Knowledge representation and Reasoning

Artificial Intelligence

Based on the books

AIMA (Russell and Norvig), MIT Open Courseware and KRR (Brachman and Levesque),

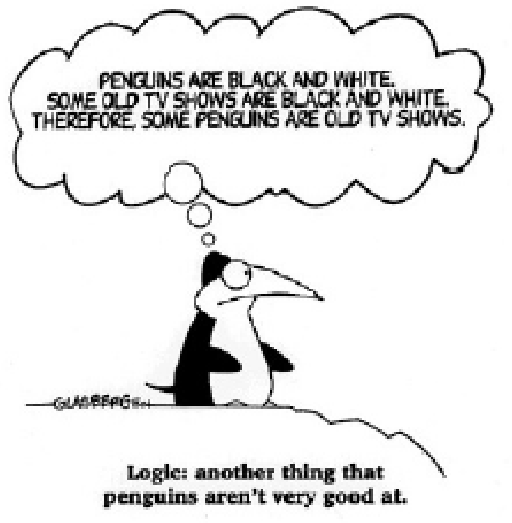
Slides are mostly adapted from

Stuart C. Shapiro (University of Buffalo), Tom Lenaerts (IRIDIA),

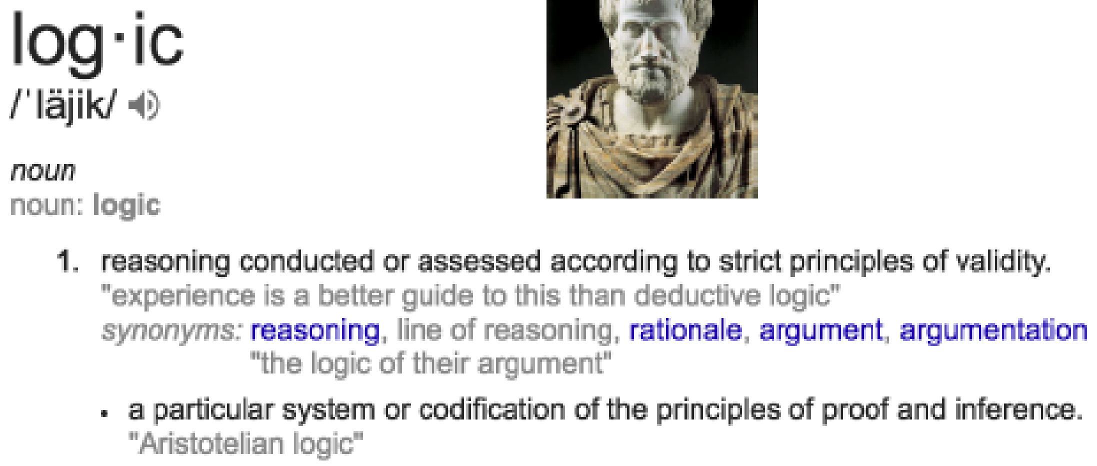
Milos Hauskrecht (U. Pittsburgh), Sanjeev Arora and Elad Hazan (Princeton U.)

Peter Lucas and Marcel van Gerven (Radboud University Nijmegen)

LOGIC


PENGUINS ARE BLACK AND WHITE.
SOME OLD TV SHOWS ARE BLACK AND WHITE.
THEREFORE, SOME PENGUINS ARE OLD TV SHOWS.

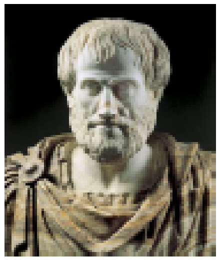
Logic: another thing that penguins aren't very good at.



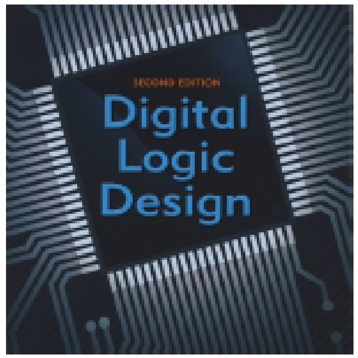
log·ic
/ˈläjik/

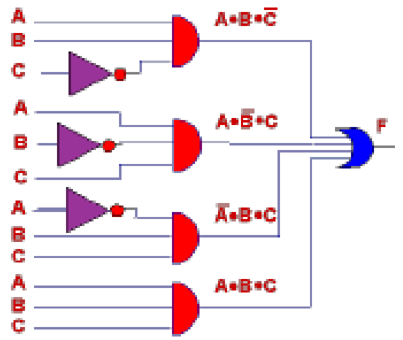
noun
noun: **logic**

1. reasoning conducted or assessed according to strict principles of validity.
   "experience is a better guide to this than deductive logic"
   synonyms: reasoning, line of reasoning, rationale, argument, argumentation
   "the logic of their argument"
   - a particular system or codification of the principles of proof and inference.
     "Aristotelian logic"



Also basis of digital circuits in computer chips




A
B
C
$A \cdot B \cdot \overline{C}$
A
B
C
$A \cdot \overline{B} \cdot C$
F
A
B
C
$\overline{A} \cdot B \cdot C$
A
B
C
$A \cdot B \cdot C$

Role of logic in AI

For 2000 years, people tried to codify "human reasoning" and came up with logic.

• Most AI work until 1980s: Build machines that represent knowledge and do reasoning via logic. "Rule based reasoning."

a program has common sense if it automatically deduces for itself a sufficiently wide class of immediate consequences of anything it is told and what it already knows. . . In order for a program to be capable of learning something it must first be capable of being told it. John McCarthy, "Programs with Common Sense", 1959.

Requirements for a knowledge based agent

- 1. *"what it already knows" [McCarthy '59]*

  A knowledge base of beliefs.

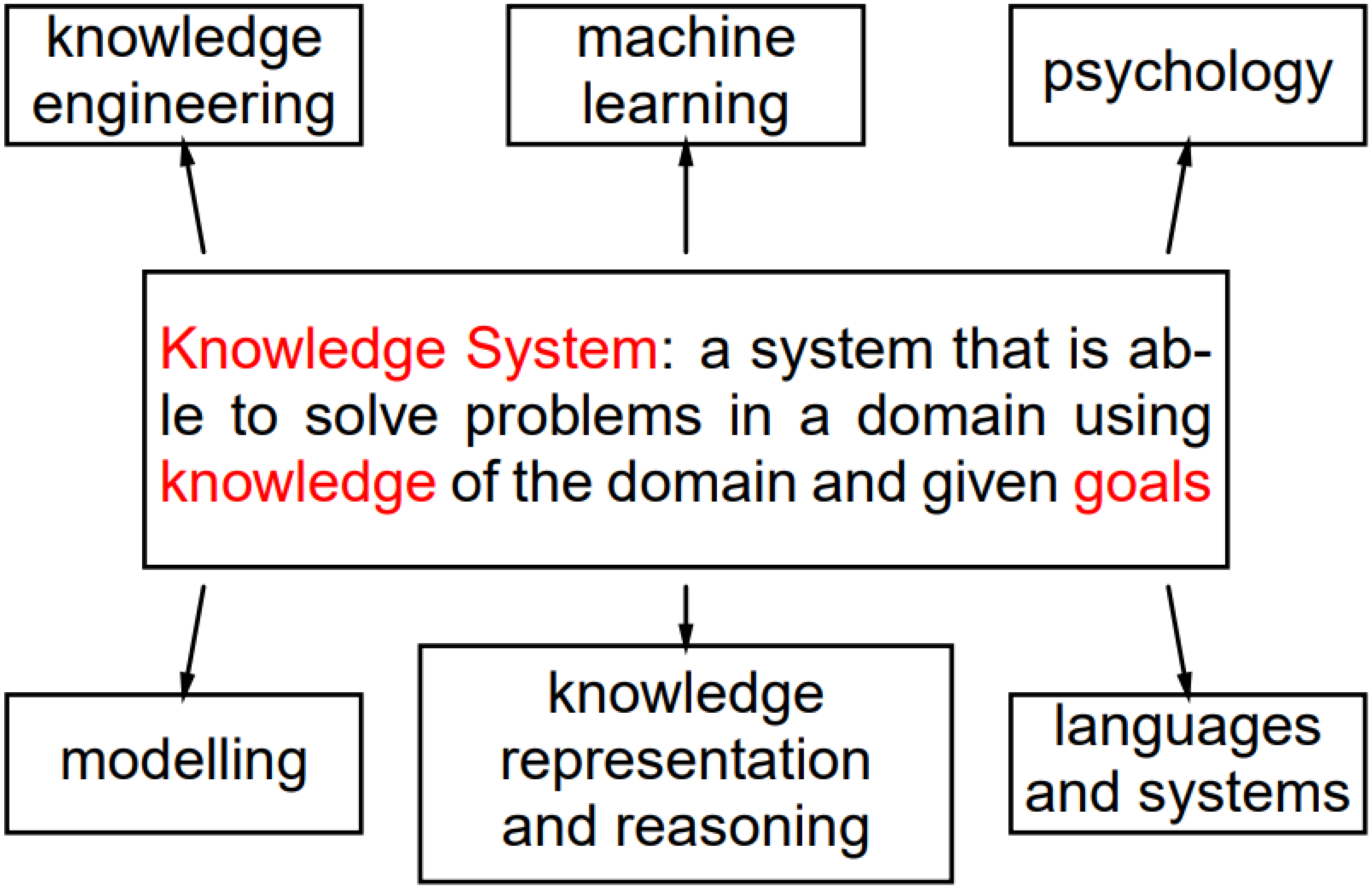
- 2. *"it must first be capable of being told" [McCarthy '59]*

  A way to put new beliefs into the knowledge base.

- 3. *"automatically deduces for itself a sufficiently wide class of immediate consequences" [McCarthy '59]*

  A reasoning mechanism to derive new beliefs from ones already in the knowledge base.

Knowledge Base System


knowledge engineering
machine learning
psychology
Knowledge System: a system that is able to solve problems in a domain using knowledge of the domain and given goals
modelling
knowledge representation and reasoning
languages and systems

Early Knowledge Base Systems

- Much of AI involves building systems that are knowledge based
  - Language understanding
  - Planning
  - Diagnosis
  - Expert systems, etc.

- Expert system: use of a large collection of symbolic expert knowledge to solve problems:
  - E.A. Feigenbaum, B.G. Buchanan, J. Lederberg – Heuristic DENDRAL (1965): contains knowledge from organic chemistry
  - E.H. Shortliffe: MYCIN (1974–1979) – diagnostics of infectious diseases
  - H.E. Pople, J.D. Myers: Internist-1 (1973-1982) – diagnosis in the big area of internal medicine
  - D. Lenat: Cyc (1984-) – representation of common sense knowledge

Advantage of Knowledge Base Systems

Knowledge-based system most suitable for *open-ended* tasks

can structurally isolate *reasons* for particular behaviour

Good for

- explanation and justification
  - "Because grass is a form of vegetation."
- informability: debugging the KB
  - "No the sky is not yellow. It's blue."
- extensibility: new relations
  - "Canaries are yellow."
- extensibility: new applications
  - returning a list of all the white things
  - painting pictures

What is knowledge?

