

CPSC 322: Introduction to Artificial Intelligence


Logics: Top-down proofs and Datalog

Textbook reference: [\[5.3.2\]](#)

Instructor: Varada Kolhatkar
University of British Columbia

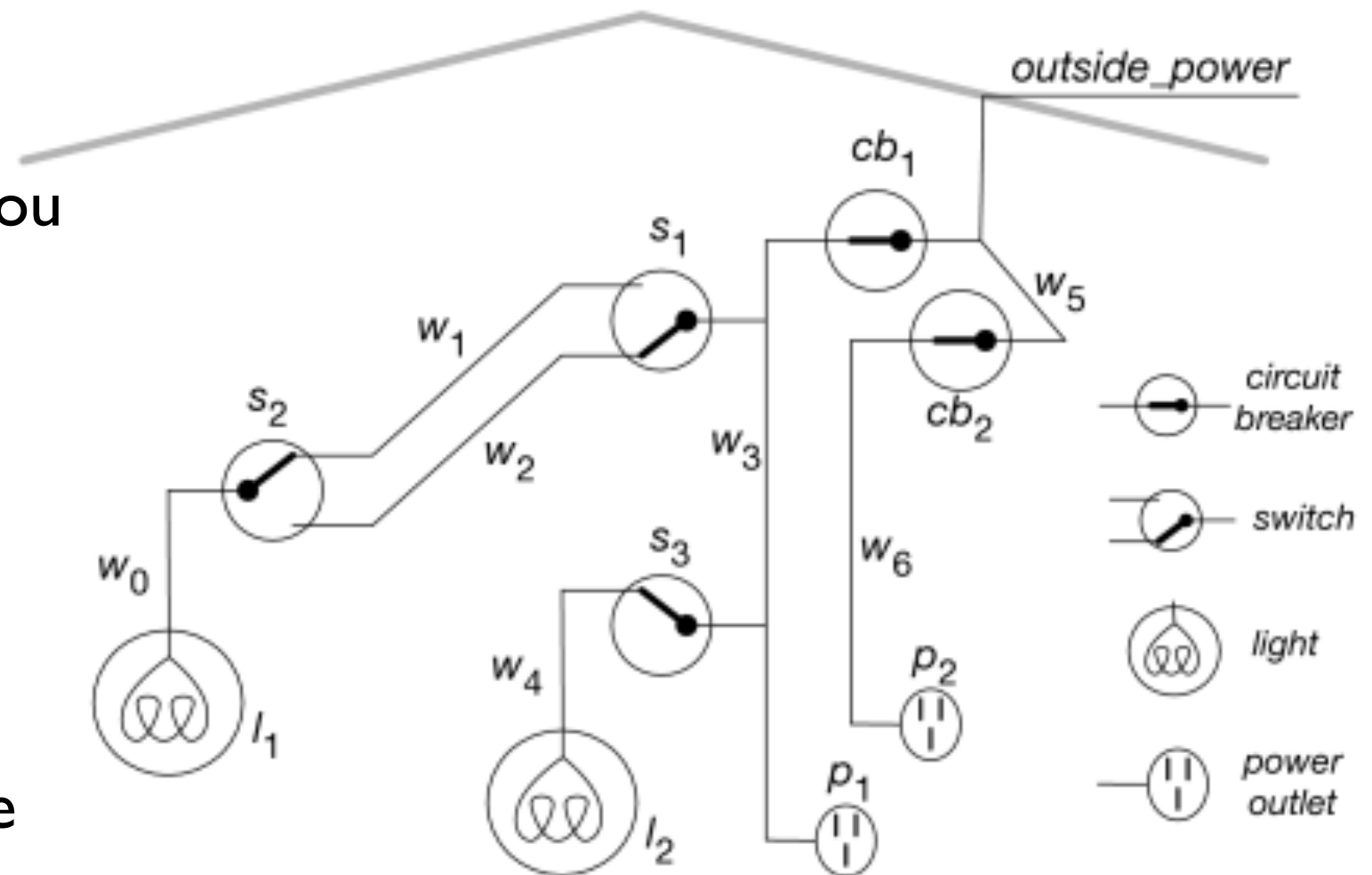
Credit: These slides are adapted from the slides of the previous offerings of the course. Thanks to all instructors for creating and improving the teaching material and making it available!

Lecture outline

- Class activity (electric environment) (~10 mins) 
- Top-down proofs (~30 mins)
- Datalog (~25 mins)
- Summary and wrap-up (~5 mins)

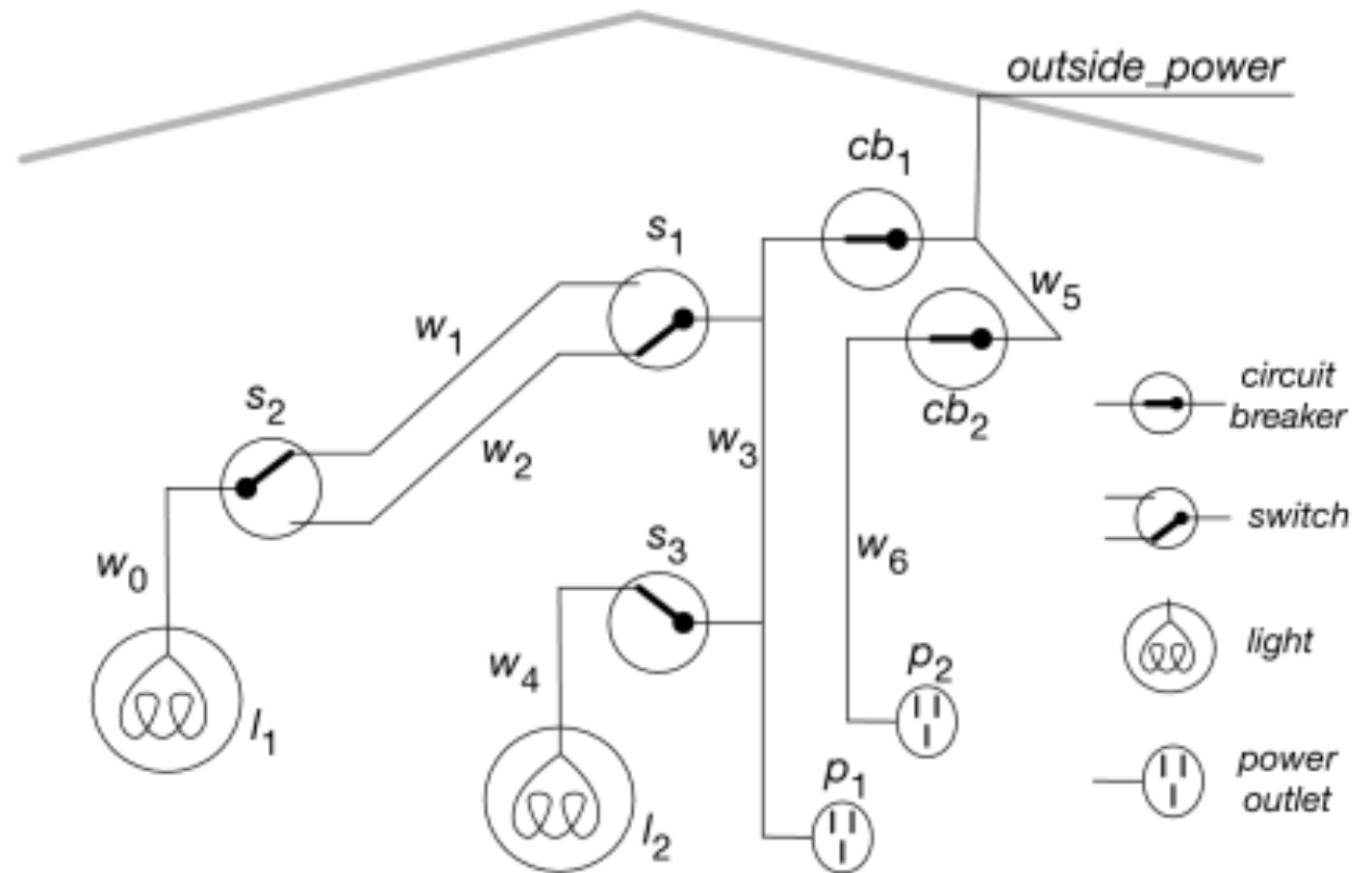
Example: Electric environment

1. Begin with a task domain.
2. Distinguish those things you want to talk about (the ontology)
3. Choose symbols in the computer to denote propositions
4. Tell the system knowledge about the domain
5. Ask the system whether new statements about the domain are true or false



