Logic: TD as search, Datalog (variables)

CPSC 322 Lecture 22

Lecture Overview

- Recap Top Down
- TopDown Proofs as search
- Datalog

Bottom-up vs. Top-down

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

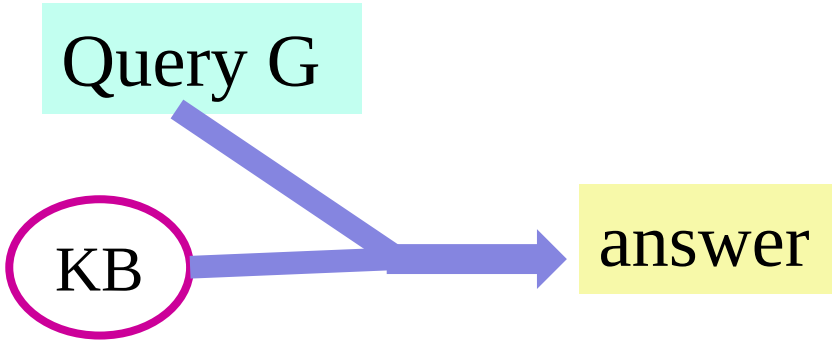
Bottom-up

G is proved if $G \subseteq C$

When does BU look at the query G?

- At the end

Top-down



TD performs a backward **search** starting at G

Top-down Proof Procedure: Elements

Notation: An answer clause is of the form:

$yes \leftarrow a_1 \wedge a_2 \wedge \ldots \wedge a_m$

Express query as an answer clause

(e.g., query $a_1 \wedge a_2 \wedge \ldots \wedge a_m$)

$yes \leftarrow a_1 \wedge \ldots \wedge a_m$

Rule of inference (called SLD Resolution)

Given an answer clause of the form:

$$yes \leftarrow a_1 \wedge a_2 \wedge \ldots \wedge a_m$$

and the KB clause:

$$a_i \leftarrow b_1 \wedge b_2 \wedge \ldots \wedge b_p$$

You can generate the answer clause

$yes \leftarrow a_1 \wedge \ldots \wedge a_{i-1} \wedge b_1 \wedge b_2 \wedge \ldots \wedge b_p \wedge a_{i+1} \wedge \ldots \wedge a_m$

- **Successful Derivation**: When by applying the inference rule you obtain the answer clause ***yes*** ← **.**

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

Query: a `(two ways)`

$yes \leftarrow a.$ $yes \leftarrow a.$

Lecture Overview

- Recap Top Down
- TopDown Proofs as search
- Datalog

Systematic Search in different R&R systems

Constraint Satisfaction (Problems):

- State: assignments of values to a subset of the variables
- Successor function: assign values to a “free” variable
- Goal test: set of constraints
- Solution: possible world that satisfies the constraints
- Heuristic function: *none (all solutions at the same distance from start)*

Planning (forward) :

- State possible world
- Successor function states resulting from valid actions
- Goal test assignment to subset of vars
- Solution sequence of actions
- Heuristic function empty-delete-list (solve simplified problem)

Logical Inference (top Down)