Easier question: how do we talk about it?

We say "John knows that …" and fill the blank with a proposition

– can be true / false, right / wrong

Contrast: "John fears that …"

– same content, different attitude

Other forms of knowledge:

- know how, who, what, when, …
- sensorimotor: typing, riding a bicycle
- affective: deep understanding

Belief: not necessarily true and/or held for appropriate reasons

and weaker yet: "John suspects that …"

Here: no distinction

the main idea

taking the world to be one way and not another

What is representation?

Symbols standing for things in the world

✚ $\Longrightarrow$ first aid

♀ $\Longrightarrow$ women

"John" $\Longrightarrow$ John

"John loves Mary" $\Longrightarrow$ the proposition that John loves Mary

Knowledge representation:

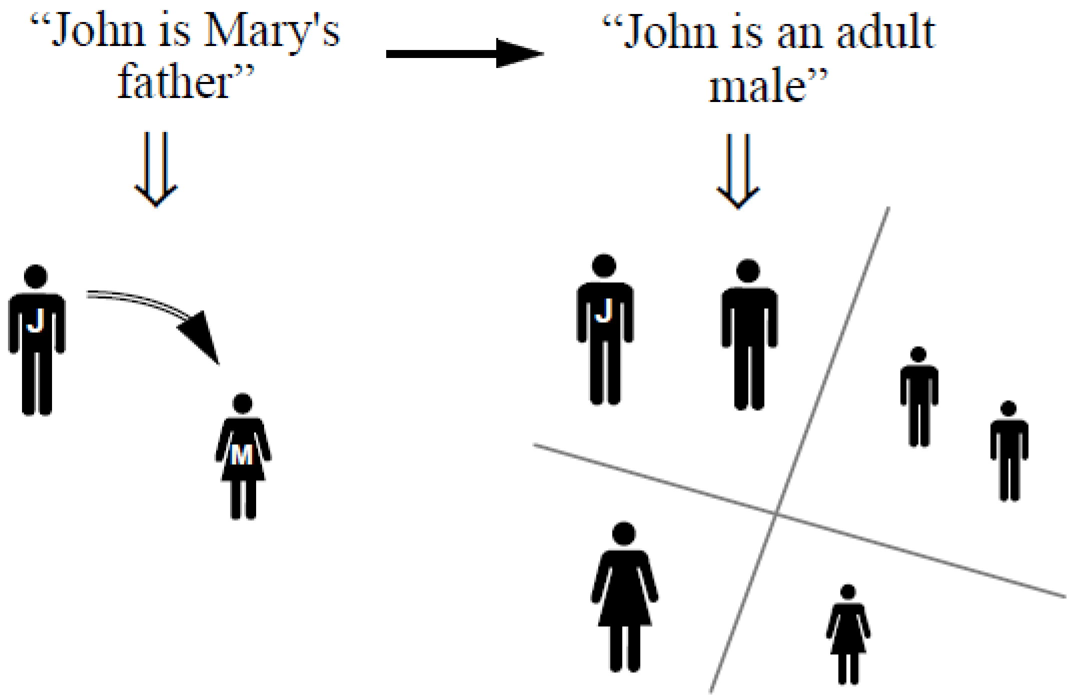
symbolic encoding of propositions believed
(by some agent)

What is reasoning?

Manipulation of symbols encoding propositions to produce representations of new propositions

Analogy: arithmetic

$$\begin{tikzcd}
\text{“1011”} \arrow[d, Rightarrow] & + & \text{“10”} \arrow[d, Rightarrow] \arrow[r] & \text{“1101”} \arrow[d, Rightarrow] \\
\text{eleven} & & \text{two} & \text{thirteen}
\end{tikzcd}$$



Knowledge representation

Understanding, designing, and implementing ways of representing information in computers so that programs (agents) can use this information

- to derive information that is implied by it,
- to converse with people in natural languages,
- to decide what to do next
- to plan future activities,
- to solve problems in areas that normally require human expertise.

Reasoning

- Deriving information that is implied by the information already present is a form of reasoning.
- Knowledge representation schemes are useless without the abilitv to reason with them.

- Represent knowledge about the world.
- Reason with that knowledge.

Why reasoning?

Want knowledge to affect action

not do action A if sentence P is in KB

but do action A if world believed in satisfies P

Difference:

P may not be *explicitly* represented

Need to apply what is known in general
to the particulars of a given situation

Example:

"Patient x is allergic to medication m."

"Anybody allergic to medication m is also
allergic to m'."

Is it OK to prescribe m' for x ?

Usually need more than just DB-style retrieval of facts in the KB

Sentences $P_1, P_2, \ldots, P_n$ <u>entail</u> sentence P iff the truth of P is implicit in the truth of $P_1, P_2, \ldots, P_n$.

If the world is such that it satisfies the P_i then it must also satisfy P.

Applies to a variety of languages (languages with truth theories)

Natural Language

- A **dime** is better than a **nickel.**
- A **nickel** is better than a **penny.**
- Therefore, a **dime** is better than a **penny.**

Knowledge

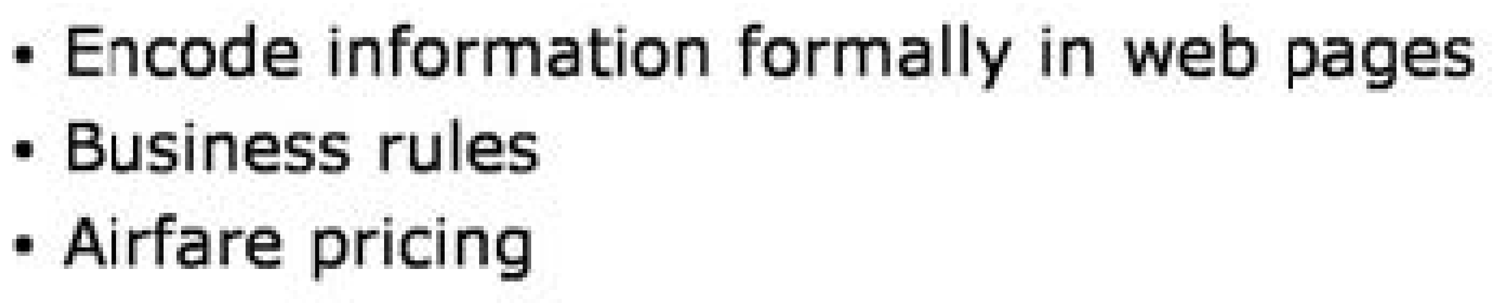
Reasoning

- A **penny** is better than a **nothing.**
- **Nothing** is better than **world peace.**
- Therefore, a **penny** is better than **world peace.**

Natural language is tricky!

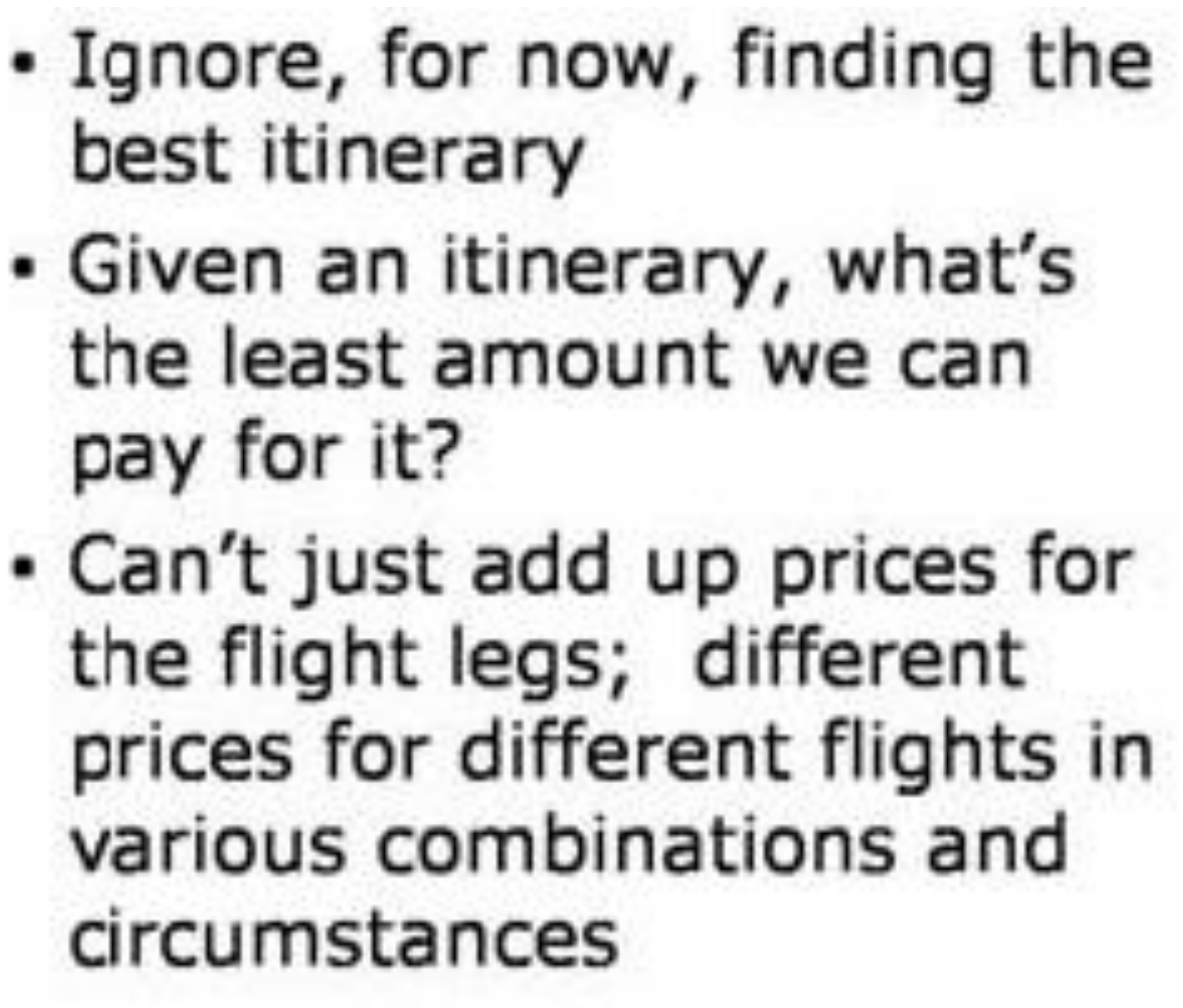
Use of logic removes ambiguity (similar to computer languages);
but also makes system less flexible. (Will study more flexible versions later.)

