Logic: TD as search, Datalog (variables)

Computer Science cpsc322, Lecture 23

(Textbook Chpt 5.2 &

some basic concepts from Chpt 12)

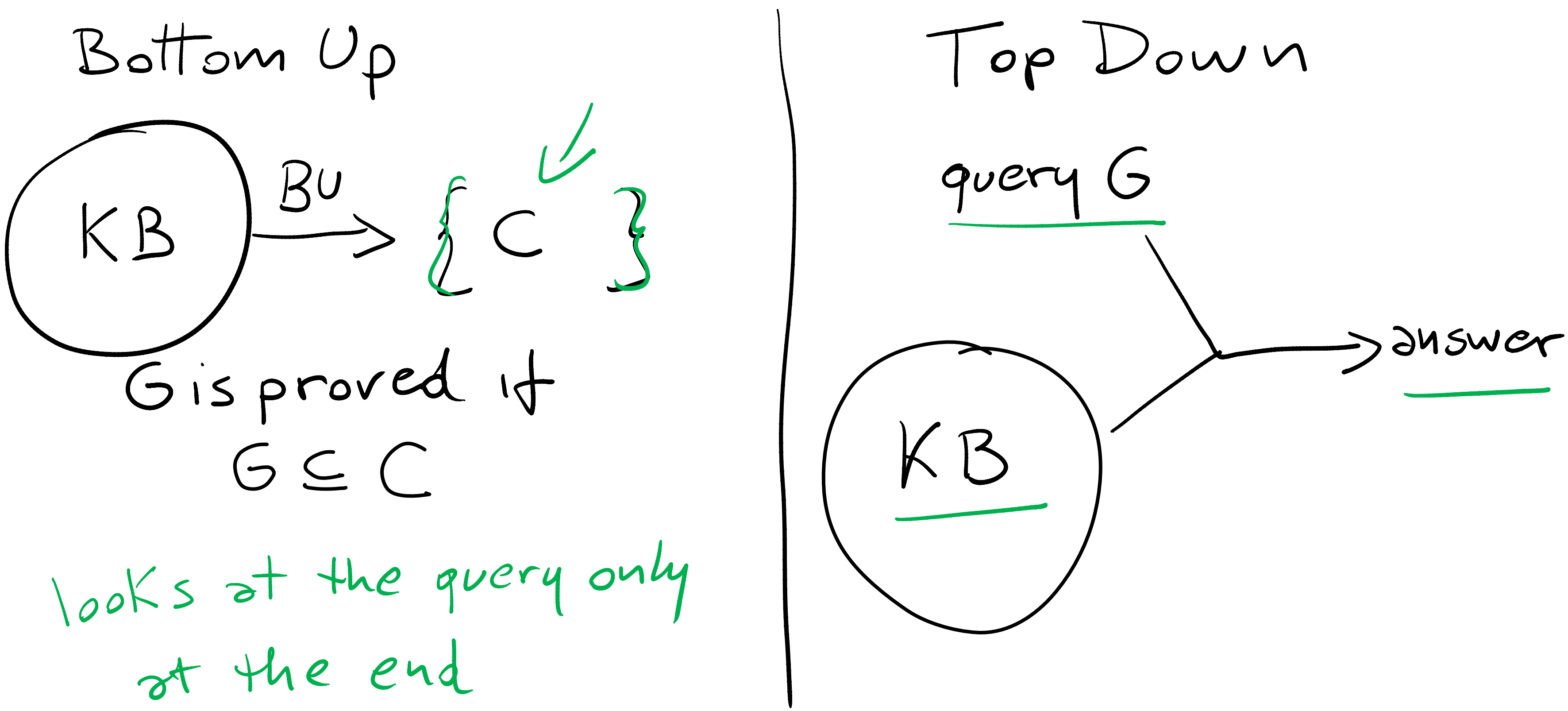
Oct, 31, 2012

Lecture Overview

- Recap Top Down
- TopDown Proofs as search
- Datalog

Top-down Ground Proof Procedure

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



Top-down Proof Procedure: Basic 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 $a_1 \wedge a_2 \wedge \ldots \wedge a_m$)