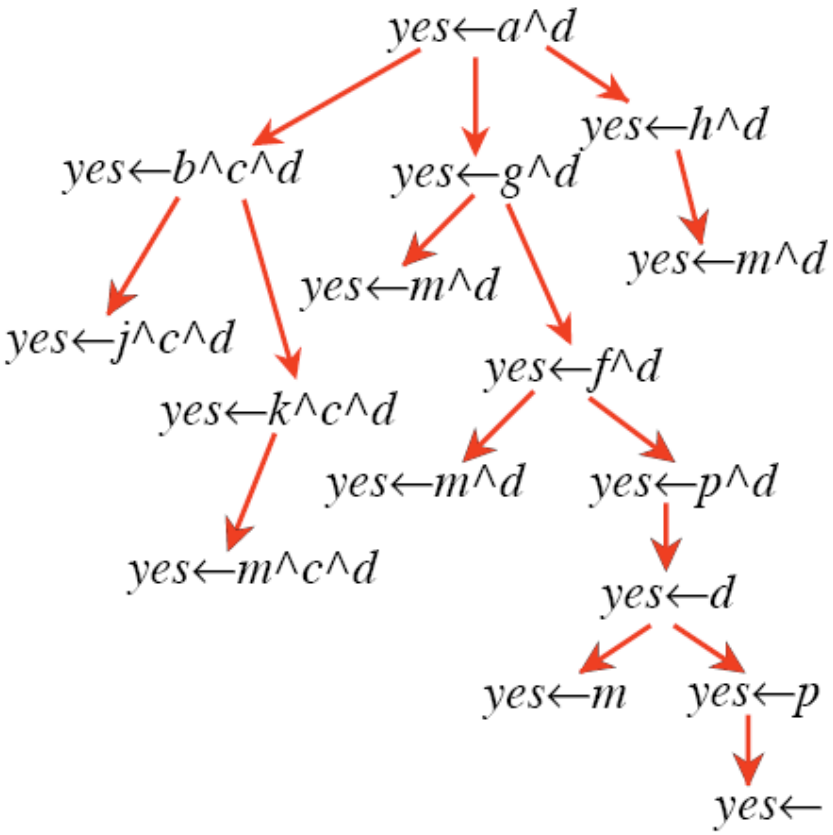
- State answer clause
- Successor function states resulting from substituting one atom with all the clauses of which it is the head
- Goal test empty answer clause
- Solution start state
- Heuristic function ……..

Search Graph

KB

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

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



Prove: ?← $a \wedge d.$

Possible Heuristic?

Number of unique atoms in the answer clause

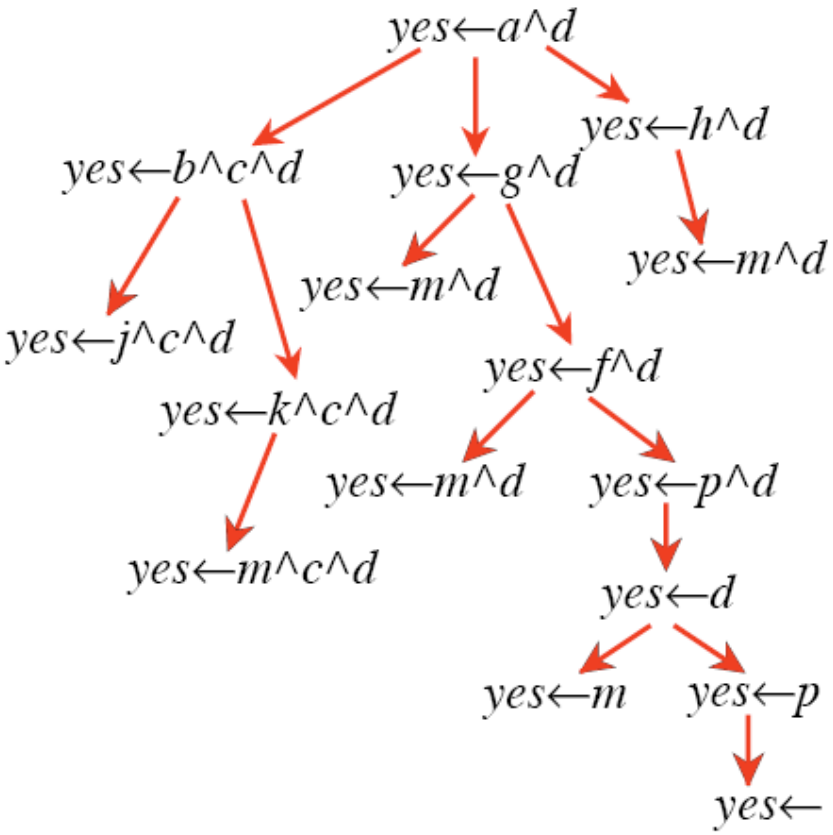
Admissible?