Logic in the real world



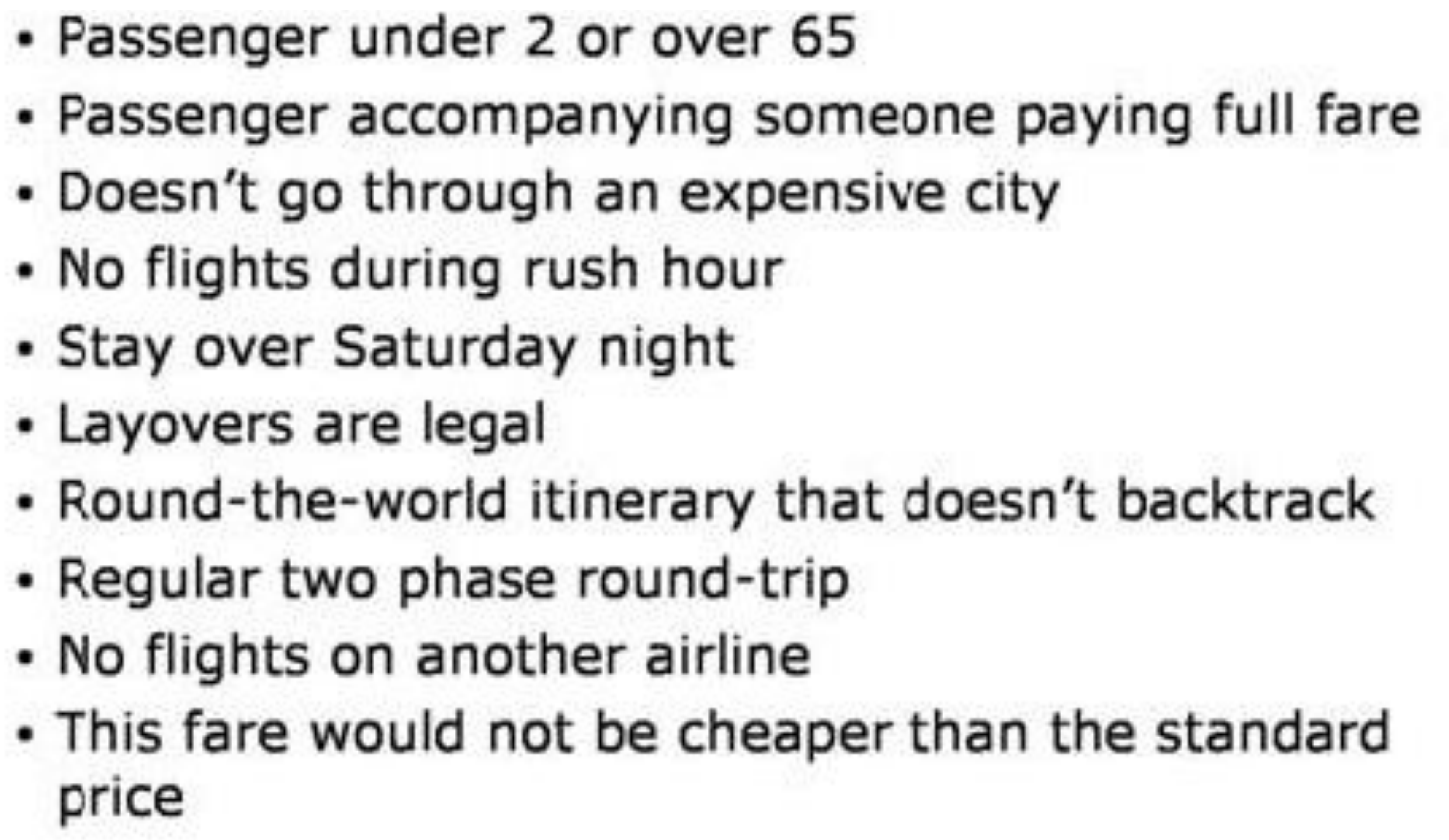
- Encode information formally in web pages
- Business rules
- Airfare pricing

Airfare Pricing



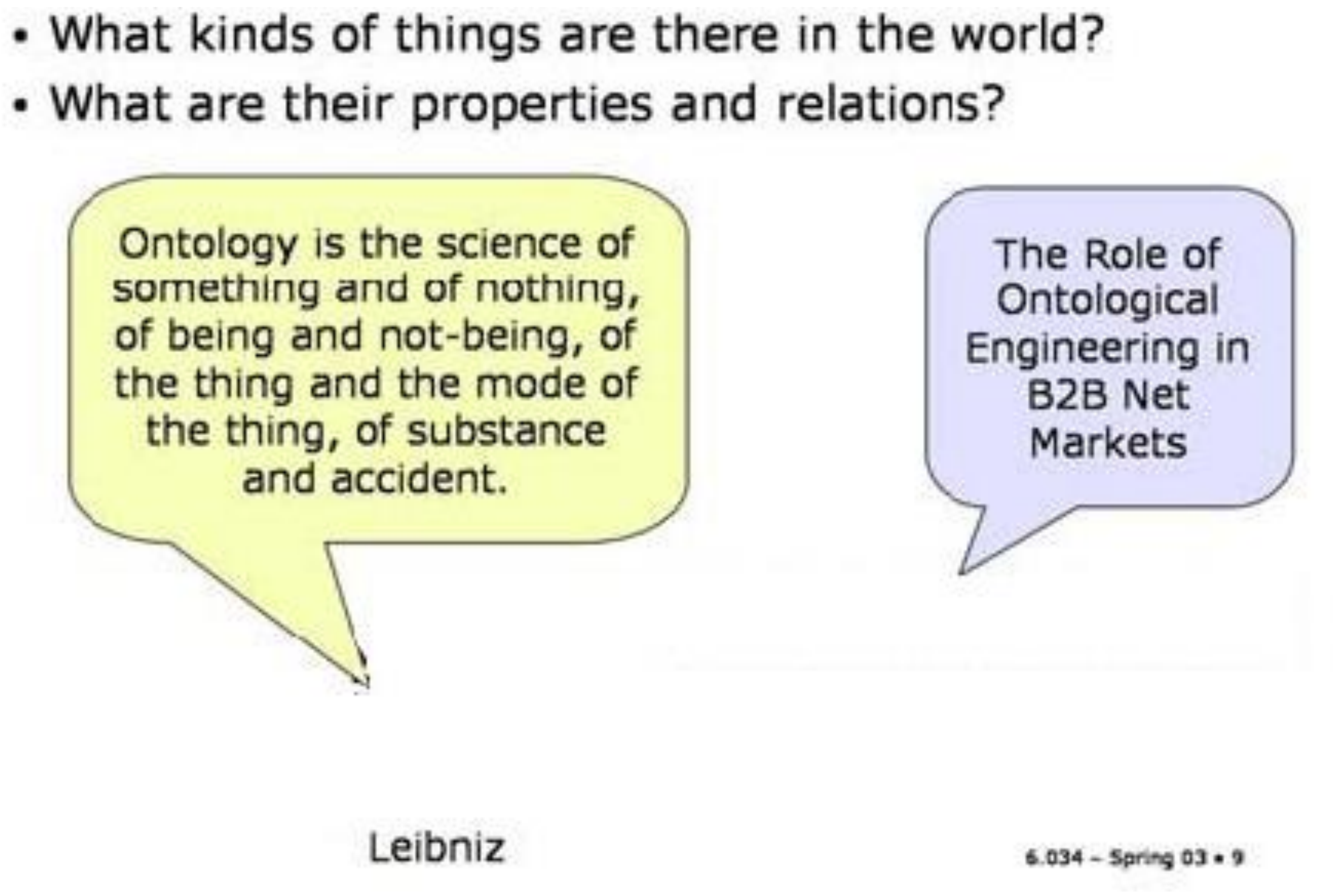
- Ignore, for now, finding the best itinerary
- Given an itinerary, what's the least amount we can pay for it?
- Can't just add up prices for the flight legs; different prices for different flights in various combinations and circumstances

Fare Restrictions



- Passenger under 2 or over 65
- Passenger accompanying someone paying full fare
- Doesn't go through an expensive city
- No flights during rush hour
- Stay over Saturday night
- Layovers are legal
- Round-the-world itinerary that doesn't backtrack
- Regular two phase round-trip
- No flights on another airline
- This fare would not be cheaper than the standard price

Ontology



- What kinds of things are there in the world?
- What are their properties and relations?

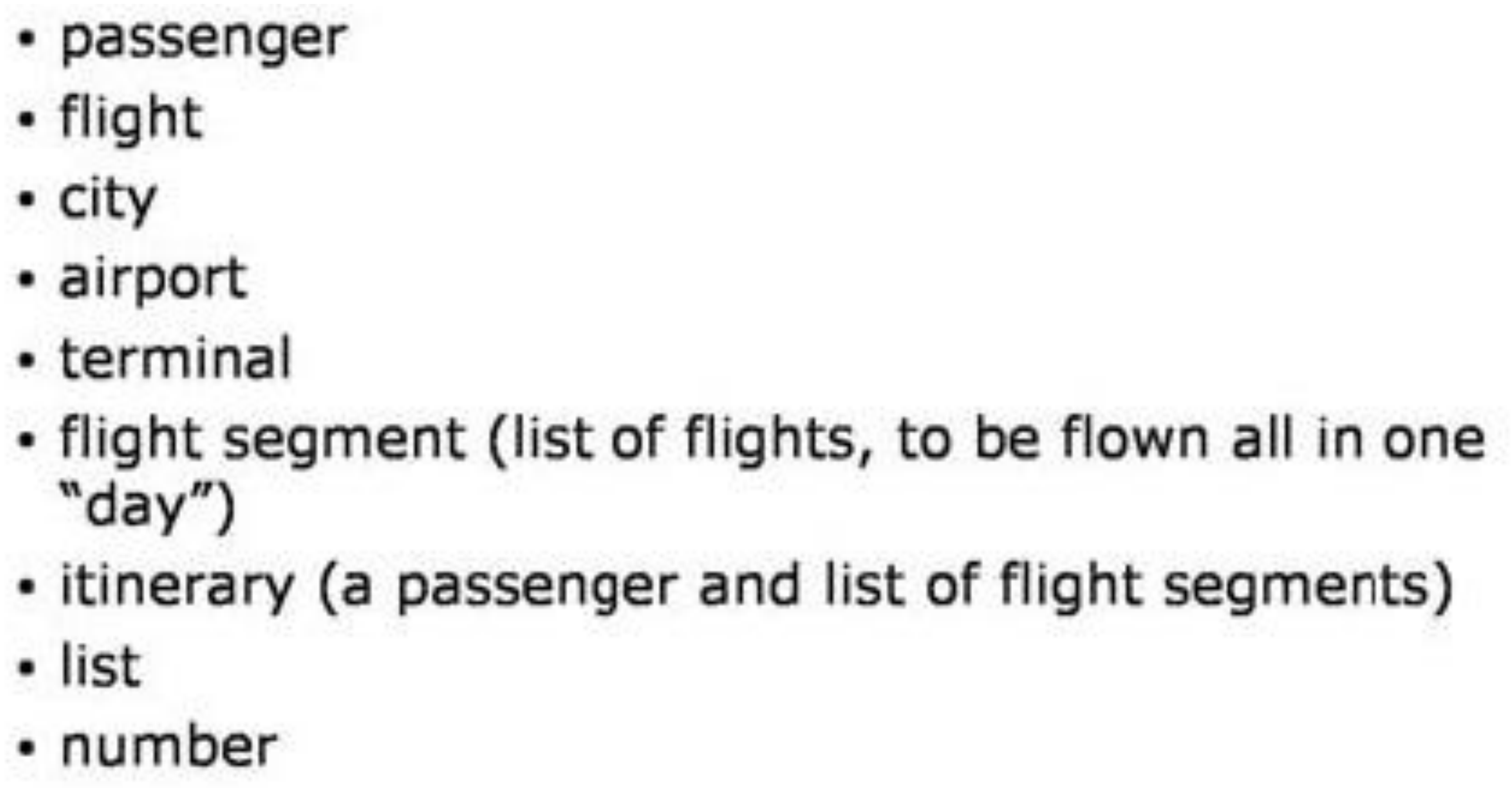
Ontology is the science of something and of nothing, of being and not-being, of the thing and the mode of the thing, of substance and accident.