$yes \leftarrow$

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 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 ← *e* ∧ *f.*	*a* ← *b* ∧ *c.*	*b* ← *k* ∧ *f.*
c ← *e.*	*d* ← *k.*	*e.*
f ← *j* ∧ *e.*	*f* ← *c.*	*j* ← *c.*

Query: a `(two ways)`

yes ← *a.*

yes ← *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)

start state: query as an answer clause

Logical Inference (top Down)

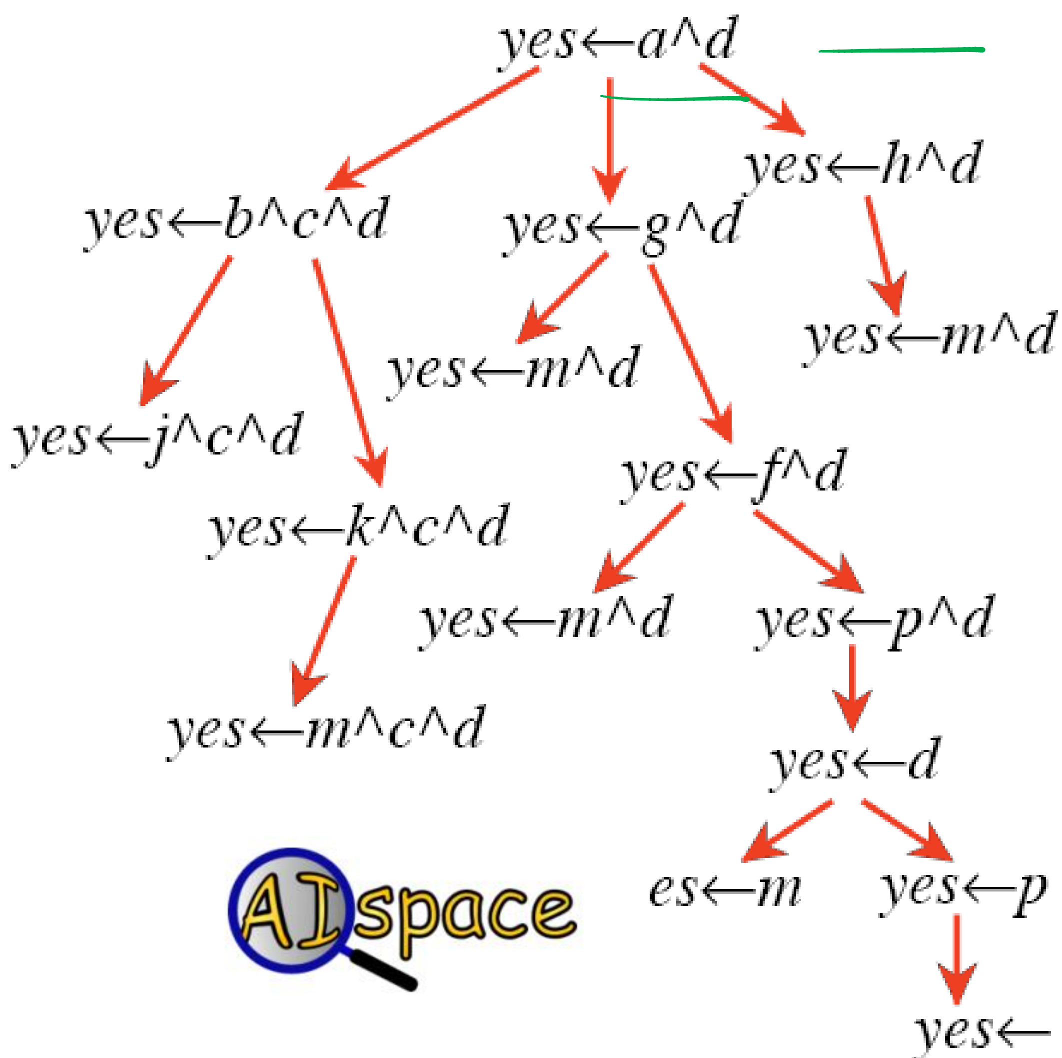
- State answer clause yes ←
- Successor function states resulting from substituting one atom with all the clauses of which it is the head
- Goal test empty answer clause yes ←
- Solution start state
- Heuristic function ………………… ✓ see next slide