Class activity: Let's define relevant propositions

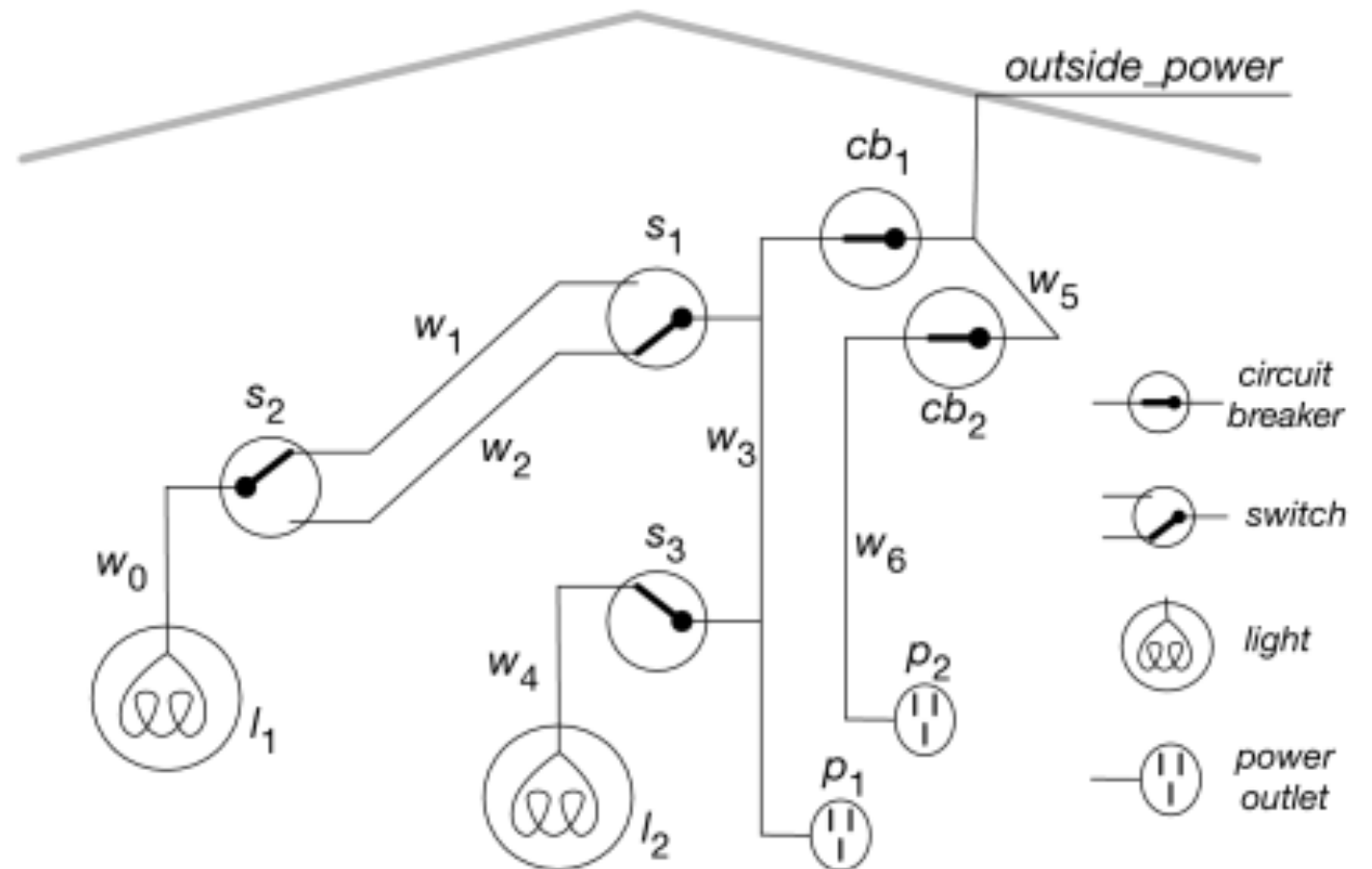
- For each wire w
- For each circuit breaker cb
- For each switch s
- For each light l
- For each outlet p



How many interpretations?

Knowledge about how domain works

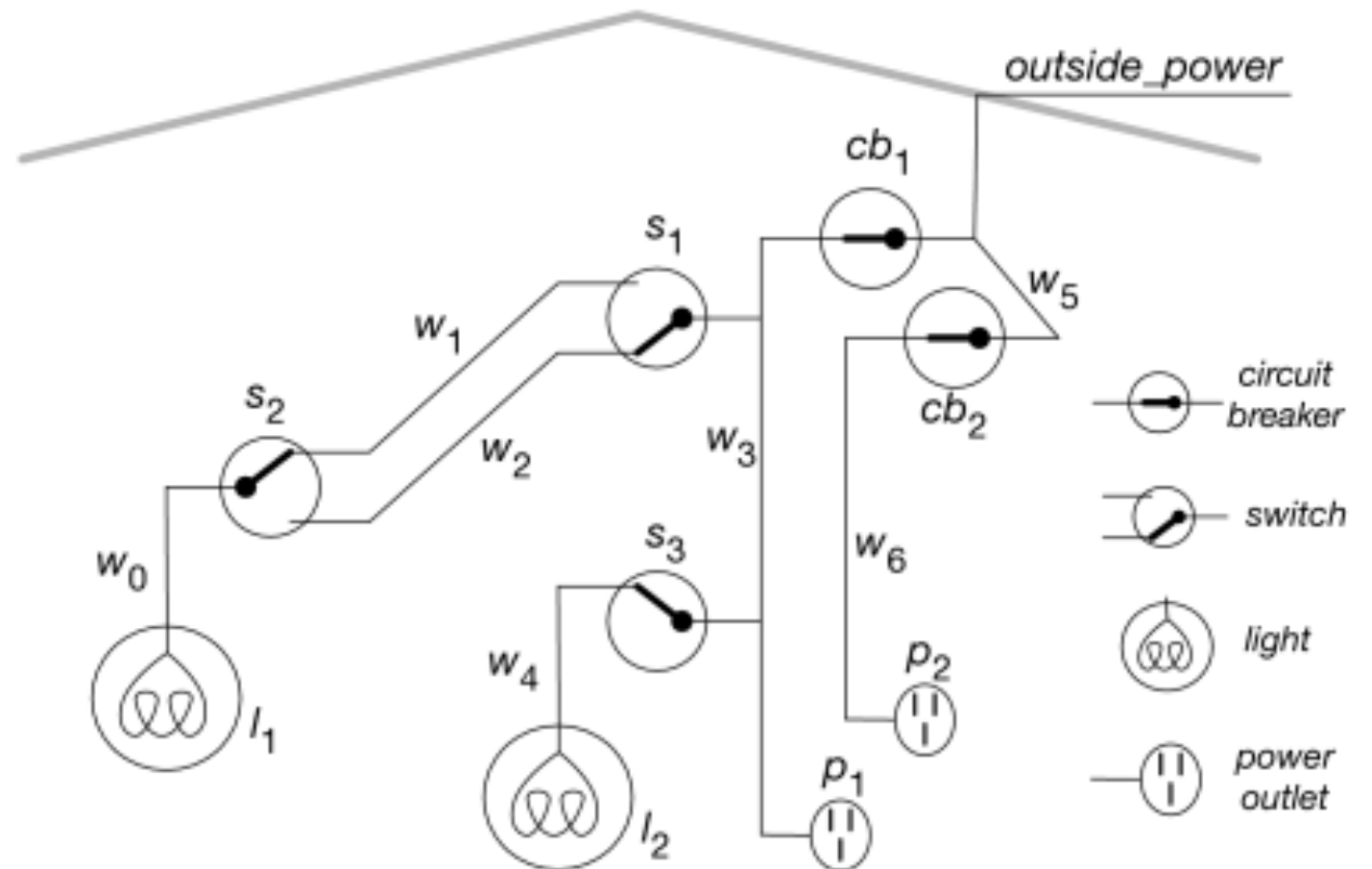
$live_{l_1} \leftarrow$
 $live_{w_0} \leftarrow$
 $live_{w_0} \leftarrow$
 $live_{w_1} \leftarrow$



How many interpretations?

Knowledge about how domain works

$live_l_1 \leftarrow live_w_0$
 $live_w_0 \leftarrow live_w_1 \wedge up_s_2$
 $live_w_0 \leftarrow live_w_2 \wedge down_s_2$
 $live_w_1 \leftarrow live_w_3 \wedge up_s_1$




How many interpretations?

Today: Learning outcomes

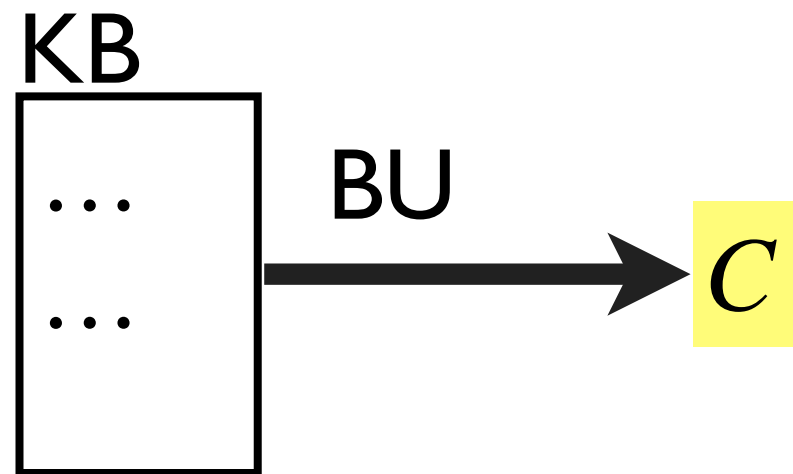
From this lecture, students are expected to be able to:

- Define/read/write/trace/debug the Top-down (SLD) proof procedure (as a search problem)
- Represent simple domains in Datalog
- Apply the Top-down proof procedure in Datalog

