faster(x, z)$
4. $Buffalo(Bob)$ $[1a, \times]$
5. $Pig(Pat)$ $[1b, \surd] \rightarrow$ 6. $Faster(Bob, Pat)$ $[3a, \times]$, $[3b, \times]$ $[2a, \times]$
7. $Slug(Steve)$ $[2b, \surd]$
    - $\rightarrow$ 8. $Faster(Pat, Steve)$ $[3a, \times]$, $[3b, \surd]$
        - $\rightarrow$ 9. $Faster(Bob, Steve)$ $[3a, \times]$, $[3b, \times]$

Adapted from slides by
S. Russell, UC Berkeley

BACKWARD CHAINING

```
When a query $q$ is asked
    if a matching fact $q'$ is known, return the unifier
    for each rule whose consequent $q'$ matches $q$
        attempt to prove each premise of the rule by backward chaining
```

(Some added complications in keeping track of the unifiers)

(More complications help to avoid infinite loops)

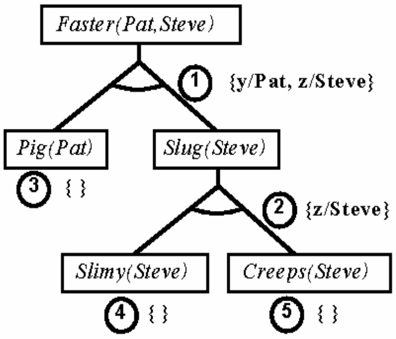
Two versions: find any solution, find all solutions

Backward chaining is the basis for logic programming, e.g., Prolog

Adapted from slides by
S. Russell, UC Berkeley

EXAMPLE:
BACKWARD CHAINING

1. $Pig(y) \wedge Slug(z) \Rightarrow Faster(y, z)$
2. $Slimy(z) \wedge Creeps(z) \Rightarrow Slug(z)$
3. $Pig(Pat)$ 4. $Slimy(Steve)$ 5. $Creeps(Steve)$



Adapted from slides by
S. Russell, UC Berkeley

BACKWARD CHAINING

```
When a query $q$ is asked
    if a matching fact $q'$ is known, return the unifier
    for each rule whose consequent $q'$ matches $q$
        attempt to prove each premise of the rule by backward chaining
```

(Some added complications in keeping track of the unifiers)

(More complications help to avoid infinite loops)

Two versions: find any solution, find all solutions

Backward chaining is the basis for logic programming, e.g., Prolog

- Answer
  - Suppose ¬*Query*, For The Sake Of Contradiction (FTSOC)
  - Attempt to prove that $KB \land \neg Query \vdash \bot$

RESOLUTION INFERENCE RULE

Basic propositional version:

$$\frac{\alpha \lor \beta, \quad \neg\beta \lor \gamma}{\alpha \lor \gamma} \quad \text{or equivalently} \quad \frac{\neg\alpha \Rightarrow \beta, \quad \beta \Rightarrow \gamma}{\neg\alpha \Rightarrow \gamma}$$

Full first-order version:

$$\frac{\begin{array}{c} p_1 \lor \dots\ p_j\ \dots \lor p_m, \\ q_1 \lor \dots\ q_k\ \dots \lor q_n \end{array}}{(p_1 \lor \dots\ p_{j-1} \lor p_{j+1}\ \dots p_m \lor q_1 \dots\ q_{k-1} \lor q_{k+1}\ \dots \lor q_n)\sigma}$$

where $p_j\sigma = \neg q_k\sigma$

For example,

$$\frac{\begin{array}{l} \neg Rich(x) \lor Unhappy(x) \\ Rich(Me) \end{array}}{Unhappy(Me)}$$

with $\sigma = \{x/Me\}$

Adapted from slides by
S. Russell, UC Berkeley

Conjunctive Normal (*aka* Clausal) Form: Conversion (Nilsson) and Mnemonic

- Implications Out
- Negations Out
- Standardize Variables Apart
- Existentials Out (Skolemize)
- Universals Made Implicit
- Distribute *And* Over *Or* (i.e., Disjunctions In)
- Operators Out
- Rename Variables
- A Memonic for *Star Trek: The Next Generation* Fans

Captain Picard:

I'll Notify Spock's Eminent Underground Dissidents On Romulus

I'll Notify Sarek's Eminent Underground Descendant On Romulus

Adapted from slides by S. Russell, UC Berkeley

Skolemization

$\exists x\ Rich(x)$ becomes $Rich(G1)$ where $G1$ is a new "Skolem constant"

$\exists\, k\ \ \frac{d}{dy}(k^y) = k^y$ becomes $\frac{d}{dy}(e^y) = e^y$

More tricky when $\exists$ is inside $\forall$

E.g., "Everyone has a heart"
$\forall x\ \ Person(x) \ \Rightarrow\ \exists y\ \ Heart(y) \wedge Has(x,y)$

Incorrect:
$\forall x\ \ Person(x) \ \Rightarrow\ Heart(H1) \wedge Has(x,H1)$

Correct:
$\forall x\ \ Person(x) \ \Rightarrow\ Heart(H(x)) \wedge Has(x,H(x))$
where H is a new symbol ("Skolem function")