The Role of Ontological Engineering in B2B Net Markets

Leibniz

6.034 – Spring 03 • 9

Airfare Domain Ontology



- passenger
- flight
- city
- airport
- terminal
- flight segment (list of flights, to be flown all in one "day")
- itinerary (a passenger and list of flight segments)
- list
- number

Representing Properties

- Object P is red
  - Red(P)
  - Color(P, Red)
  - color(P) = Red
  - Property(P, Color, Red)

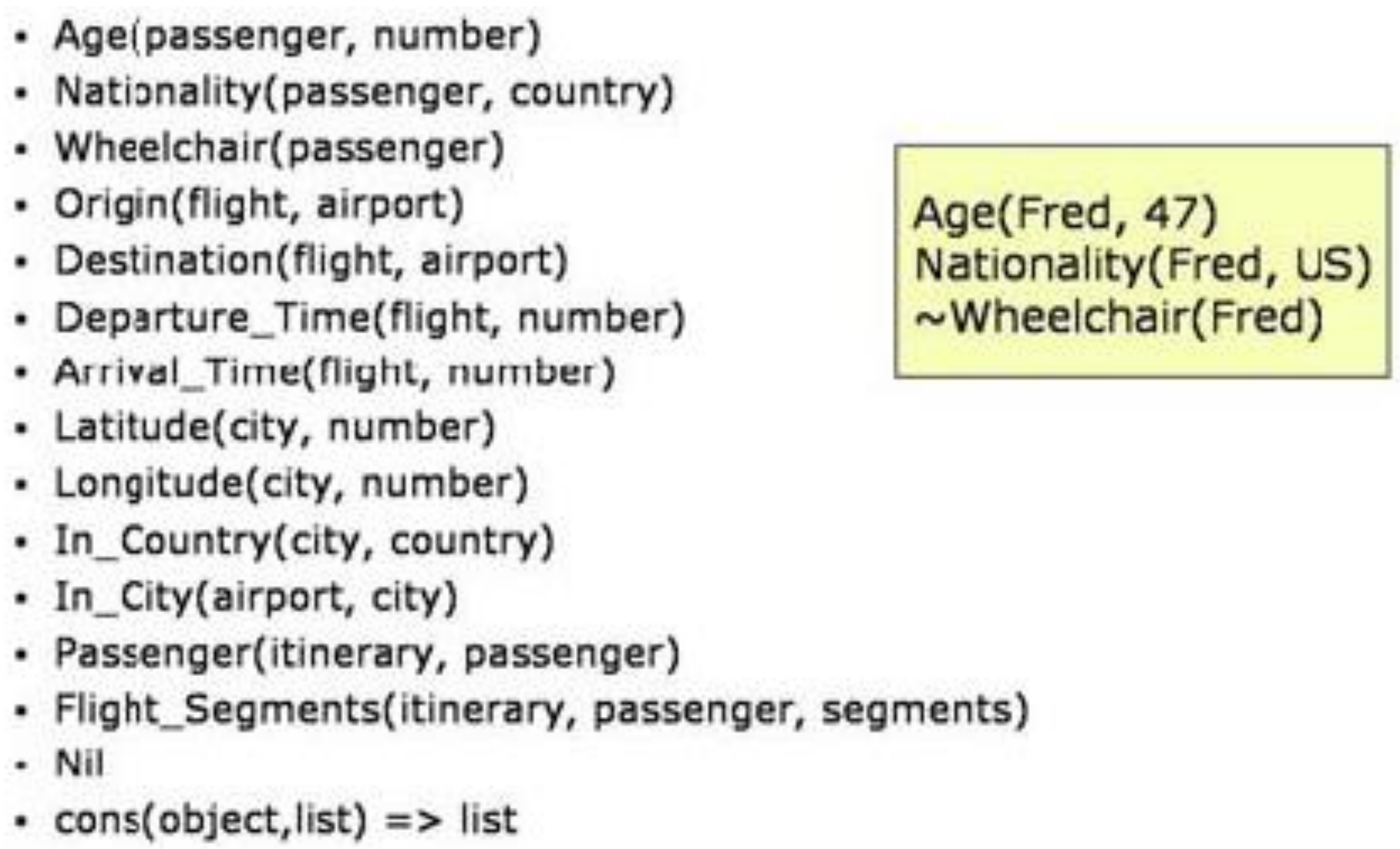
- All the blocks in stack S are the same color

$$\exists c.\ \forall b.\ \mathrm{In}(b, S) \rightarrow \mathrm{Color}(b, c)$$

- All the blocks in stack S have the same properties

$$\forall p.\ \exists v.\ \forall b.\ \mathrm{In}(b, S) \rightarrow \mathrm{Property}(b, p, v)$$

Basic Relations



- Age(passenger, number)
- Nationality(passenger, country)
- Wheelchair(passenger)
- Origin(flight, airport)
- Destination(flight, airport)
- Departure_Time(flight, number)
- Arrival_Time(flight, number)
- Latitude(city, number)
- Longitude(city, number)
- In_Country(city, country)
- In_City(airport, city)
- Passenger(itinerary, passenger)
- Flight_Segments(itinerary, passenger, segments)
- Nil
- cons(object,list) => list

Age(Fred, 47)
Nationality(Fred, US)
~Wheelchair(Fred)

Defined Relations

- Define complex relations in terms of basic ones
- Like using subroutines

$$\forall i.\ P(i) \wedge Q(i) \rightarrow \text{Qualifies37}(i)$$

- Implication rather than equivalence
  - easier to specify definitions in pieces

$$\forall i.\ R(i) \wedge S(i) \rightarrow \text{Qualifies37}(i)$$

  - can't use the other direction

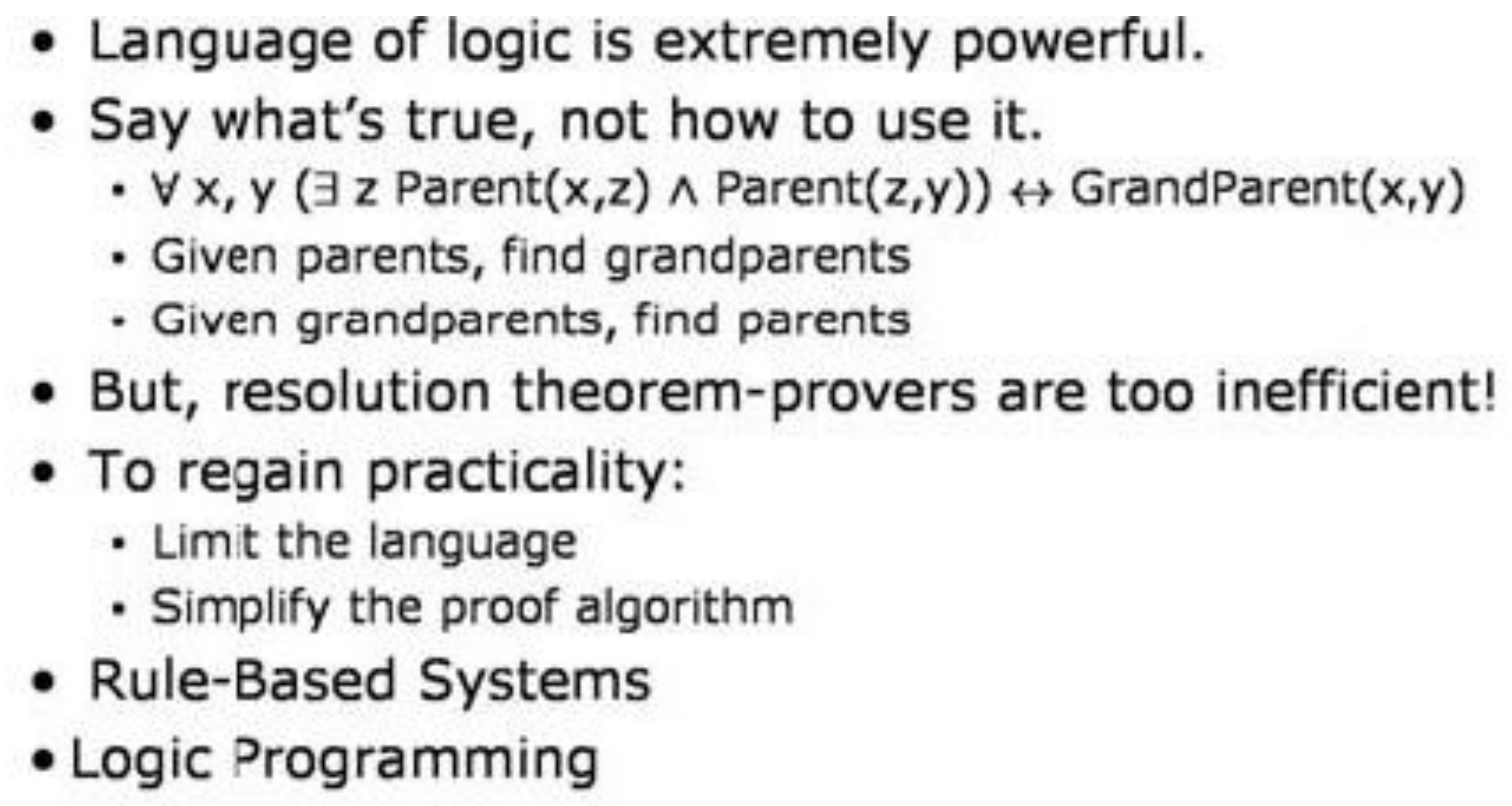
$$\text{Qualifies37}(i) \rightarrow ?$$

  - if you need it, write the equivalence

$$\forall i.\ (P(i) \wedge Q(i)) \vee (R(i) \wedge S(i)) \leftrightarrow \text{Qualifies37}(i)$$