Lecture outline

- Class activity (electric environment) (~10 mins)
- Top-down proofs (~30 mins) 
- Datalog (~25 mins)
- Summary and wrap-up (~5 mins)

Recap: Bottom-up (BU) procedure



We showed that BU is **sound**
 $KB \vdash g$ implies $KB \models g$
and **complete**
 $KB \models g$ implies $KB \vdash g$

How do we know whether
query g can be proved from KB or not?

g is proved if $g \subseteq C$

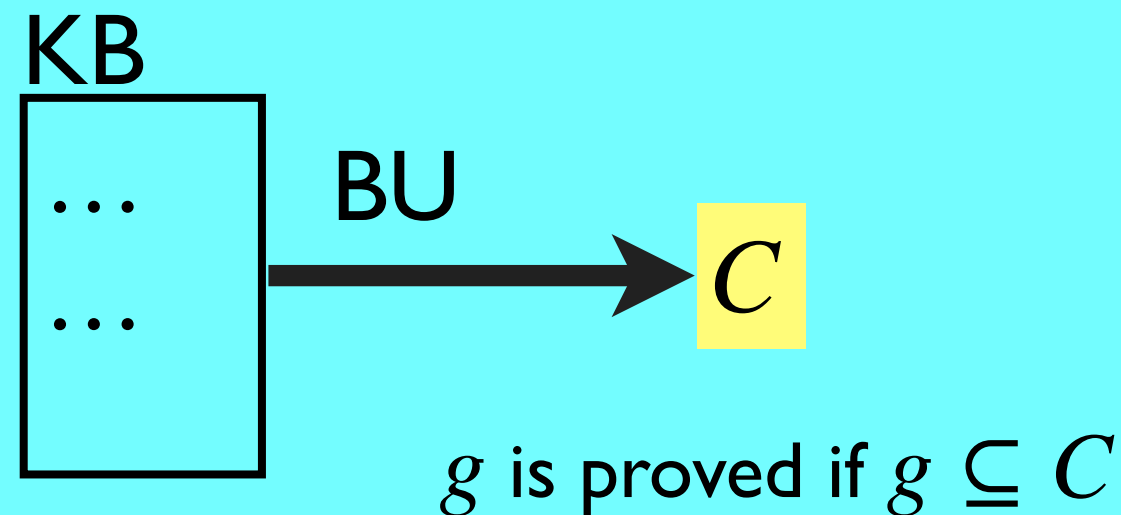
Example:

$C = \{a, b, c, e, h\}$

query e can be proved from KB
because $e \in C$.

Bottom-up (BU) procedure

iclicker.

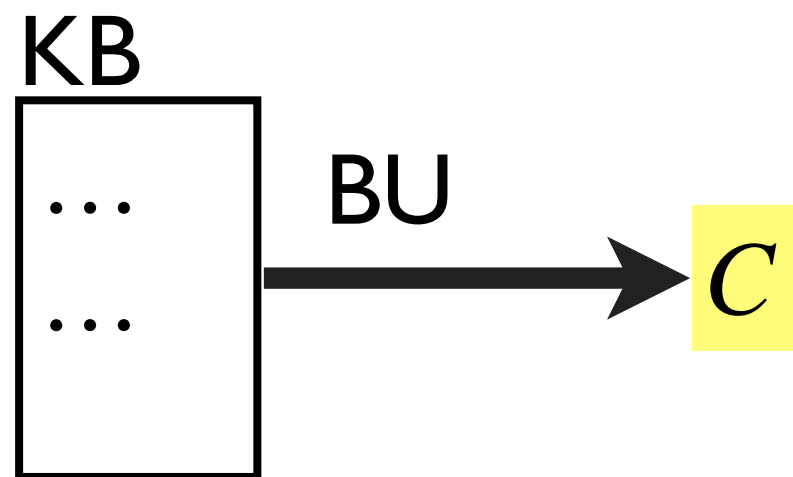


When does BU look at the query g ?

- A. In every loop iteration
- B. Only at the end ☒
- C. Only at the beginning
- D. Never

Bottom-up (BU) vs. Top-down (TD)

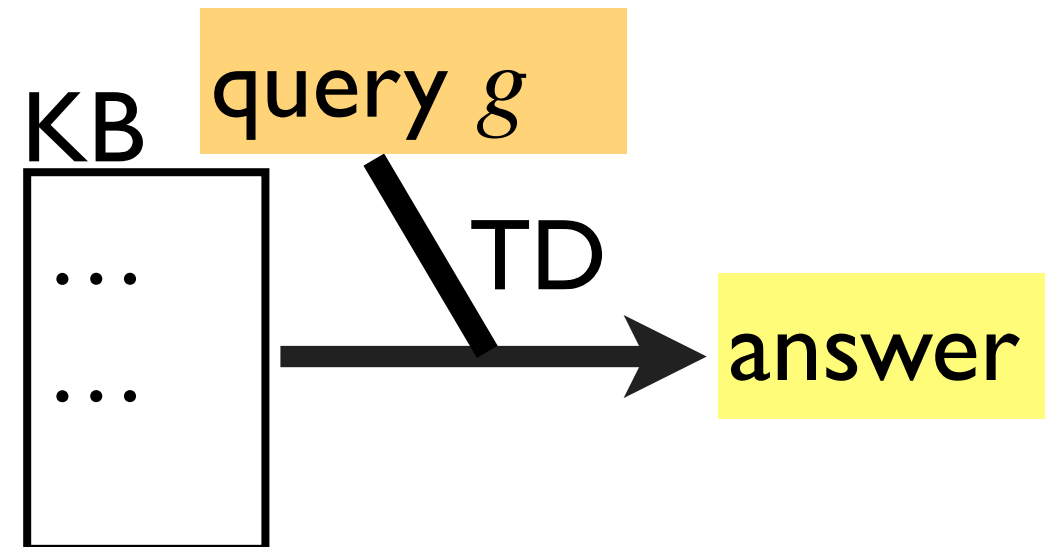
Key idea of top-down: search backward from a query g to determine if it can be derived from KB.



g is proved if $g \subseteq C$

BU looks at query g at the end.

Forward search



TD performs a backward search starting at g

Backward search

Top-down Proof procedure: Basic elements

Idea: search backward from a query

An answer clause is of the form: $yes \leftarrow a_1 \wedge \dots \wedge a_m$, where a_1, \dots, a_m are atoms.

We express the query as an answer clause.

Example: query $q_1 \wedge \dots \wedge q_m$ is expressed as $yes \leftarrow q_1 \wedge \dots \wedge q_m$

Top-down Proof procedure: Basic elements

Idea: search backward from a query

Basic operation: **SLD resolution** of an answer clause

SL: Select a clause using linear strategy.

D: Definite clause

Answer clause: $yes \leftarrow C_1 \wedge \dots \wedge C_{i-1} \wedge C_i \wedge C_{i+1} \wedge \dots \wedge C_m$

The KB clause: $C_i \leftarrow b_1 \wedge \dots \wedge b_p$

Yields: $yes \leftarrow C_1 \wedge \dots \wedge C_{i-1} \wedge b_1 \wedge \dots \wedge b_p \wedge C_{i+1} \wedge \dots \wedge C_m$

Rules of derivation in TD and BU

Top-down: SLD resolution

$$yes \leftarrow C_1 \wedge \dots \wedge C_{i-1} \wedge C_i \wedge C_{i+1} \wedge \dots \wedge C_m$$

$$C_i \leftarrow b_1 \wedge \dots \wedge b_p$$

$$yes \leftarrow C_1 \wedge \dots \wedge C_{i-1} \wedge b_1 \wedge \dots \wedge b_p \wedge C_{i+1} \wedge \dots \wedge C_m$$

Answer clause: $yes \leftarrow x \wedge y \wedge z$

KB clause: $y \leftarrow r$

yields: $yes \leftarrow x \wedge r \wedge z$

Bottom-up: Generalized modus ponens

$$h \leftarrow b_1 \wedge \dots \wedge b_m$$

$$b_1 \wedge \dots \wedge b_m$$

$$h$$

Rule of inference: Examples

SLD resolution

Answer clause: $yes \leftarrow C_1 \wedge \dots \wedge C_{i-1} \wedge C_i \wedge C_{i+1} \wedge \dots \wedge C_m$

The KB clause: $C_i \leftarrow b_1 \wedge \dots \wedge b_p$

Yields: $yes \leftarrow C_1 \wedge \dots \wedge C_{i-1} \wedge b_1 \wedge \dots \wedge b_p \wedge C_{i+1} \wedge \dots \wedge C_m$

answer clause	KB clause	resulting inference
$yes \leftarrow b \wedge c$	$b \leftarrow k \wedge f$	
$yes \leftarrow e \wedge f$	e	

Rule of inference: Examples

SLD resolution

Answer clause: $yes \leftarrow C_1 \wedge \dots \wedge C_{i-1} \wedge C_i \wedge C_{i+1} \wedge \dots \wedge C_m$

The KB clause: $C_i \leftarrow b_1 \wedge \dots \wedge b_p$

Yields: $yes \leftarrow C_1 \wedge \dots \wedge C_{i-1} \wedge b_1 \wedge \dots \wedge b_p \wedge C_{i+1} \wedge \dots \wedge C_m$

answer clause	KB clause	resulting inference
$yes \leftarrow b \wedge c$	$b \leftarrow k \wedge f$	$yes \leftarrow k \wedge f \wedge c$
$yes \leftarrow e \wedge f$	e	$yes \leftarrow f$

TD derivations: example

KB

$a \leftarrow b \wedge c$	$c \leftarrow e$	$f \leftarrow j \wedge e$
$a \leftarrow e \wedge f$	$d \leftarrow k$	$f.$
$b \leftarrow f \wedge k$	$e.$	$j \leftarrow c$

Query: $?a$
(Can we derive a
from KB?)