Search Graph

KB

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

Prove: ?← $a \wedge d.$



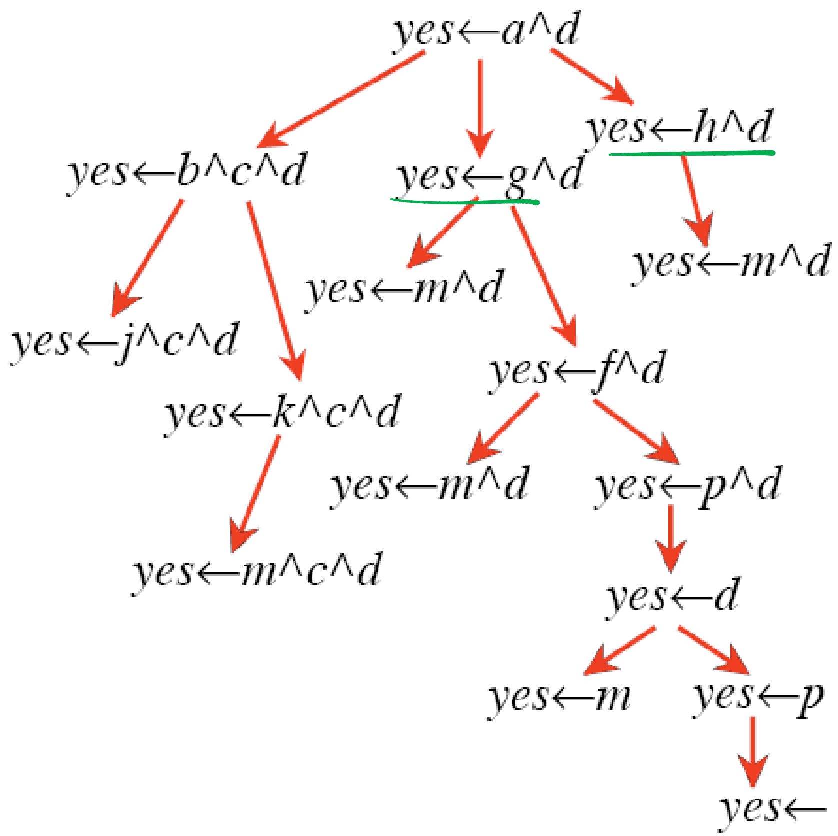
Heuristics?

Search Graph

KB

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

Prove: ? $\leftarrow a \wedge d.$



Possible Heuristic?

Number of atoms in the answer clause

Admissible?