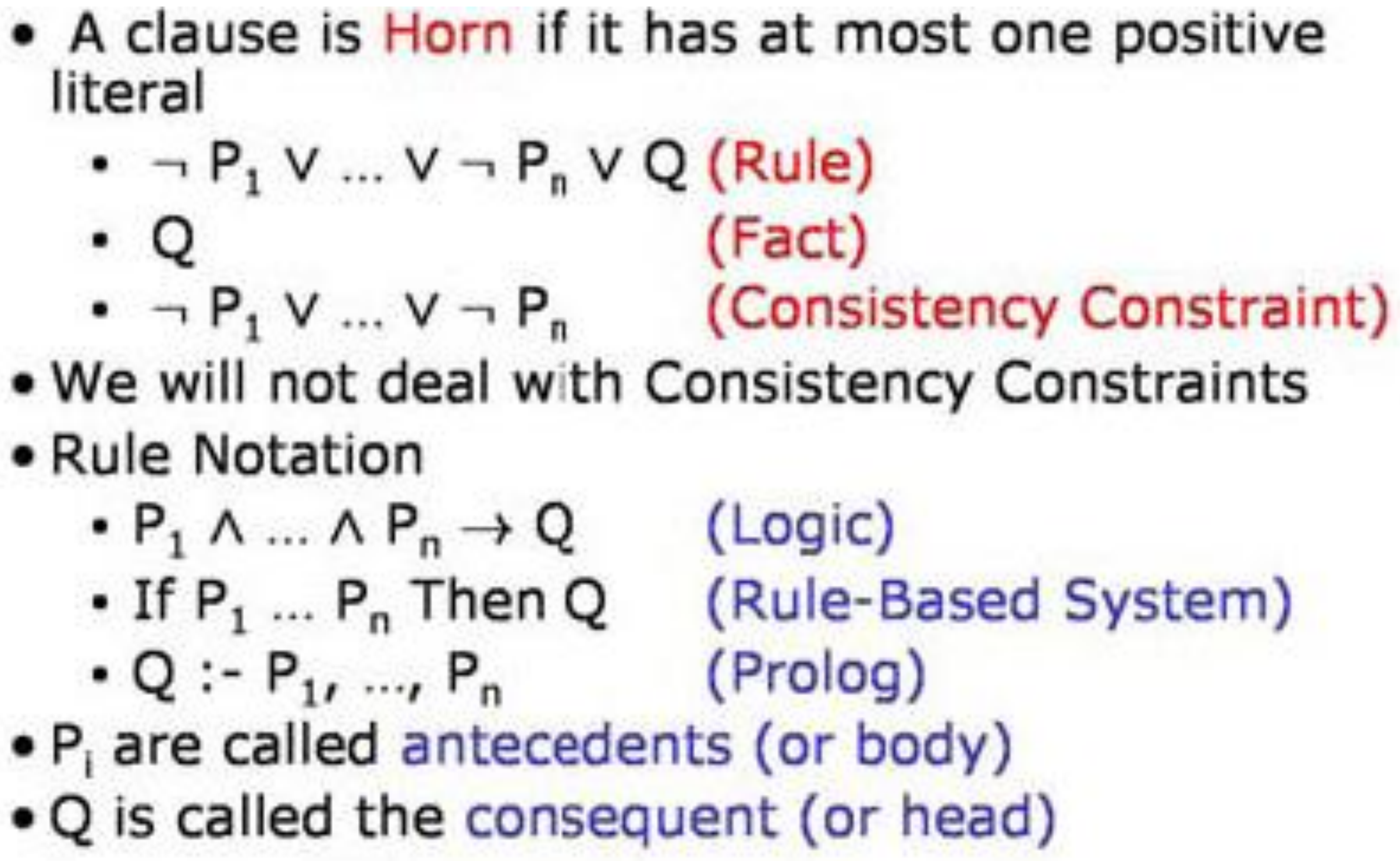
$$\forall i, a, p.\ \text{Passenger}(i, p) \wedge \text{Age}(p, a) \wedge a < 2 \rightarrow \text{InfantFare}(i)$$

Rules and Logic Programming



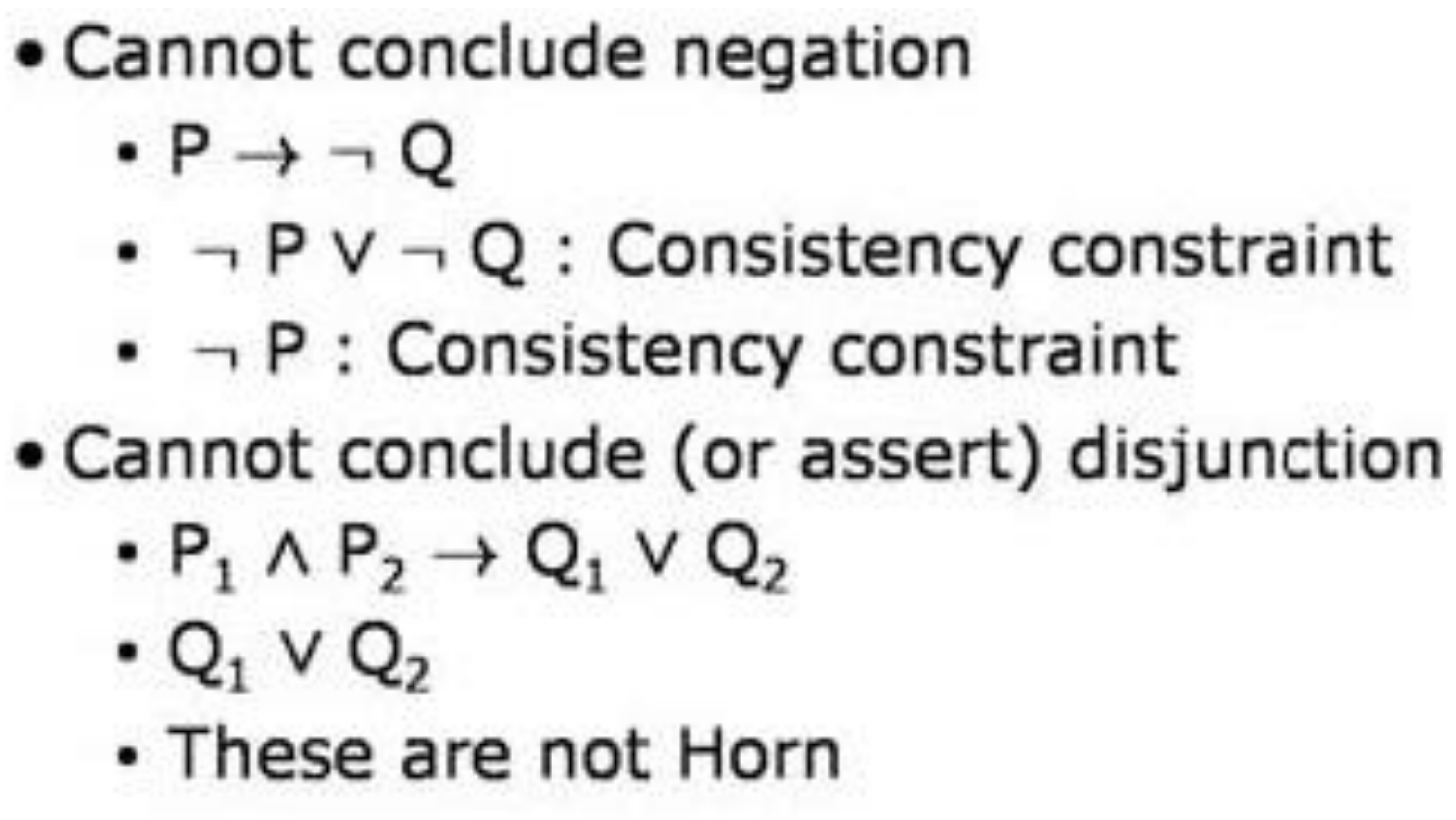
- Language of logic is extremely powerful.
- Say what's true, not how to use it.
  - $\forall$ x, y ($\exists$ z Parent(x,z) $\wedge$ Parent(z,y)) $\leftrightarrow$ GrandParent(x,y)
  - Given parents, find grandparents
  - Given grandparents, find parents
- But, resolution theorem-provers are too inefficient!
- To regain practicality:
  - Limit the language
  - Simplify the proof algorithm
- Rule-Based Systems
- Logic Programming