Derivation



TD derivations: example



KB

$a \leftarrow b \wedge c$	$c \leftarrow e$	$f \leftarrow j \wedge e$
$a \leftarrow e \wedge f$	$d \leftarrow k$	$f.$
$b \leftarrow f \wedge k$	$e.$	$j \leftarrow c$

Query: $?a$
(Can we derive a from KB?)

Derivation

$\gamma_0 : yes \leftarrow a$

$\gamma_1 : yes \leftarrow e \wedge f$

$\gamma_2 : yes \leftarrow e$

$\gamma_3 : yes \leftarrow .$

Yes, we can derive a from KB!

Successful derivation

An **answer** is an answer clause with $m = 0$. That is, it is the answer clause $yes \leftarrow$.

A (successful) derivation from KB of query $?q_1 \wedge \dots \wedge q_k$ is a sequence of answer clauses $\gamma_0, \gamma_1, \dots, \gamma_n$ such that

- γ_0 is the answer clause $yes \leftarrow q_1 \wedge \dots \wedge q_k$
- γ_i is obtained by resolving γ_{i-1} with a clause in KB, and
- γ_n is an “empty” answer.

Unsuccessful derivation

An unsuccessful derivation from KB of query $?q_1 \wedge \dots \wedge q_k$

- We get to something like $yes \leftarrow b_0 \wedge \dots \wedge b_k$, where there is no clause in KB with any of the b_i as its head.

TD derivations: failed derivation example

KB

$a \leftarrow b \wedge c$	$c \leftarrow e$	$f \leftarrow k$
$a \leftarrow e \wedge f$	$d \leftarrow k$	$f.$
$b \leftarrow f \wedge k$	$e.$	$j \leftarrow c$

Query: $?a$
(Can we derive a
from KB?)

Derivation



TD derivations: failed derivation example

KB

$a \leftarrow b \wedge c$	$c \leftarrow e$	$f \leftarrow k$
$a \leftarrow e \wedge f$	$d \leftarrow k$	$f.$
$b \leftarrow f \wedge k$	$e.$	$j \leftarrow c$

Query: $?a$
(Can we derive a from KB?)

Derivation

$\gamma_0 : yes \leftarrow a$

$\gamma_1 : yes \leftarrow e \wedge f$

$\gamma_2 : yes \leftarrow e \wedge k$

$\gamma_3 : yes \leftarrow k$

This time we **failed** because there is no rule with k as its head.



Correspondence between BU and TD proofs

If the following is a top-down (TD) derivation in a given KB, what would be the bottom-up (BU) derivation of the same query?

TD derivation

$yes \leftarrow a$

$yes \leftarrow b \wedge f$

$yes \leftarrow b \wedge g \wedge h$

$yes \leftarrow c \wedge d \wedge g \wedge h$

$yes \leftarrow d \wedge g \wedge h$

$yes \leftarrow g \wedge h$

$yes \leftarrow h$

$yes \leftarrow .$

BU derivation

$\{ \}$

$\{ h \}$

$\{ g, h \}$

$\{ d, g, h \}$

$\{ c, d, g, h \}$

$\{ b, c, d, g, h \}$

$\{ b, c, d, f, g, h \}$

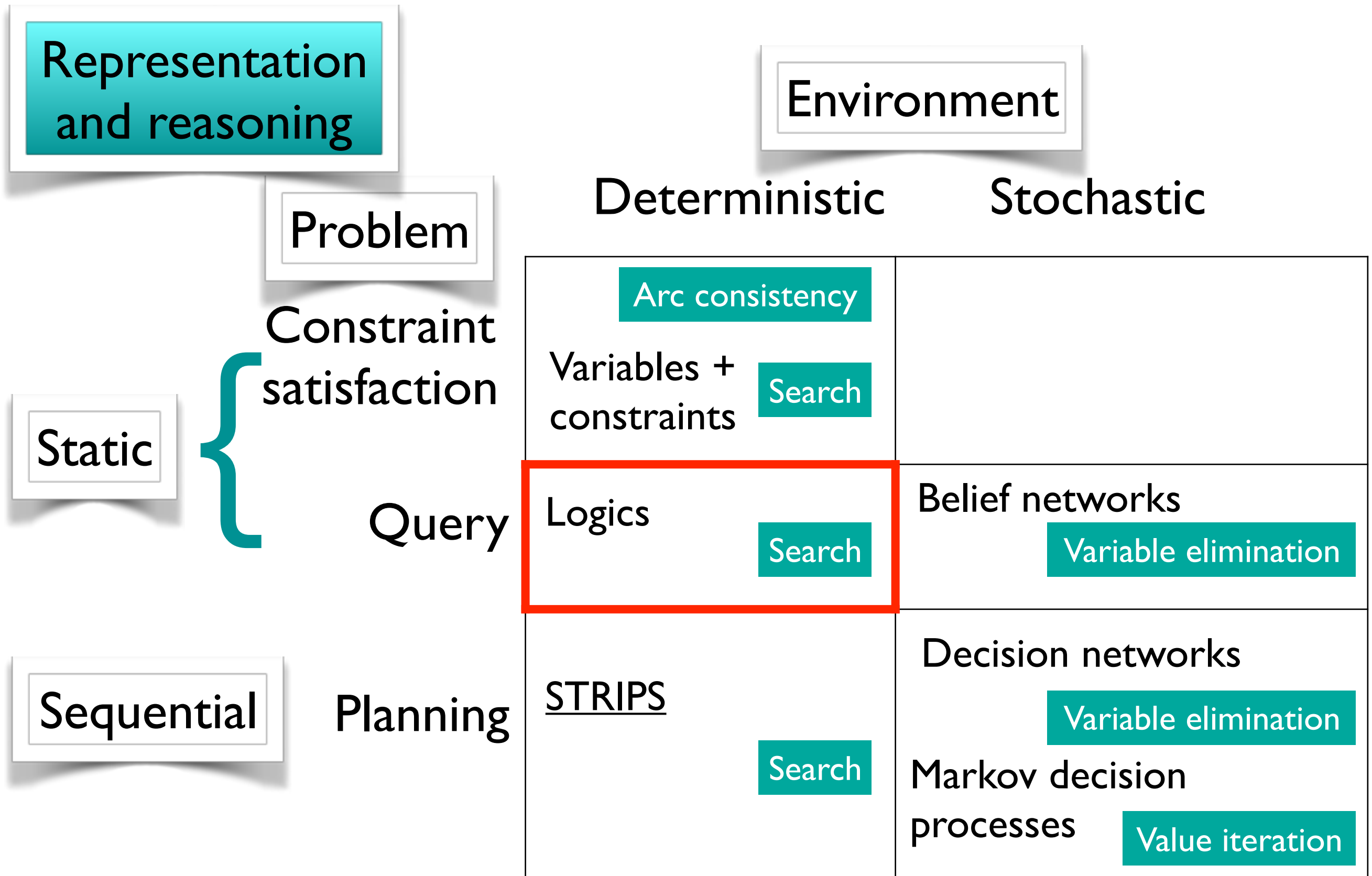
$\{ a, b, c, d, f, g, h \}$

TD: Completeness and soundness

Is top-down proof procedure **sound** and **complete**?

- Yes, since there is a 1:1 correspondence between top-down and bottom-up proofs
- The two methods derive exactly the same atoms (if the SLD resolution picks the successful derivations)