Skolem function arguments: all enclosing universally quantified variables

Adapted from slides by S. Russell, UC Berkeley

RESOLUTION THEOREM PROVING

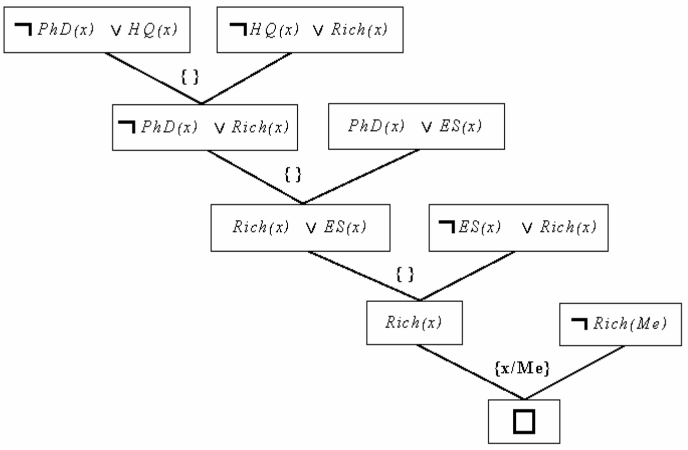
To prove α:
- negate it
- convert to CNF
- add to CNF KB
- infer contradiction

E.g., to prove $Rich(me)$, add $\neg Rich(me)$ to the CNF KB

$$\neg PhD(x) \lor HighlyQualified(x)$$
$$PhD(x) \lor EarlyEarnings(x)$$
$$\neg HighlyQualified(x) \lor Rich(x)$$
$$\neg EarlyEarnings(x) \lor Rich(x)$$

Adapted from slides by S. Russell, UC Berkeley

EXAMPLE: RESOLUTION PROOF



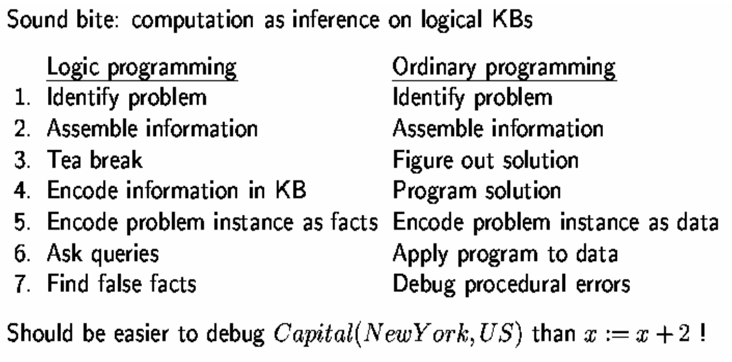
Adapted from slides by S. Russell, UC Berkeley

OFFLINE EXERCISE: READ-AND-EXPLAIN PAIRS

- For Class Participation (PS5)
- With Your Assigned Partner(s)

- Read: Chapter 10 R&N
- By 14 Oct 2006

LOGIC PROGRAMMING VS. IMPERATIVE PROGRAMMING



Sound bite: computation as inference on logical KBs

	Logic programming	Ordinary programming
1.	Identify problem	Identify problem
2.	Assemble information	Assemble information
3.	Tea break	Figure out solution
4.	Encode information in KB	Program solution
5.	Encode problem instance as facts	Encode problem instance as data
6.	Ask queries	Apply program to data
7.	Find false facts	Debug procedural errors

Should be easier to debug $Capital(NewYork, US)$ than $x := x + 2$!

Adapted from slides by S. Russell, UC Berkeley

A LOOK AHEAD: LOGIC PROGRAMMING AS HORN CLAUSE RESOLUTION

Basis: backward chaining with Horn clauses + bells & whistles
Widely used in Europe, Japan (basis of 5th Generation project)
Compilation techniques ⇒ 10 million LIPS

Program = set of clauses = head :- literal$_1$, ... literal$_n$.
Efficient unification by open coding
Efficient retrieval of matching clauses by direct linking
Depth-first, left-to-right backward chaining
Built-in predicates for arithmetic etc., e.g., `X is Y*Z+3`
Closed-world assumption ("negation as failure")
  e.g., `not PhD(X)` succeeds if `PhD(X)` fails

Adapted from slides by S. Russell, UC Berkeley

A LOOK AHEAD: LOGIC PROGRAMMING (PROLOG) EXAMPLES

Depth-first search from a start state `X`:

```
dfs(X) :- goal(X).
dfs(X) :- successor(X,S),dfs(S).
```

No need to loop over `S`: `successor` succeeds for each

Appending two lists to produce a third:

```
append([],Y,Y).
append([X|L],Y,[X|Z]) :- append(L,Y,Z).

query:    append(A,B,[1,2]) ?
answers: A=[]     B=[1,2]
         A=[1]    B=[2]
         A=[1,2]  B=[]
```

Adapted from slides by S. Russell, UC Berkeley

SUMMARY POINTS

- From Propositional to First-Order Proofs
  - Generalized Modus Ponens
  - Resolution
- Unification Problem
- Roles in Computer Science
  - Type inference
  - Theorem proving
  - What do these have to do with each other?
- Search Patterns
  - Forward chaining
  - Backward chaining
  - Fan-in, fan-out

TERMINOLOGY

- From Propositional to First-Order Proofs
  - Generalized Modus Ponens
  - Resolution
- Unification Problem
- Most General Unifier (MGU)
- Roles in Computer Science
  - Type inference
  - Theorem proving
  - What do these have to do with each other?
- Search Patterns
  - Forward chaining
  - Backward chaining
  - Fan-in, fan-out