Horn Clauses



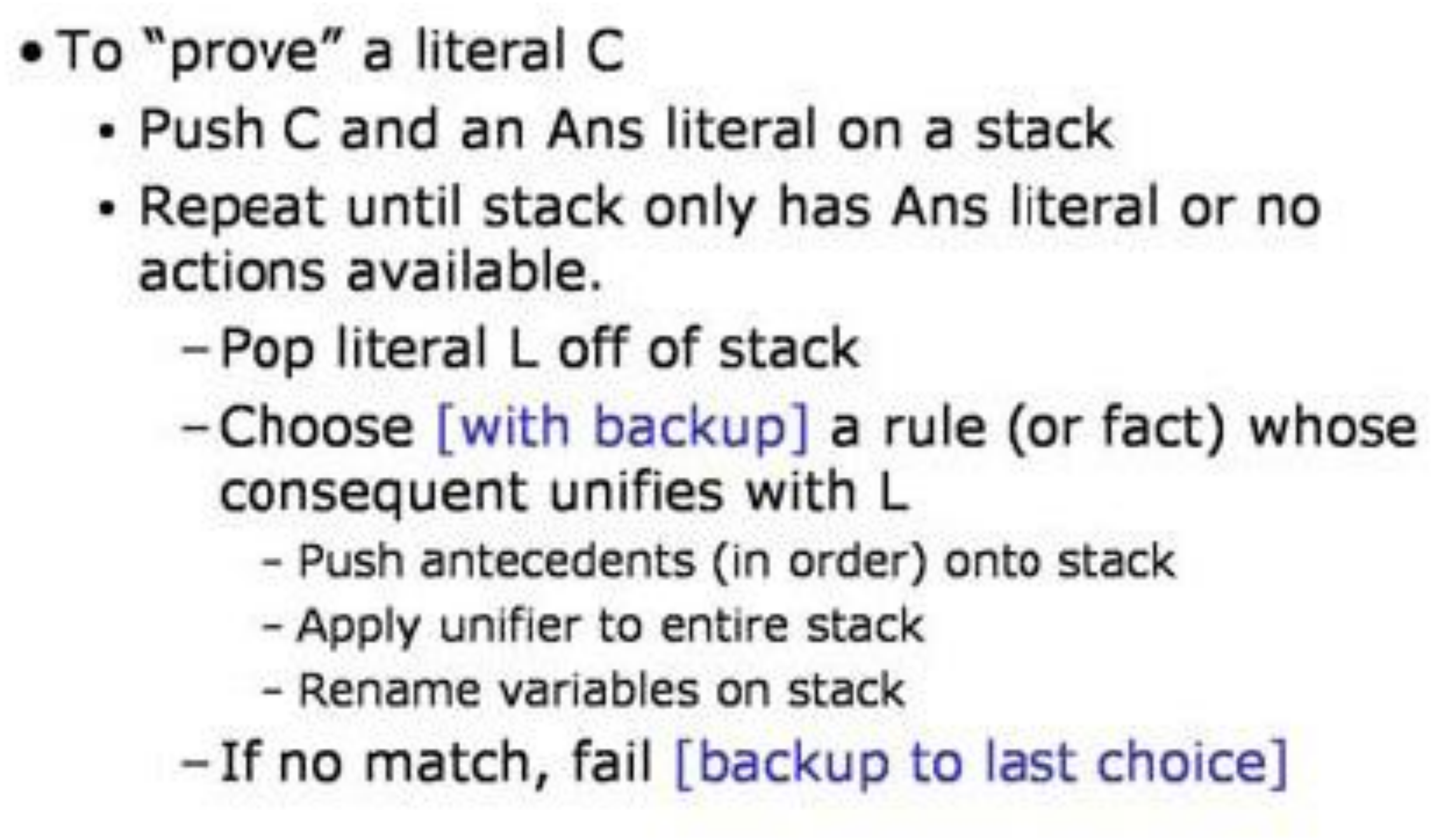
- A clause is Horn if it has at most one positive literal
  - $\neg P_1 \vee \ldots \vee \neg P_n \vee Q$ (Rule)
  - Q (Fact)
  - $\neg P_1 \vee \ldots \vee \neg P_n$ (Consistency Constraint)
- We will not deal with Consistency Constraints
- Rule Notation
  - $P_1 \wedge \ldots \wedge P_n \rightarrow Q$ (Logic)
  - If $P_1 \ldots P_n$ Then Q (Rule-Based System)
  - Q :- $P_1, \ldots, P_n$ (Prolog)
- P_i are called antecedents (or body)
- Q is called the consequent (or head)

Limitations



- Cannot conclude negation
  - $P \rightarrow \neg Q$
  - $\neg P \vee \neg Q$: Consistency constraint
  - $\neg P$: Consistency constraint
- Cannot conclude (or assert) disjunction
  - $P_1 \wedge P_2 \rightarrow Q_1 \vee Q_2$
  - $Q_1 \vee Q_2$
  - These are not Horn

Inference: Backchaining

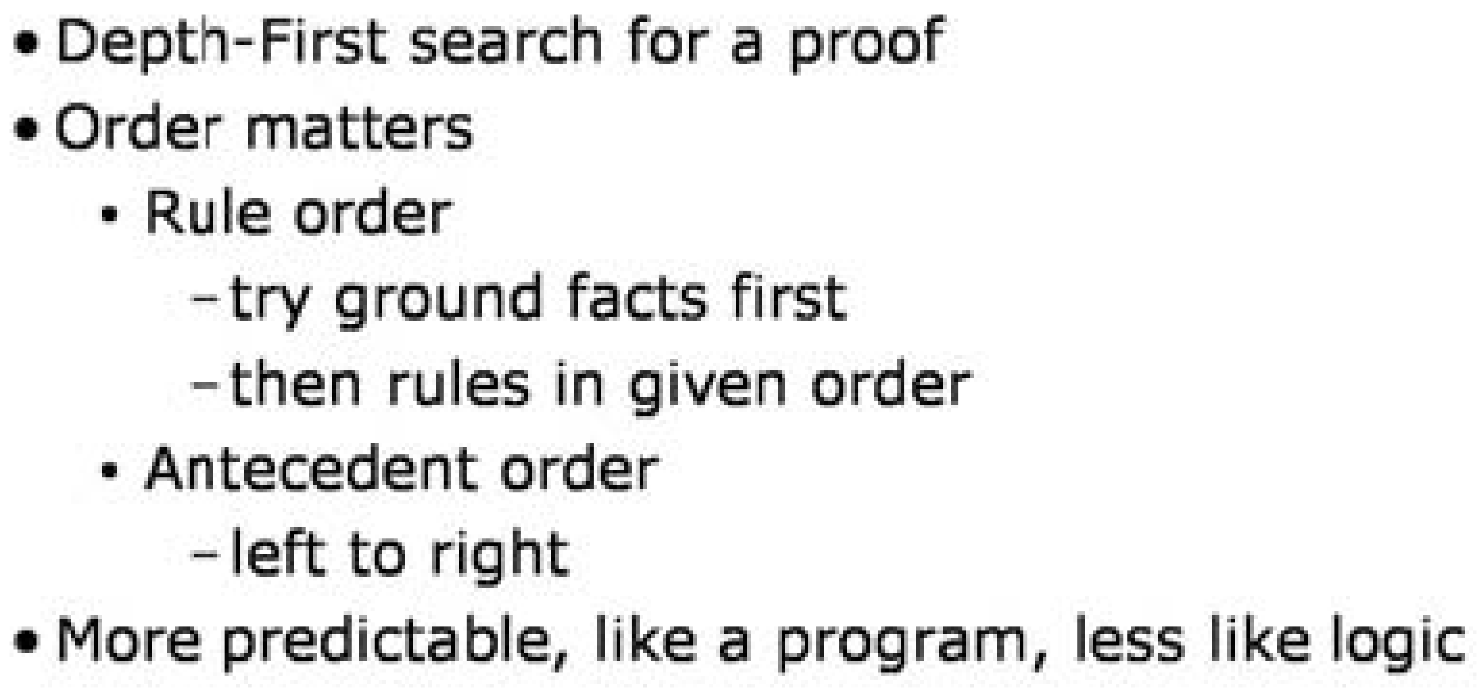


- To "prove" a literal C
  - Push C and an Ans literal on a stack
  - Repeat until stack only has Ans literal or no actions available.
    - Pop literal L off of stack
    - Choose [with backup] a rule (or fact) whose consequent unifies with L
      - Push antecedents (in order) onto stack
      - Apply unifier to entire stack
      - Rename variables on stack
    - If no match, fail [backup to last choice]

Backchaining and Resolution

- Backchaining is just resolution
- To prove C (propositional case)
  - Negate C $\Rightarrow \neg$ C
  - Find rule $\neg P_1 \lor \ldots \lor \neg P_n \lor C$
  - Resolve to get $\neg P_1 \lor \ldots \lor \neg P_n$
  - Repeat for each negative literal
- First order case introduces unification but otherwise the same.

Proof Strategy



- Depth-First search for a proof
- Order matters
  - Rule order
    - try ground facts first
    - then rules in given order
  - Antecedent order
    - left to right
- More predictable, like a program, less like logic

Example

```
1. Father(A,B)        ; ground fact
2. Mother(B,C)        ; ground fact
3. GrandP(?x,?z):- Parent(?x,?y),Parent(?y,?z)
4. Parent(?x,?y):- Father(?x,?y)
5. Parent(?x,?y):- Mother(?x,?y)
```

[illegible]

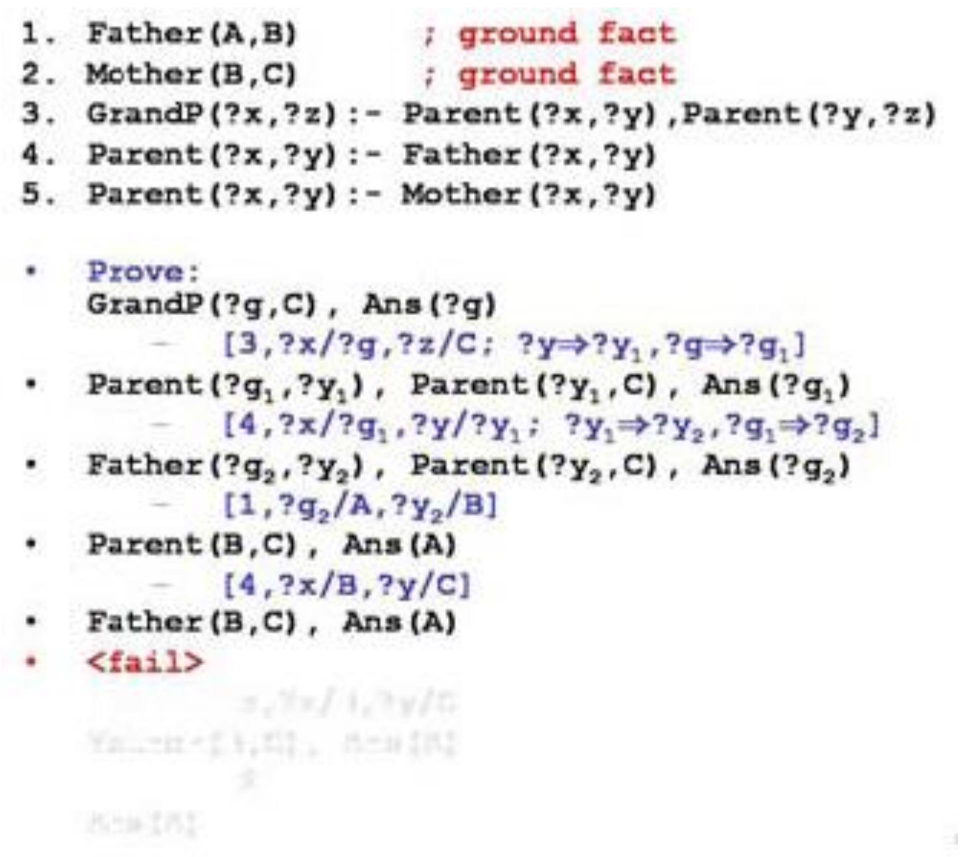
Example

```
1.  Father(A,B)          ; ground fact
2.  Mother(B,C)          ; ground fact
3.  GrandP(?x,?z):- Parent(?x,?y),Parent(?y,?z)
4.  Parent(?x,?y):- Father(?x,?y)
5.  Parent(?x,?y):- Mother(?x,?y)
```

- **Prove:**

```
GrandP(?g,C), Ans(?g)
           3,?x/?g,?z/C; ?y->?y1,?g->?g1
Parent(?g1,?y1), Parent(?y1,C), Ans(?g1)
           4,?x/?g1,?y/?y1; ?y1->?y2,?g1->?g2
Father(?g2,?y2), Parent(?y2,C), Ans(?g2)
           1,?g2/A,?y2/B
Parent(B,C), Ans(A)
           4,?x/B,?y/C
Father(B,C), Ans(A)
<fail>
           5,?x/B,?y/C
Mother(B,C), Ans(A)
           2
Ans(A)
```