A rough CPSC 322 overview



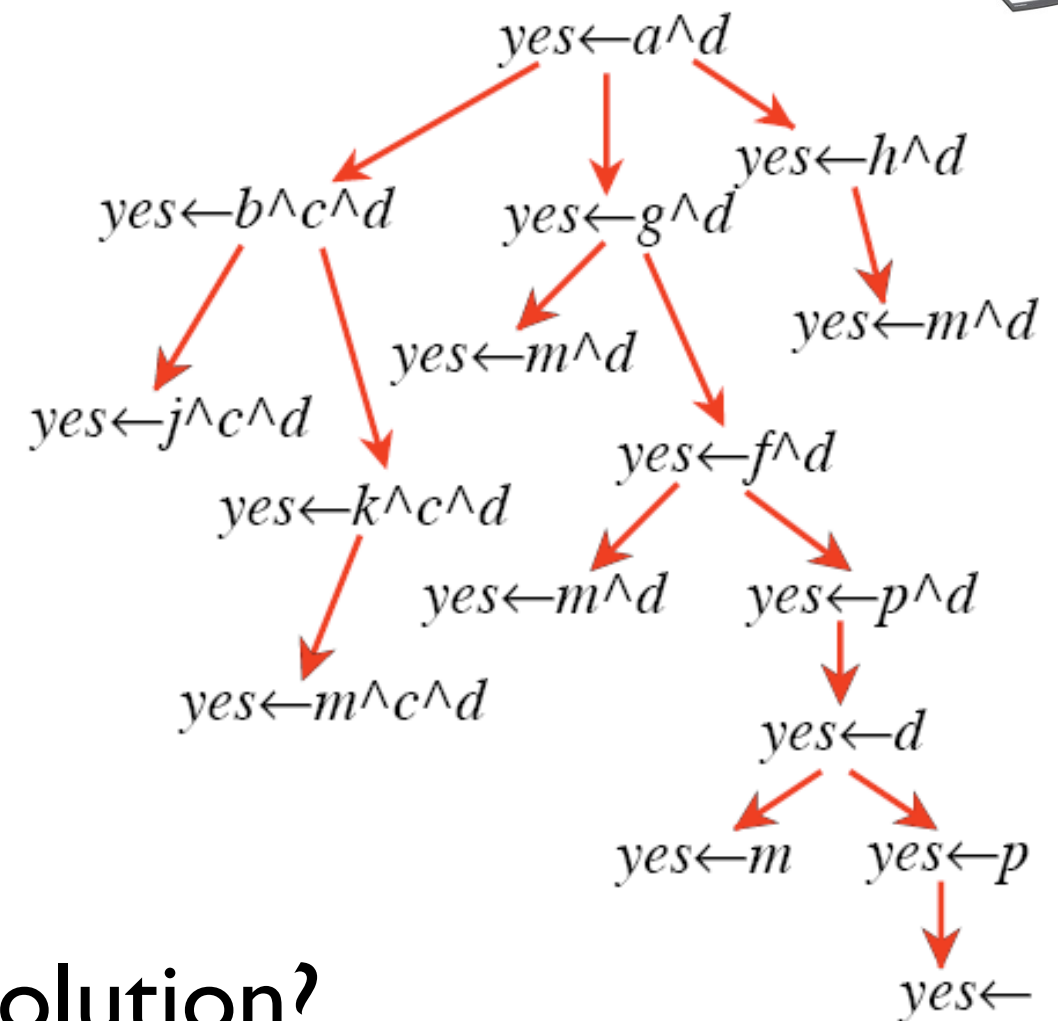
Search graph for TD proofs



KB

$a \leftarrow b \wedge c.$	$a \leftarrow g.$
$a \leftarrow h.$	$b \leftarrow j.$
$b \leftarrow k.$	$d \leftarrow m.$
$d \leftarrow p.$	$f \leftarrow m.$
$f \leftarrow p.$	$g \leftarrow m.$
$g \leftarrow f.$	$k \leftarrow m.$
$h \leftarrow m.$	$p.$

Query: $?a \wedge d$



What kind of search is SLD resolution?

1. BFS
2. DFS

It's a depth-first-search. Failing resolutions are paths where the search has to backtrack.

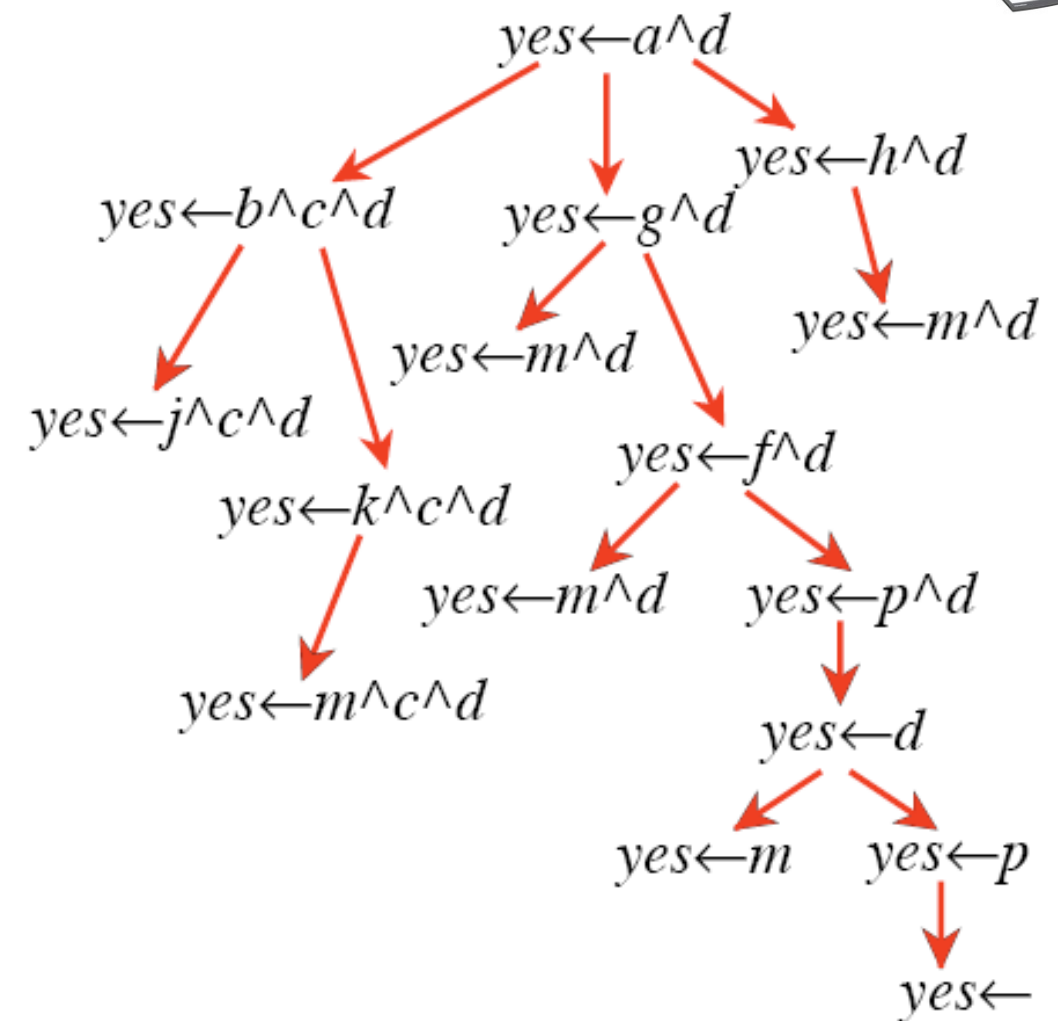
Search graph for TD proofs



KB

$a \leftarrow b \wedge c.$	$a \leftarrow g.$
$a \leftarrow h.$	$b \leftarrow j.$
$b \leftarrow k.$	$d \leftarrow m.$
$d \leftarrow p.$	$f \leftarrow m.$
$f \leftarrow p.$	$g \leftarrow m.$
$g \leftarrow f.$	$k \leftarrow m.$
$h \leftarrow m.$	$p.$

Query: $?a \wedge d$



Can we use heuristics?

A: Yes!

Example: number of atoms in the answer clause

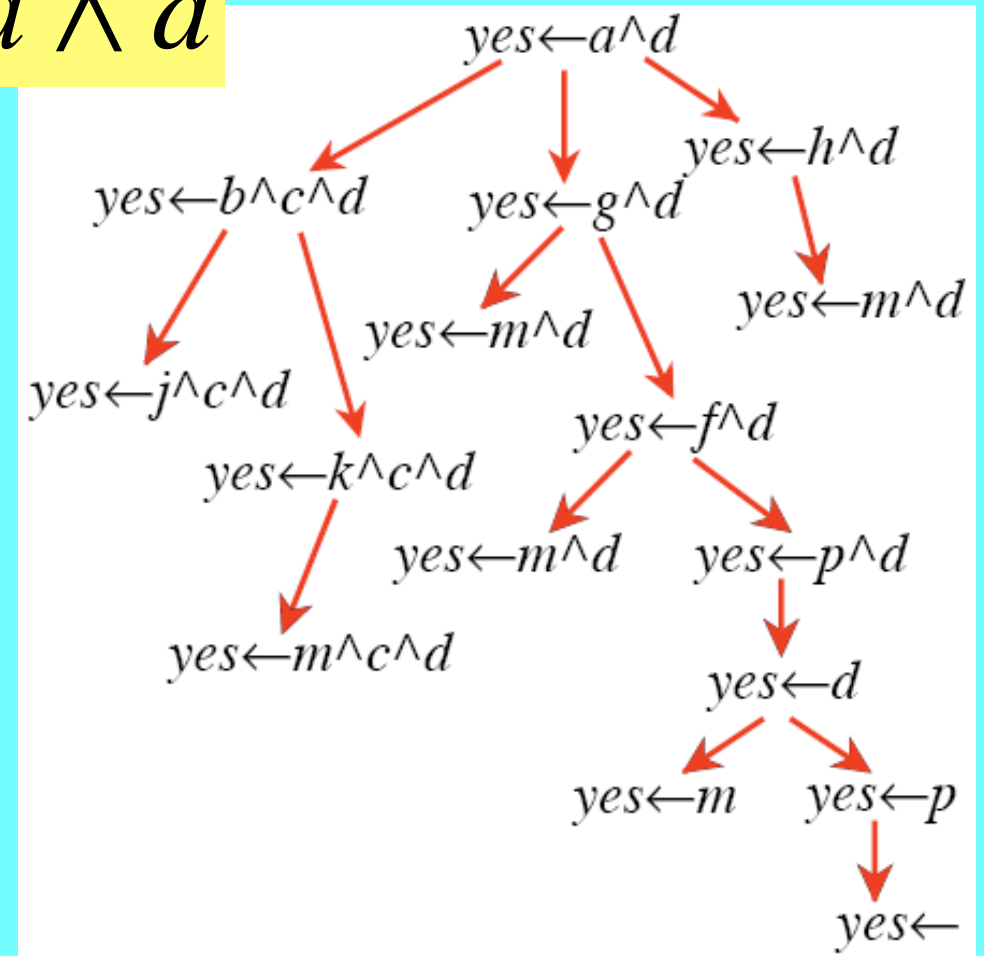
Search graph for TD proofs

iclicker.