A. Yes B. No C. It Depends

Search Graph

Prove: ?← ***a*** ∧ ***d***.

a ← ***b*** ∧ ***c***.	***a*** ← ***g***.
a ← ***h***.	*b* ← *j*.
b ← *k*.	*d* ← *m*.
d ← ***p***.	*f* ← *m*.
f ← *p*.	*g* ← *m*.
g ← *f*.	*k* ← *m*.
h ← *m*.	*p*.



"Heuristic" for clause selection?

Better Heuristics?

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. The number of atoms in that body
E. 42

Lecture Overview

- Recap Top Down
- TopDown Proofs as search
- Datalog

Representation and Reasoning in Complex domains

- In complex domains expressing knowledge with **propositions** can be quite limiting

  up_s_2
  up_s_3
  ok_cb_1
  ok_cb_2
  $live\_w_1$
  $connected\_w_1\_w_2$

- It is often natural to consider **individuals** and their **properties**

  $up(\ s_2\ )$
  $up(\ s_3\ )$
  $ok(\ cb_1\ )$
  $ok(\ cb_2\ )$
  $live(\ w_1)$
  $connected(\ w_1\ ,\ w_2\ )$

There is no notion that

up_s_2
up_s_3 share a meaning

$live\_w_1$
$connected\_w_1\_w_2$ are about the same individual

What do we gain….

…by turning **propositions** into **relations** applied to **individuals**?

- Express **knowledge** that **holds for set of individuals** (by introducing variables)

$$live(W) \leftarrow connected\_to(W,W1) \wedge live(W1) \wedge wire(W) \wedge wire(W1).$$

- We can **ask generic queries** (i.e., containing variables)

$$? \; connected\_to(W, w_1)$$

Datalog vs PDCL (better with colors)

First Order Logic

$\forall X \exists Y p(X,Y) \Leftrightarrow \neg q(Y)$

$p(a_1, a_2)$

$\neg q(a_5)$

Propositional Logic

$\neg(p \vee q) \rightarrow (r \wedge s \wedge f)$,

p, r

Datalog

$p(X) \leftarrow q(X) \wedge r(X,Y)$

$r(X,Y) \leftarrow s(Y)$

$s(a_1), q(a_2)$

PDCL

$p \leftarrow s \wedge f$

$r \leftarrow s \wedge q \wedge p$

r

p

Datalog: a relational rule language

Datalog expands the syntax of PDCL….

A variable is a symbol starting with an upper case letter

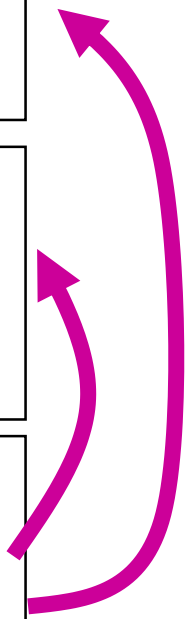
Examples: X, Y

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

Examples: alan, w1

A term is either a variable or a constant.

Examples: X, Y, alan, w1



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

Examples: live, connected, part-of, in

Datalog Syntax (cont'd)

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

Examples: sunny, in(alan,X)

A definite clause is either an atom (a fact) or of the form:

$$h \leftarrow b_1 \wedge \ldots \wedge b_m$$

where h and the b_i are atoms (Read this as ``h if b.")

Example: in(X,Z) ← in(X,Y) ∧ part-of(Y,Z)

A knowledge base is a set of definite clauses

Datalog: Top Down Proof Procedure

in(alan, r123).

part_of(r123,cs_building).

in(X,Y) ← part_of(Z,Y) ∧ in(X,Z).

- Extension of Top-Down procedure for PDCL.
  How do we deal with variables?
  - Idea:
    - Find a clause with head that matches the query
    - Substitute variables in the clause with their matching constants
  - Example:

Query: yes ← in(alan, cs_building).

in(X,Y) ← part_of(Z,Y) ∧ in(X,Z).
with Y = cs_building
X = alan

yes ← part_of(Z,cs_building) ∧ in(alan, Z).