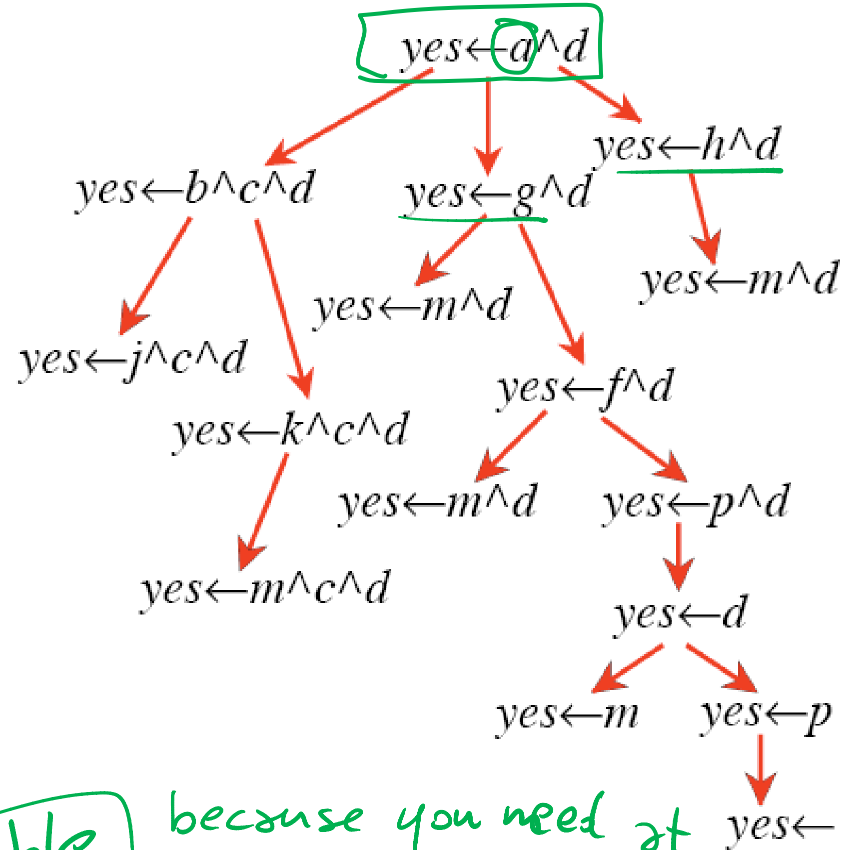
Yes No

Search Graph

Prove: ?← a ∧ d.

KB

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.



Heuristics?

Admissible

# of atoms in answer clause

because you need at least that number of resolution steps to obtain yes← i.e. the goal state

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
$live\_w_1$
$connected\_w_1\_w_2$

p q r s

- 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 the system can reason about

up_s_2
up_s_3

up are about the same property

$live\_w_1$
$connected\_w_1\_w_2$

w_1 are about the same individual

What do we gain….

By breaking propositions into relations applied to individuals?

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

  live(W) <- connected_to(W,W1) ∧ live(W1) ∧ wire(W) ∧ wire(W1).

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

  ? connected_to(W, w_1)

Datalog vs PDCL (better with colors)

First Order Logic

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

$p(a_1, a_2)$

$\neg q(a_5)$

Datalog

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

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

$s(a_1), q(a_2)$

Propositional Logic

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

p, r

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 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).

- We will not cover the formal details of this process, called ***unification***. See P&M Section 12.4.2, p. 511 for the details.

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

yes ← in(alan, r123).

Using clause: in(alan, r123).

yes ←.

Using clause: in(X,Y) ← part_of(Z,Y) & in(X,Z). With X = alan, Y = r123

yes ← part_of(Z, r123), in(alan, Z).

No clause with matching head: part_of(Z,r123).

fail

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 asks 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).