KB

$a \leftarrow b \wedge c.$	$a \leftarrow g.$
$a \leftarrow h.$	$b \leftarrow j.$
$b \leftarrow k.$	$d \leftarrow m.$
$d \leftarrow p.$	$f \leftarrow m.$
$f \leftarrow p.$	$g \leftarrow m.$
$g \leftarrow f.$	$k \leftarrow m.$
$h \leftarrow m.$	$p.$

Query: $?a \wedge d$



Possible heuristic is the number of unique atoms in the answer clause.
Is it admissible?

A. Yes ☒


B. No

C. It depends

Heuristics for TD as search



If the body of an answer clause contains a symbol that does not match the head of any clause in the KB what should the most informative heuristic value for that answer clause be ?

- A. Zero
- B. Infinity 
- C. Twice the number of clauses in the KB
- D. None of the above

Representation and reasoning with logical inference (TD)

State: answer clause of the form $yes \leftarrow q_1 \wedge \dots \wedge q_k$


Successor function: states resulting from substituting first atom a with $b_1 \wedge \dots \wedge b_m$ if there is a clause $a \leftarrow b_1 \wedge \dots \wedge b_m$

Goal test: Is the answer clause empty? (i.e., $yes \leftarrow .$)

Solution: the proof, i.e., the sequence of SLD resolution

Heuristic function: number of atoms in the query clauses.

Lecture outline

- Class activity (electric environment) (~10 mins)
- Top-down proofs (~30 mins)
- Datalog (~25 mins) 
- Summary and wrap-up (~5 mins)

Representation and reasoning in complex domains

In complex domains
expressing knowledge with
propositions can be quite
limiting

up_s₂
up_s₃
ok_cb₁
ok_cb₂
live_w₁
connected_w₁_w₂

E.g., there is no notion that
w₁ is the same in *live_w₁*
and *connected_w₁_w₂*

It is often more natural to
consider **individuals** and
their **properties**.

up(s₂)
up(s₃)
ok(cb₁)
ok(cb₂)
live(w₁)
connected(w₁ , w₂)

Now there is a notion that
w₁ is the same in *live_w₁*
and *connected_w₁_w₂*

Datalog: What do we gain?

- An extension of propositional definite clause (PDC) logic
We now have **variables**.
We now have **relationships** between variables.

- We can express knowledge that holds for a set of individuals, writing more powerful clauses by introducing variables, such as:

$live(W) \leftarrow wire(W) \wedge connected_to(W, W_1) \wedge wire(W_1) \wedge live(W_1)$

- We can ask **generic queries**:

E.g., Which wires are connected to W_1 ?

$?connected_to(W, W_1)$

Datalog: a relational rule language

Datalog expands the syntax of PDCL...

A **variable** is a symbol starting with an upper case letter.

Examples: *X*, *Y*

A **constant** is a symbol starting with a lower case letter or a sequence of digits.

Examples: *322*, *cpsc*

A **term** is either a variable or a constant.

Examples: *X*, *Y*, *322*, *cpsc*

A **predicate symbol** is a symbol starting with a lower case letter.

Examples: *live*, *connected*, *in*, *part_of*

Datalog: a relational rule language

An **atom** is a symbol of the form p or $p(t_1, \dots, t_n)$ where p is a predicate symbol and t_i are terms.

Examples: *sunny*, *connected*(W_1, W_2)

A **definite clause** is either an atom (a fact) or of the form $h \leftarrow a_1 \wedge \dots \wedge a_m$, where h and a_i are atoms

Examples: $in(X, Z) \leftarrow in(X, Y) \wedge part_of(Y, Z)$

A **knowledge base** is a set of definite clauses.

Example: KB =
 $\{in(kim, r123),$
 $part_of(r123, cs_building),$
 $in(X, Y) \leftarrow part_of(Z, Y) \wedge in(X, Z)\}$

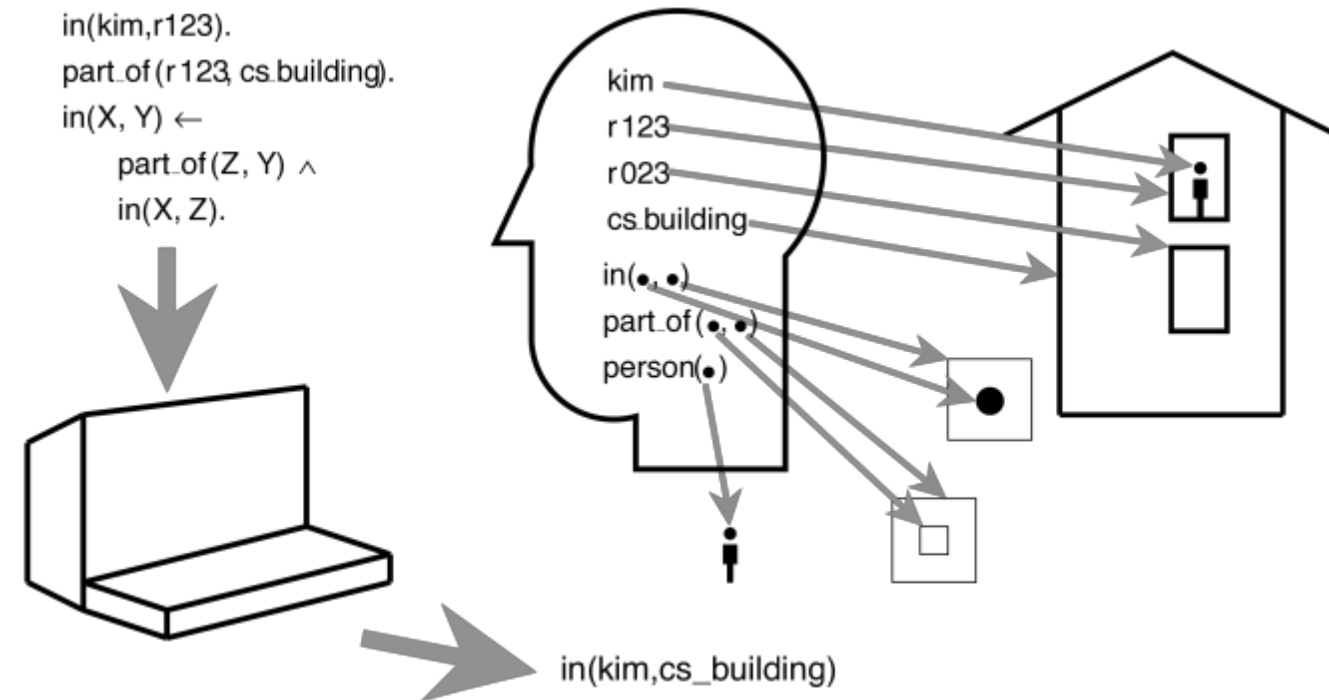


Datalog vs. PDCL

<p><u>First order logic (FOL)</u></p> $\forall X \exists Y p(X, Y) \iff \neg q(Y)$ $p(a_1, a_2)$ $q(a_4)$	<p><u>Datalog (DL)</u></p> $p(X) \leftarrow q(X) \wedge r(X, Y)$ $r(X, Y) \leftarrow s(Y)$ $s(a_1), q(a_4)$
<p><u>Propositional logic (PL)</u></p> $\neg p \vee q \rightarrow r \wedge s$ p	<p><u>PDCL</u></p> $p \leftarrow q \wedge r$ p $p \leftarrow q \wedge r \wedge s$

Intended interpretation

- User chooses task domain: **intended interpretation**. This is the interpretation of the symbols the user has in mind.
- User tells the system clauses (the knowledge base KB).
- Each clause is true in the user's intended interpretation. Thus, the intended interpretation is a model.
- **The computer does not know the intended interpretation.** But if it can derive something that's true in all models, then it is true in the intended interpretation



If $KB \models g$, then

- g is true in the intended interpretation
- g is true in every model of KB (by definition)

Datalog: Top-down proof procedure

Extension of top-down procedure for PDCL

How do we deal with variables?