Example proof of a Datalog query

in(alan, r123).

part_of(r123,cs_building).

in(X,Y) ← part_of(Z,Y) ∧ in(X,Z).

Query: yes ← in(alan, cs_building).

Using clause: in(X,Y) ← part_of(Z,Y) ∧ in(X,Z), with Y = cs_building X = alan

yes ← part_of(Z,cs_building) ∧ in(alan, Z).

Using clause: part_of(r123,cs_building) with Z = r123

??????

i>clicker

A. yes ← part_of(Z, r123) ∧ in(alan, Z).

B. yes ← in(alan, r123).

C. yes ←.

D. None of the above

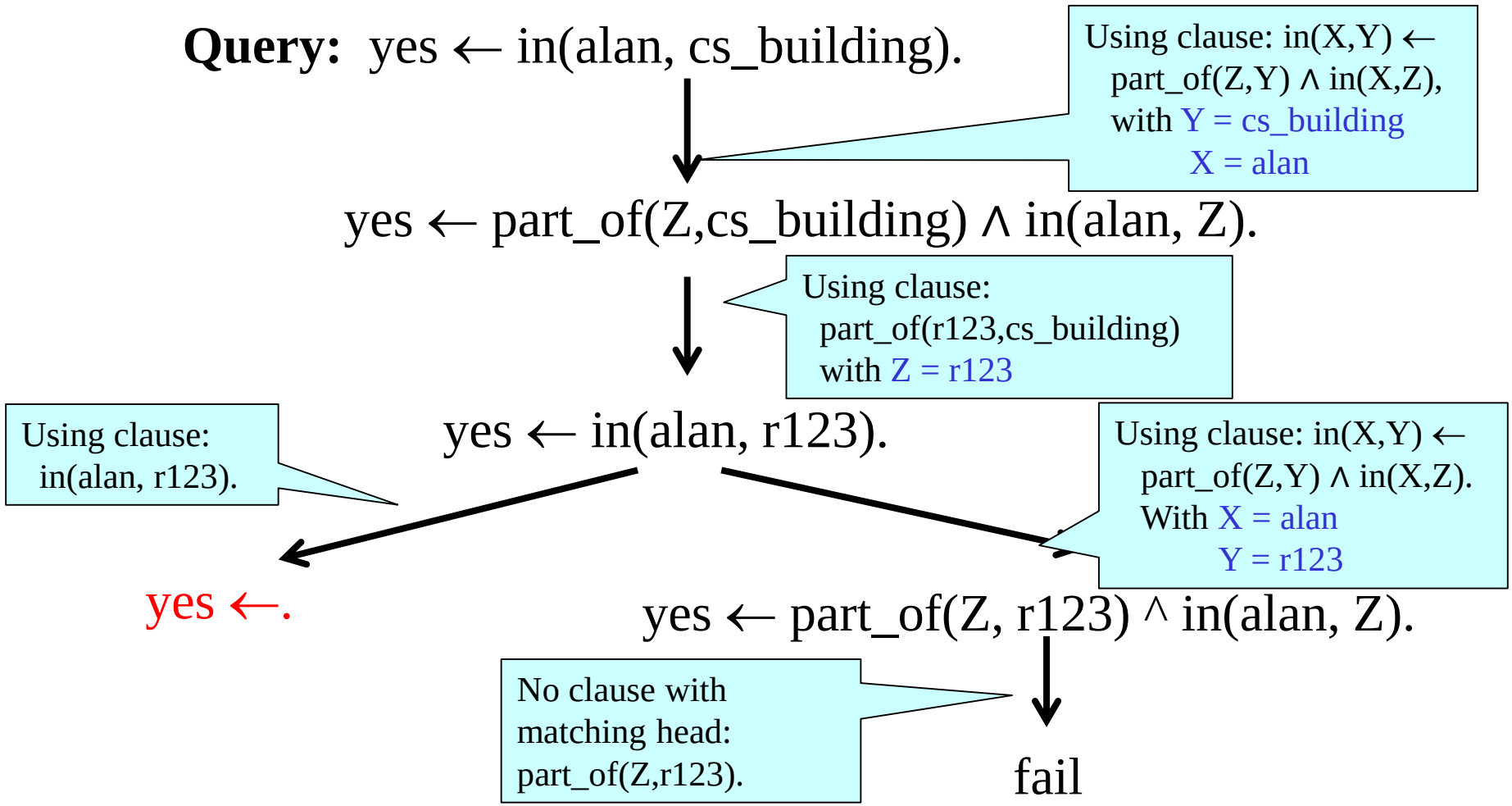
E. You'd need a babelfish in your ear to figure this out