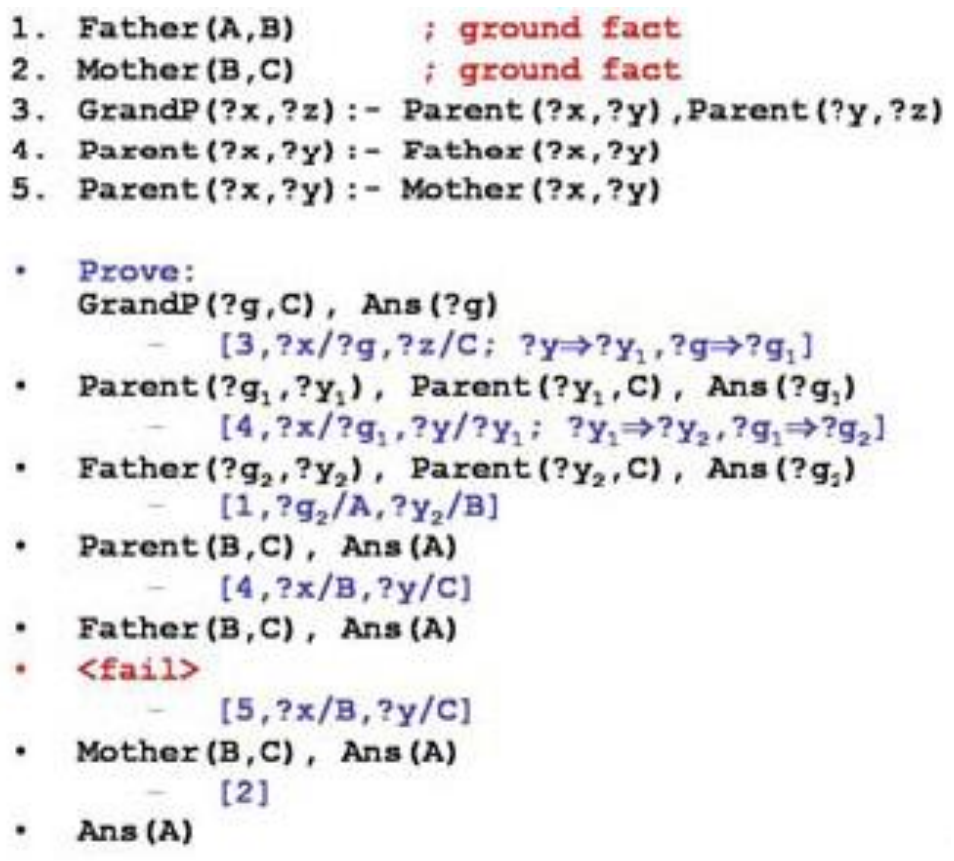
Example



1. Father(A,B)   ; ground fact
2. Mother(B,C)   ; ground fact
3. GrandP(?x,?z) :- Parent(?x,?y),Parent(?y,?z)
4. Parent(?x,?y) :- Father(?x,?y)
5. Parent(?x,?y) :- Mother(?x,?y)

- Prove:
  GrandP(?g,C), Ans(?g)
  - [3,?x/?g,?z/C; $?y \Rightarrow ?y_1$, $?g \Rightarrow ?g_1$]
- Parent($?g_1$,$?y_1$), Parent($?y_1$,C), Ans($?g_1$)
  - [4,?x/$?g_1$,?y/$?y_1$; $?y_1 \Rightarrow ?y_2$, $?g_1 \Rightarrow ?g_2$]
- Father($?g_2$,$?y_2$), Parent($?y_2$,C), Ans($?g_2$)
  - [1,$?g_2$/A,$?y_2$/B]
- Parent(B,C), Ans(A)
  - [4,?x/B,?y/C]
- Father(B,C), Ans(A)
- <fail>

Example



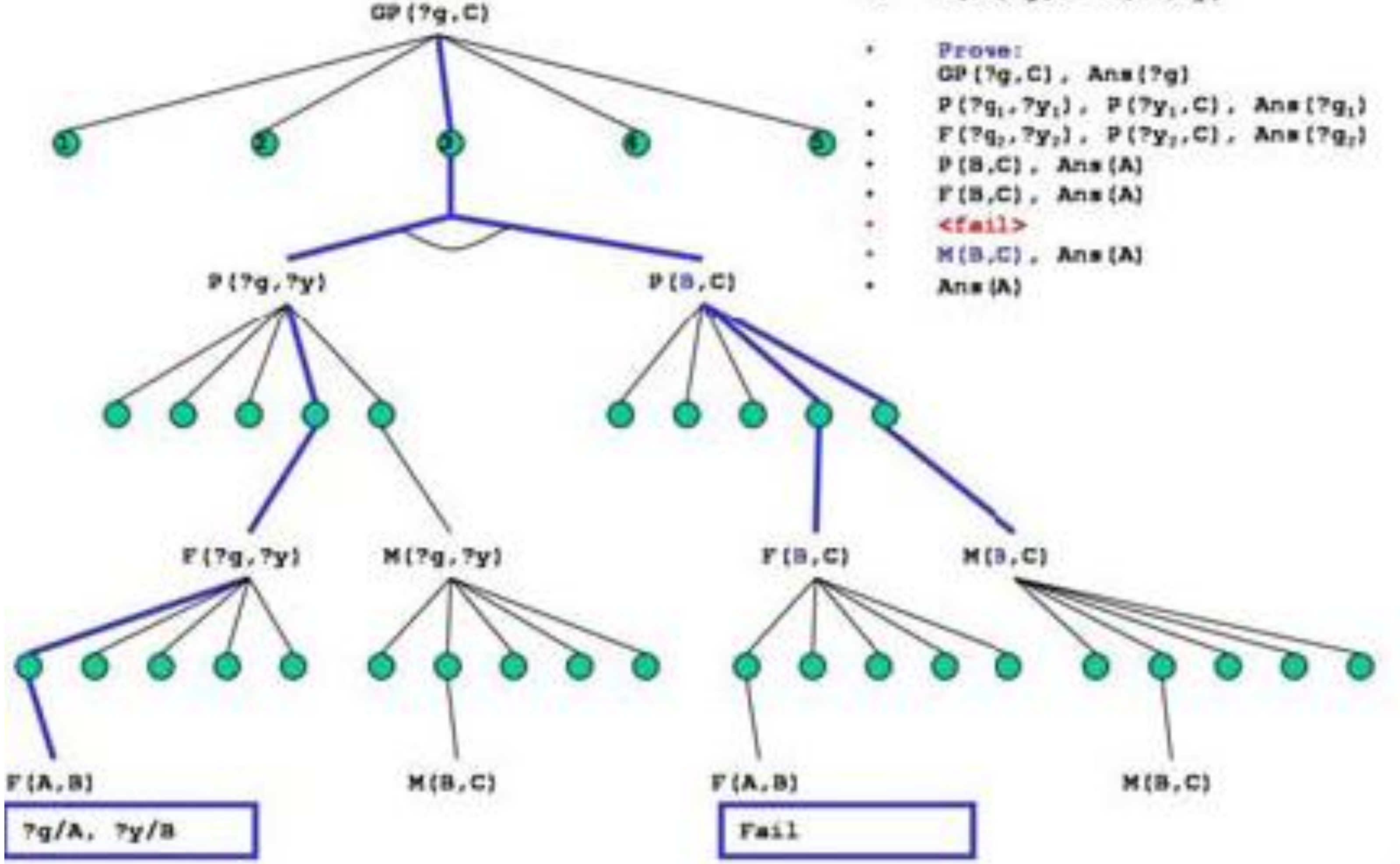
1. Father(A,B) ; ground fact
2. Mother(B,C) ; ground fact
3. GrandP(?x,?z) :- Parent(?x,?y),Parent(?y,?z)
4. Parent(?x,?y) :- Father(?x,?y)
5. Parent(?x,?y) :- Mother(?x,?y)

- Prove:
  GrandP(?g,C), Ans(?g)
  - [3,?x/?g,?z/C; ?y⇒?y_1,?g⇒?g_1]
- Parent(?g_1,?y_1), Parent(?y_1,C), Ans(?g_1)
  - [4,?x/?g_1,?y/?y_1; ?y_1⇒?y_2,?g_1⇒?g_2]
- Father(?g_2,?y_2), Parent(?y_2,C), Ans(?g_2)
  - [1,?g_2/A,?y_2/B]
- Parent(B,C), Ans(A)
  - [4,?x/B,?y/C]
- Father(B,C), Ans(A)
- <fail>
  - [5,?x/B,?y/C]
- Mother(B,C), Ans(A)
  - [2]
- Ans(A)

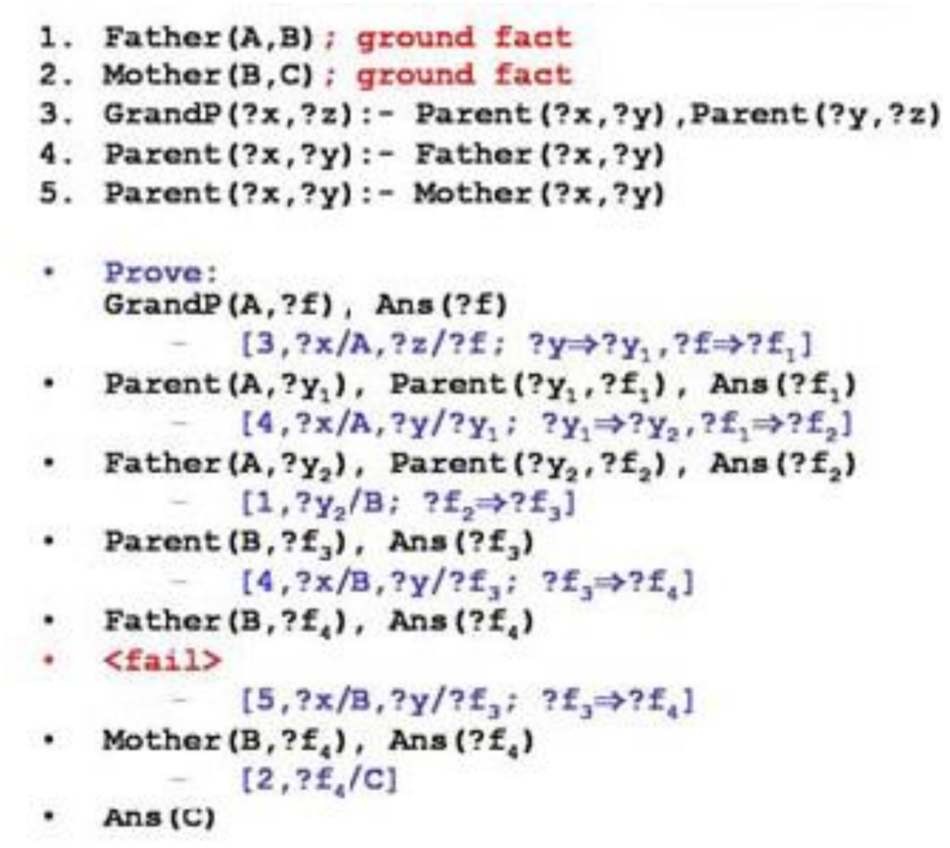
Proof Tree

1. F(A,B)
2. M(B,C)
3. GP(?x,?z) :- P(?x,?y),P(?y,?z)
4. P(?x,?y) :- F(?x,?y)
5. P(?x,?y) :- M(?x,?y)

- Prove:
  GP(?g,C), Ans(?g)
- P(?g$_1$,?y$_1$), P(?y$_1$,C), Ans(?g$_1$)
- F(?g$_2$,?y$_2$), P(?y$_2$,C), Ans(?g$_2$)
- P(B,C), Ans(A)
- F(B,C), Ans(A)
- <fail>
- M(B,C), Ans(A)
- Ans(A)