- Find a clause with head that matches the query
- Substitute variables in the clause with their matching constants

Datalog: Top-down proof procedure

enrolled(jane,322)

department(322,comp_science)

in_dept(S,D) ← enrolled(S,C) ∧ department(C,D)

[in_dept(jane, comp_science)]

Query: *in_dept(jane, comp_science)*

**Find a clause with
head that matches the
query**

Substitute variables in the clause
with their matching constants

Datalog: Top-down proof procedure

enrolled(jane,322)

department(322,comp_science)

in_dept(S,D) ← enrolled(S,C) ∧ department(C,D)

[in_dept(jane, comp_science)]



[enrolled(jane, C), department(C, comp_science)]

Query: *in_dept(jane, comp_science)*

Find a clause with head that matches the query

Use clause

in_dept(S,D) ← enrolled(S,C) ∧

department(C,D) with

D = comp_science, S = jane

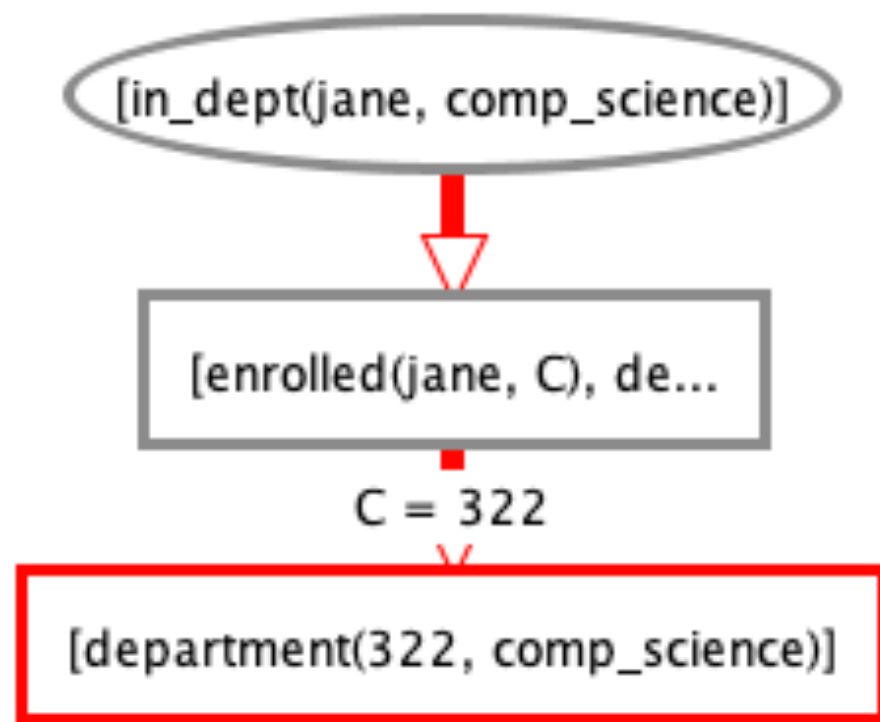
Substitute variables in the clause with their matching constants

Datalog: Top-down proof procedure

enrolled(jane,322)

department(322,comp_science)

in_dept(S,D) ← enrolled(S,C) ∧ department(C,D)



Query: *in_dept(jane, comp_science)*

Find a clause with head that matches the query

Substitute variables in the clause with their matching constants

Use clause

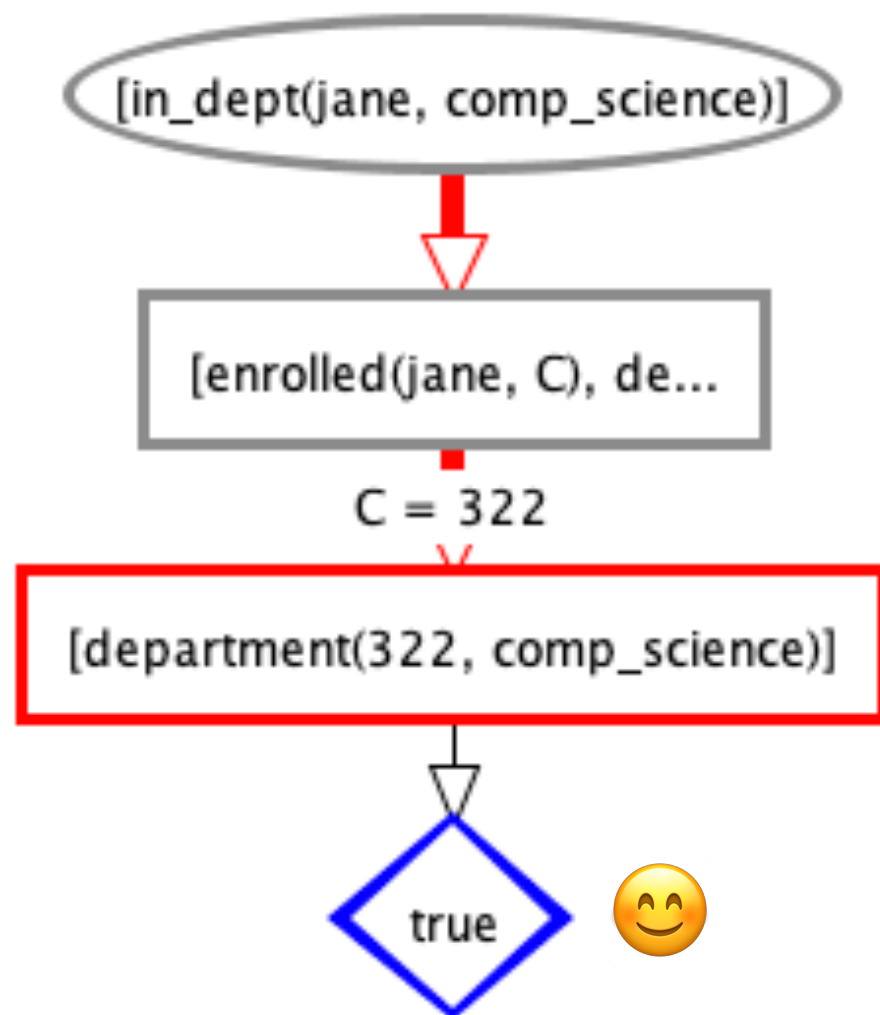
department(C, comp_science) with
C = 322

Datalog: Top-down proof procedure

enrolled(jane,322)

department(322,comp_science)

in_dept(S,D) ← enrolled(S,C) ∧ department(C,D)



Query: *in_dept(jane, comp_science)*

Find a clause with head that matches the query

Substitute variables in the clause with their matching constants

Tracing Datalog proofs in Alspace



You can trace the example from the last slide in the Alspace Deduction Applet at <http://aispace.org/deduction/> using the sample problem “A Course Database”.

Question 4 of Assignment 3 will ask you to use this applet

Datalog queries with variables



Knowledge Base

```
course(312).
course(322).
course(315).
course(371).
department(312, comp_science).
department(322, comp_science).
department(315, math).
department(371, physics).
student(mary).
student(jane).
student(john).
student(harold).
female(mary).
female(jane).
enrolled(mary, 322).
enrolled(mary, 312).
enrolled(john, 322).
enrolled(john, 315).
enrolled(harold, 322).
enrolled(mary, 315).
enrolled(jane, 312).
enrolled(jane, 322).
cs_course(C) <- department(C, comp_science).
math_course(C) <- department(C, math).
cs_or_math_course(C) <- cs_course(C).
cs_or_math_course(C) <- math_course(C).
in_dept(S, D) <- enrolled(S, C) & department(C, D).
```

Alspace deduction applet

Load sample knowledge base ->
A Course Database

Query: *in_dept(X, comp_science)*

Practice exercise



Load the following on Alspace applet

[https://raw.githubusercontent.com/kvarada/
CPSC-322_students/master/Alspace_files/in-part-of.pl](https://raw.githubusercontent.com/kvarada/CPSC-322_students/master/Alspace_files/in-part-of.pl)

Try the following queries:

Query 1: *in(kim, cs_building)*

Query 2: *in(kim, X)*

Practice exercises


Try to do these on your own, without looking at the solution.

<http://www.aispace.org/exercises/exercise12-b-1.shtml>

One important Datalog detail

- In its SLD resolution proof, Datalog always chooses the first clause with a matching head it finds in KB
- What does that mean for recursive function definitions?
 - The clause(s) defining your base case(s) have to appear first in KB
 - Otherwise, you can get infinite recursions
 - This is similar to recursion in imperative and functional programming languages
- Datalog is a subset of Prolog (Programming in Logic)

Lecture outline

- Class activity (electric environment) (~10 mins)
- Top-down proofs (~30 mins)
- Datalog (~25 mins)
- Summary and wrap-up (~5 mins) 

Revisit: Learning outcomes for logics

PDCL syntax and semantics

- Verify whether a logical statement belongs to the language of propositional definite clauses
- Verify whether an interpretation is a model of a PDCL KB.
- Verify when a conjunction of atoms is a logical consequence of a KB

Bottom-up proof procedure

- Define/read/write/trace/debug the Bottom Up (BU) proof procedure
- Prove that the BU proof procedure is sound and complete

Revisit: Learning outcomes for logics

Top-down proof procedure

- Define/read/write/trace/debug the Top-down (SLD) proof procedure (as a search problem)

Datalog

- Represent simple domains in Datalog
- Apply the Top-down proof procedure in Datalog

Final remarks

We only covered rather simple logics

There are much more powerful representation and reasoning systems based on logics, e.g., full first order logic (with negation, disjunction and function symbols), second-order logics, non-monotonic logics, modal logics, ...