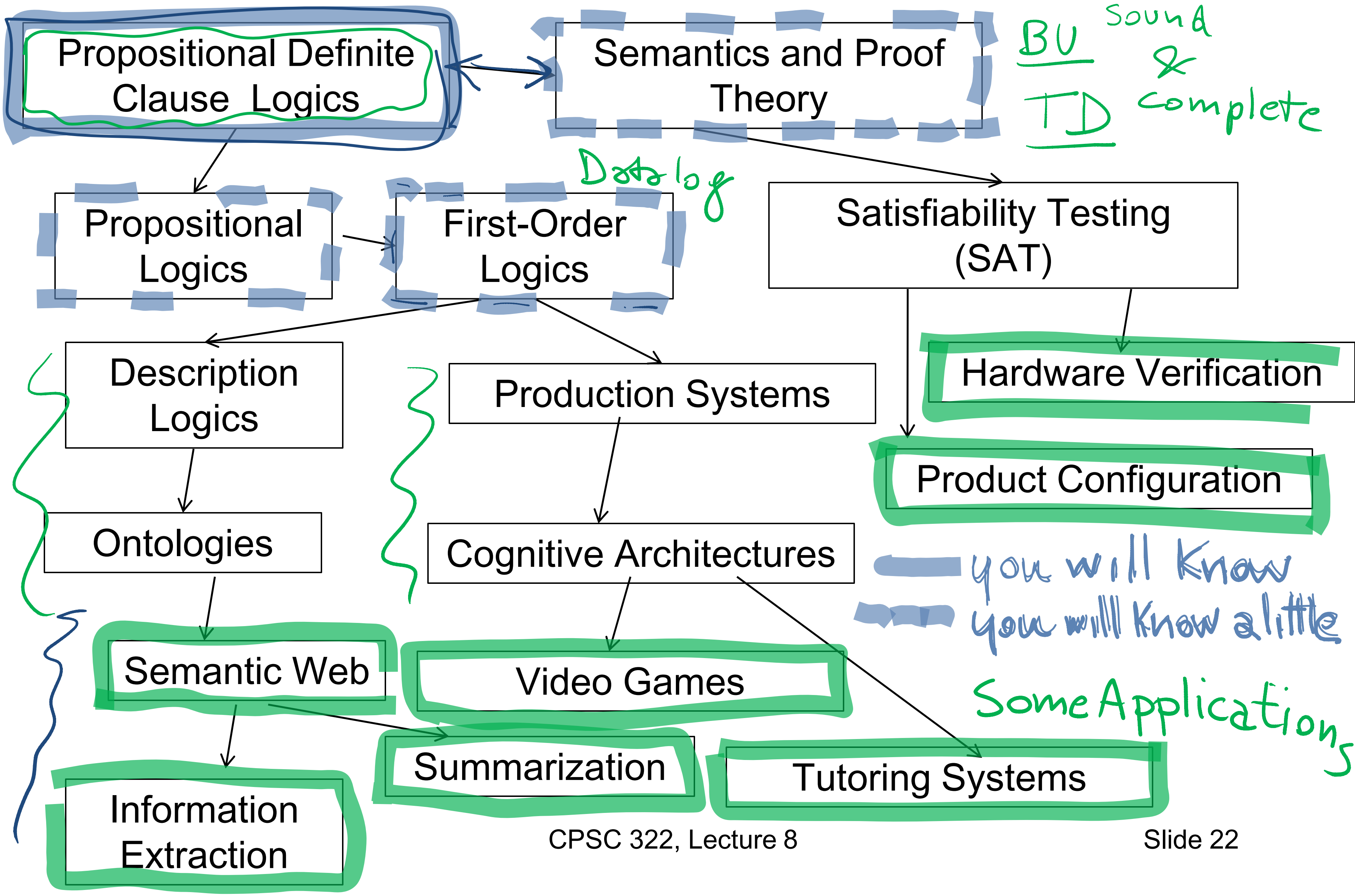
Query: in(alan, X1).
yes(X1) ← in(alan, X1).

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: Similar slide to the one for planning



Big Picture: R&R systems

Problem		Environment: Deterministic	Environment: Stochastic
Static	Constraint Satisfaction	*Vars + Constraints*: Arc Consistency, Search (→ for CSP), SLS	
Static	Query	*Logics*: Search (→ for Inference) — CSP	*Belief Nets*: Var. Elimination
Sequential	Planning	*STRIPS* → CSP; Search (→ for complex planning)	*Decision Nets*: Var. Elimination; *Markov Processes*: Value Iteration

Representation

Reasoning Technique

Midterm review

Average 73.4

Best 107 !

20 students > 90%

14 students <50%

How to learn more from midterm

- Carefully examine your mistakes (and our feedback)
- If you still do not see the correct answer/solution go back to your notes, the slides and the textbook
- If you are still confused come to office hours with specific questions

Full Propositional Logics

DEFs.

Literal: an atom or a negation of an atom $p \quad \neg q \quad r$

Clause: is a disjunction of literals $p \vee \neg r \vee q$

Conjunctive Normal Form (CNF): a conjunction of clauses $(p) \wedge (q \vee \neg r) \wedge (\neg q \vee p)$

INFERENCE: $KB \overset{?}{\models} \alpha$ ← formula

- **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)** $p \vee q \quad r \vee \neg q \rightarrow p \vee r$
- **Termination**
  - **No new clause can be added** $KB \nvdash \alpha$
  - **Two clause resolve into an empty clause** $KB \vdash \alpha$

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