Example proof of a Datalog query

in(alan, r123).

part_of(r123,cs_building).

in(X,Y) ← part_of(Z,Y) ∧ in(X,Z).



Tracing Datalog proofs in AIspace

- You can trace the example from the last slide in the AIspace Deduction Applet at http://aispace.org/deduction/ using file *ex-Datalog* available in course schedule

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

Datalog: queries with variables

in(alan, r123).
part_of(r123,cs_building).
in(X,Y) ← part_of(Z,Y) & in(X,Z).

Query: in(alan, X1).

yes(X1) ← in(alan, X1).

What would the answer(s) be?

Datalog: queries with variables

in(alan, r123).
part_of(r123,cs_building).
in(X,Y) ← part_of(Z,Y) & in(X,Z).

Query: in(alan, X1).

yes(X1) ← in(alan, X1).

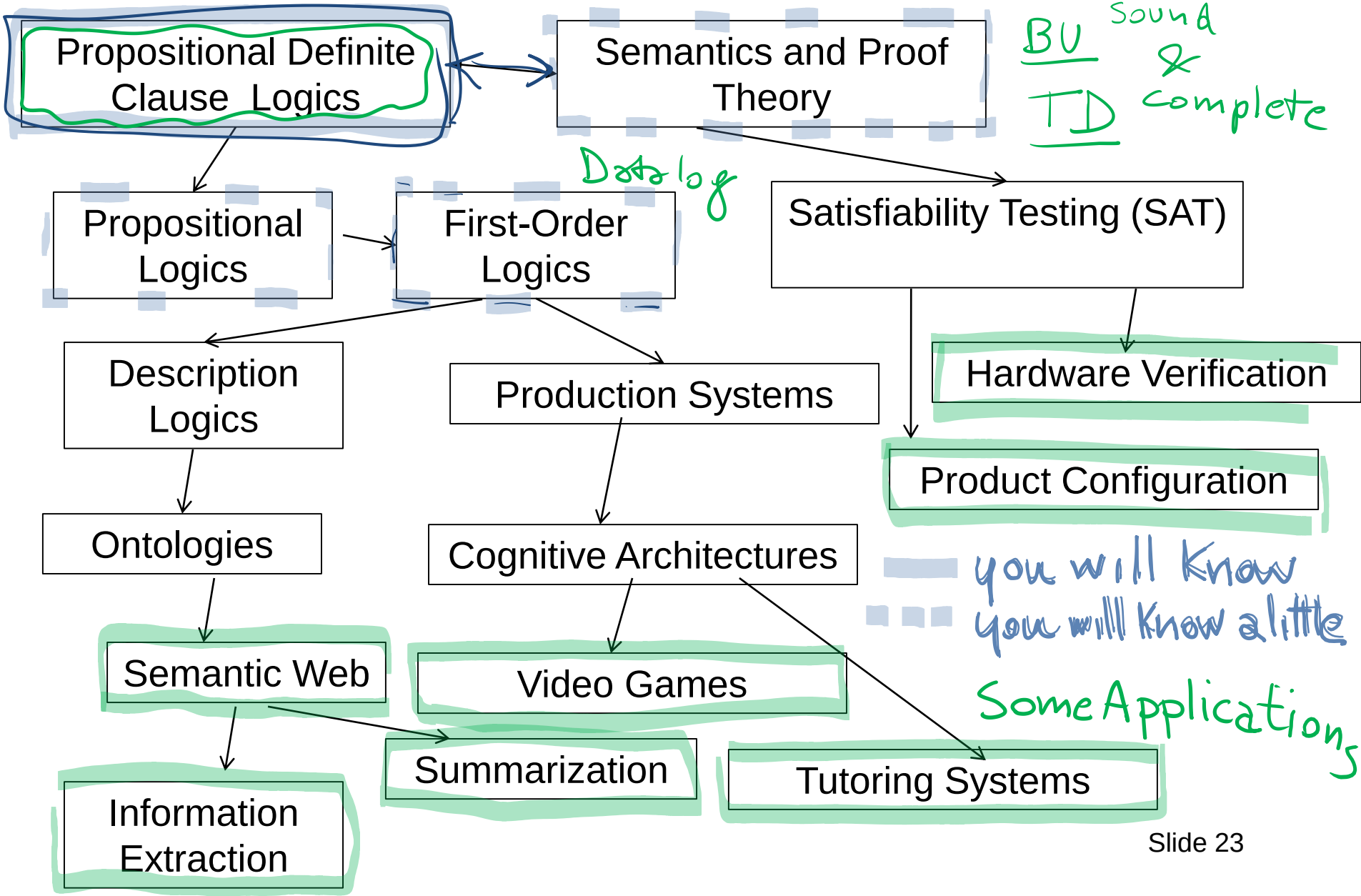
What would the answer(s) be?