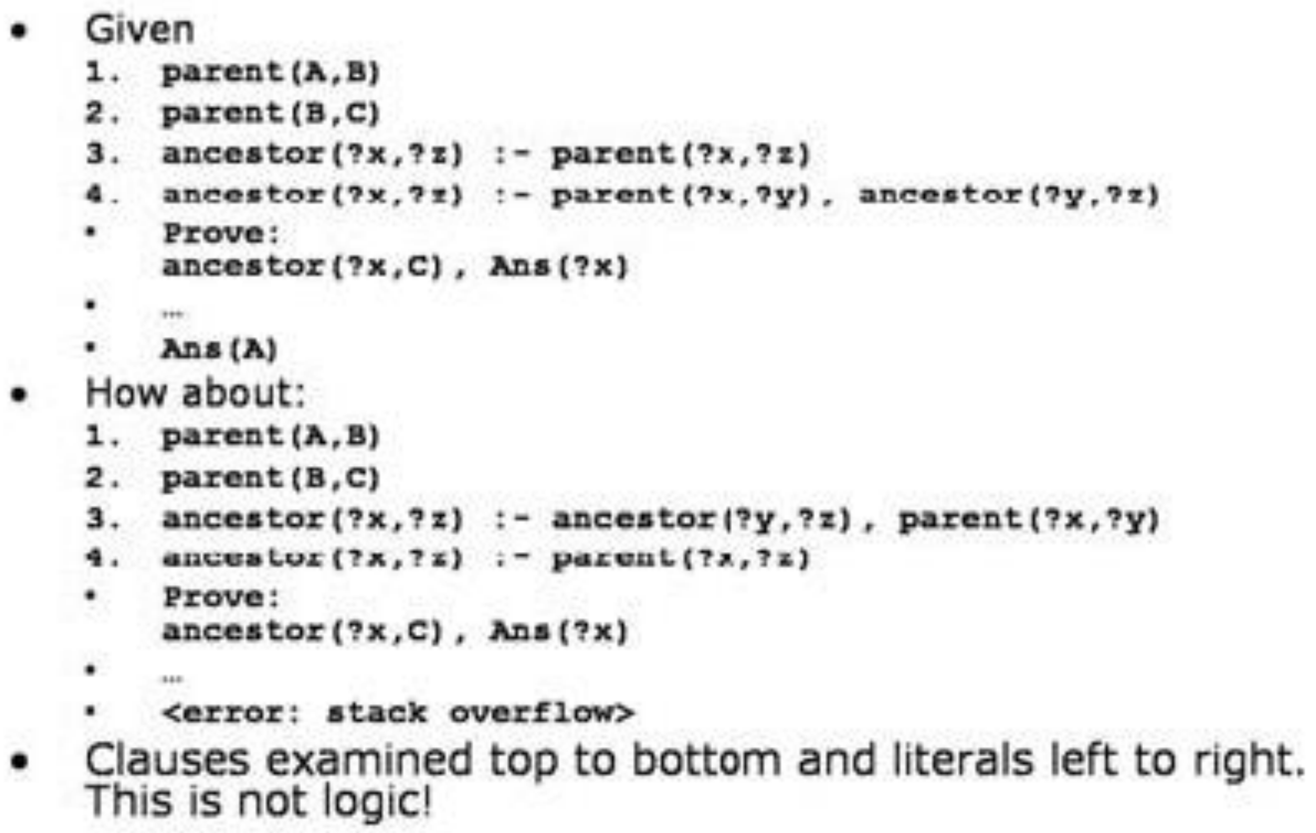
Relations not Functions



1. Father(A,B) ; ground fact
2. Mother(B,C) ; ground fact
3. GrandP(?x,?z):- Parent(?x,?y),Parent(?y,?z)
4. Parent(?x,?y):- Father(?x,?y)
5. Parent(?x,?y):- Mother(?x,?y)

- Prove:
  GrandP(A,?f), Ans(?f)
  - [3,?x/A,?z/?f; $?y \Rightarrow ?y_1, ?f \Rightarrow ?f_1$]
- Parent(A,$?y_1$), Parent($?y_1$,$?f_1$), Ans($?f_1$)
  - [4,?x/A,?y/$?y_1$; $?y_1 \Rightarrow ?y_2, ?f_1 \Rightarrow ?f_2$]
- Father(A,$?y_2$), Parent($?y_2$,$?f_2$), Ans($?f_2$)
  - [1,$?y_2$/B; $?f_2 \Rightarrow ?f_3$]
- Parent(B,$?f_3$), Ans($?f_3$)
  - [4,?x/B,?y/$?f_3$; $?f_3 \Rightarrow ?f_4$]
- Father(B,$?f_4$), Ans($?f_4$)
- <fail>
  - [5,?x/B,?y/$?f_3$; $?f_3 \Rightarrow ?f_4$]
- Mother(B,$?f_4$), Ans($?f_4$)
  - [2,$?f_4$/C]
- Ans(C)

Order Revisited



- Given

```
1.  parent(A,B)
2.  parent(B,C)
3.  ancestor(?x,?z) :- parent(?x,?z)
4.  ancestor(?x,?z) :- parent(?x,?y), ancestor(?y,?z)
```

  - Prove:
    `ancestor(?x,C), Ans(?x)`
  - …
  - `Ans(A)`

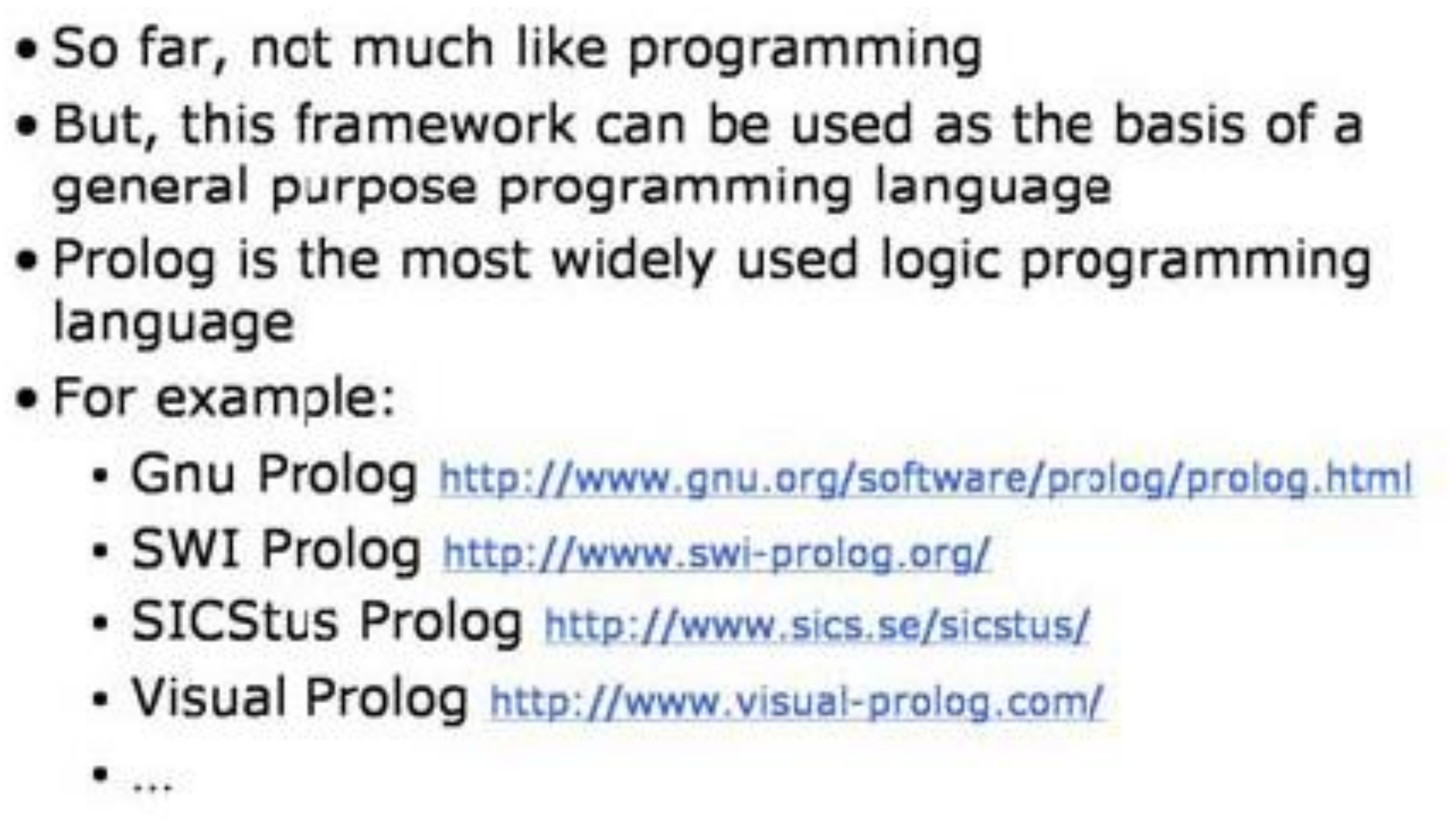
- How about:

```
1.  parent(A,B)
2.  parent(B,C)
3.  ancestor(?x,?z) :- ancestor(?y,?z), parent(?x,?y)
4.  ancestor(?x,?z) :- parent(?x,?z)
```

  - Prove:
    `ancestor(?x,C), Ans(?x)`
  - …
  - `<error: stack overflow>`

- Clauses examined top to bottom and literals left to right. This is not logic!

Logic Programming



- So far, not much like programming
- But, this framework can be used as the basis of a general purpose programming language
- Prolog is the most widely used logic programming language
- For example:
  - Gnu Prolog http://www.gnu.org/software/prolog/prolog.html
  - SWI Prolog http://www.swi-prolog.org/
  - SICStus Prolog http://www.sics.se/sicstus/
  - Visual Prolog http://www.visual-prolog.com/
  - ...