Applications: Automated travel agent

Examples for typical queries

- How much is a typical flight to Mexico for a given date?
- What's the cheapest vacation package to some place in the Caribbean in a given week?
- Plus, the hotel should have a white sandy beach and scuba diving

Semantic Web

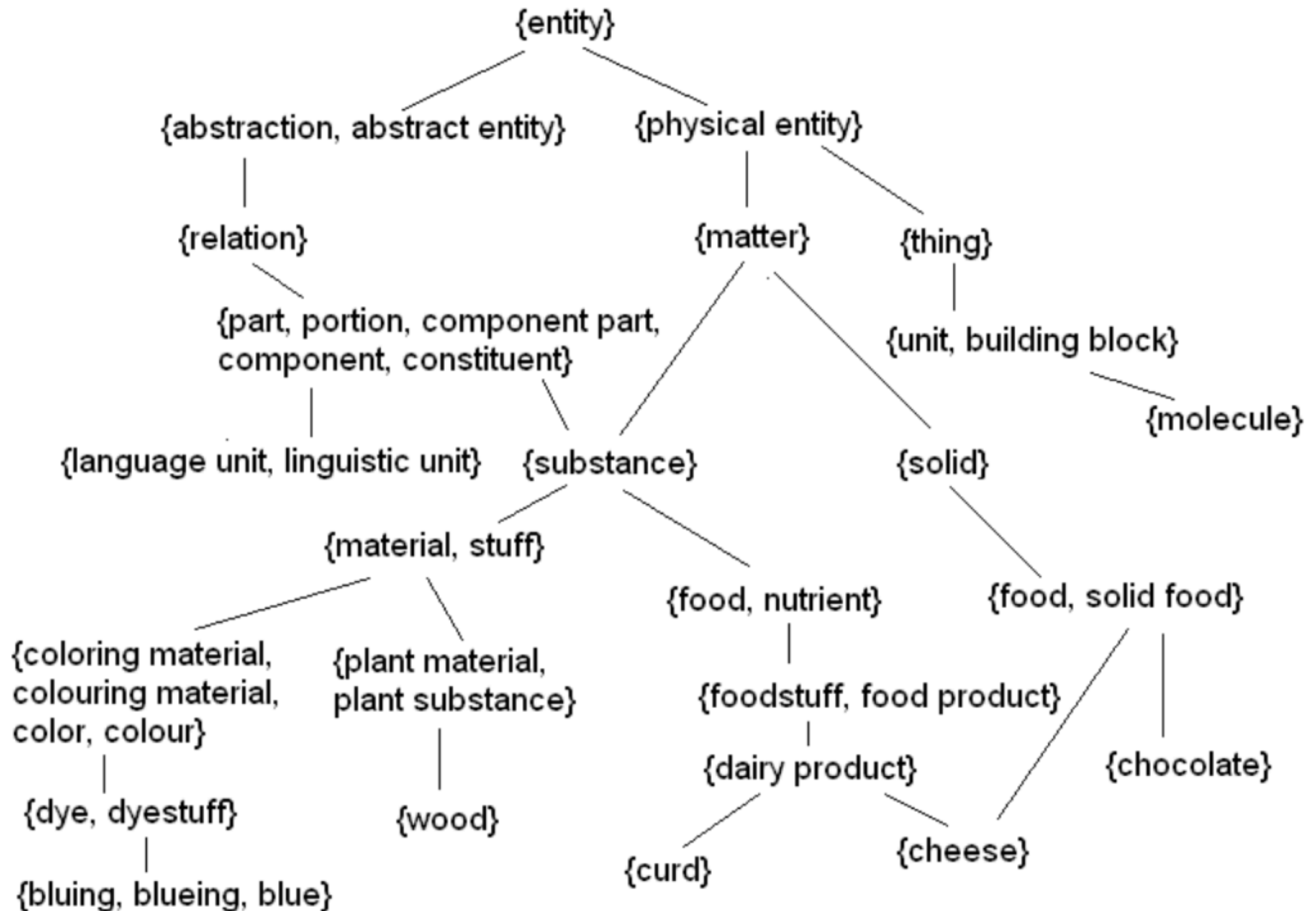
- Languages and formalisms based on logics that allow websites to include information in a more structured format
- Goal: software agents that can roam the web and carry out sophisticated tasks on our behalf.
- This is different than searching content for keywords and popularity!
- For further material, see P&M text, Chapter 13 and the Introduction to the Semantic Web tutorial given at 2011 Semantic Technology Conference <http://www.w3.org/People/Ivan/CorePresentations/SWTutorial/>

Example ontologies for the Semantic Web

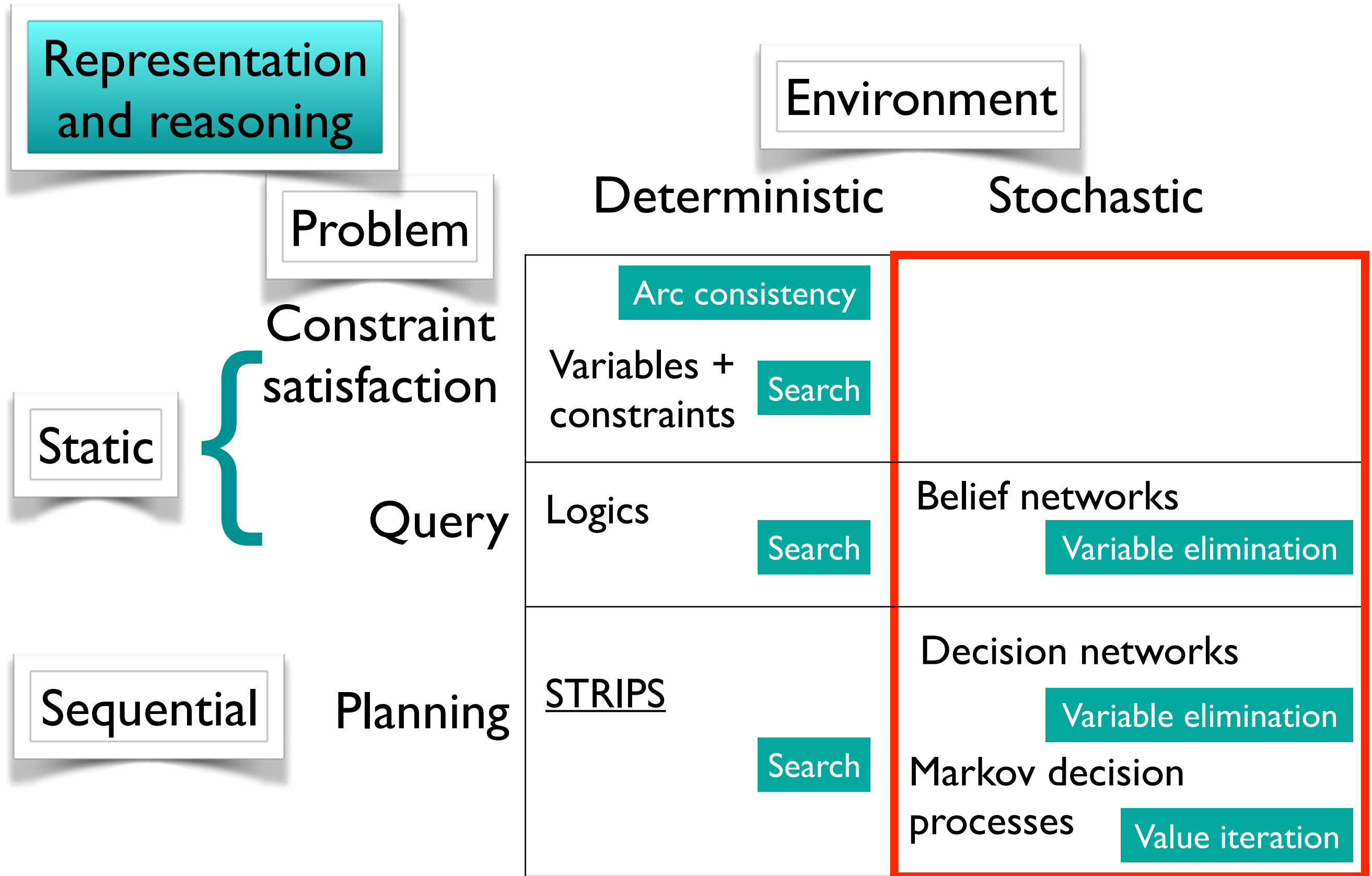
“Ontology”: logic-based representation of the world

- ClassOwl: eBusiness ontology
for products and services
75,000 classes (types of individuals) and 5,500 properties
- National Cancer Institute’s ontology: 58,000 classes
- Open Biomedical Ontologies Foundry: several ontologies including the Gene Ontology to describe, gene and gene product attributes in any organism or protein sequence, annotation terminology and data
- OpenCyc project: a 150,000-concept ontology including
Top-level ontology describes general concepts such as numbers, time, space, etc
Hierarchical composition: superclasses and subclasses
Many specific concepts such as “OLED display” , “iPhone”

Example ontology: WordNet



A rough CPSC 322 overview



Coming up

Introduction to probability

8.1 Probability

Random Variable, Probability Distribution, Marginalization, Conditional Probability, Chain Rule, Bayes' Rule, Marginal and Conditional Independence