yes(r123).

yes(cs_building).

Again, you can trace the SLD derivation for this query
in the AIspace Deduction Applet

Logics in AI


Propositional Definite Clause Logics
Semantics and Proof Theory
BU
TD
sound & complete
Propositional Logics
First-Order Logics
Datalog
Satisfiability Testing (SAT)
Description Logics
Production Systems
Hardware Verification
Product Configuration
Ontologies
Cognitive Architectures
you will know
you will know a little
Some Applications
Semantic Web
Video Games
Summarization
Tutoring Systems
Information Extraction

R&R systems we'll cover in this course

Problem		Environment: Deterministic	Environment: Stochastic
Static	Constraint Satisfaction	*Variables + Constraints* Search Arc Consistency Local Search	
	Query	*Logics* Search	*Bayesian (Belief) Networks* Variable Elimination
Sequential	Planning	*STRIPS* Search	*Decision Networks* Variable Elimination

Representation
Reasoning Technique

Next Up

Intro to probability

- Random Variable
- Prob. Distribution
- Marginalization
- Conditional Probability
- Chain Rule
- Bayes' Rule
- Marginal and Conditional Independence

Full Propositional Logics (not for 322)

DEFs.

Literal: an atom or a negation of an atom

Clause: is a disjunction of literals

Conjunctive Normal Form (CNF): a conjunction of clauses

INFERENCE:

- **Convert all formulas in KB and $\neg\alpha$ in CNF**
- **Apply Resolution Procedure (at each step combine two clauses containing complementary literals into a new one)**
- **Termination**
  - **No new clause can be added**
  - **Two clause resolve into an empty clause**

Propositional Logics: Satisfiability (SAT problem)

Does a set of formulas have a model? Is there an interpretation in which all the formulas are true?

(Stochastic) Local Search Algorithms can be used for this task**!**

Evaluation Function: number of unsatisfied clauses

WalkSat: One of the simplest and most effective algorithms:

Start from a randomly generated interpretation

- Pick an unsatisfied clause
- Pick an proposition to flip (randomly 1 or 2)
  1. To minimize # of unsatisfied clauses
  2. Randomly

Full First-Order Logics (FOLs)

We have constant symbols, predicate symbols and function symbols

So interpretations are much more complex (but the same basic idea – one possible configuration of the world)

constant symbols => individuals, entities

predicate symbols => relations

function symbols => functions

INFERENCE:

- **Semidecidable:** algorithms exists that says yes for every entailed formulas, but no algorithm exists that also says no for every non-entailed sentence
- **Resolution Procedure can be generalized to FOL**