Logic: Top-down proof procedure, Datalog, Big Picture

Alan Mackworth

UBC CS 322 - Logic 4

March 8, 2013

Textbook §5.2, §12.3

Lecture Overview

- Recap: Bottom-up proof procedure is sound and complete
- Top-down Proof Procedure
- Datalog
- Logics: Big Picture

Logical consequence and BU proofs

Definition (logical consequence)
If KB is a set of clauses and g is a conjunction of atoms,
g is a logical consequence of KB, written KB $\models$ g,
if g is true in every model of KB

Example: KB = {$h \leftarrow a$, a, $a \leftarrow c$}. Then KB $\models$?

	a	c	h	$h \leftarrow a$	a	$a \leftarrow c$	Model of KB
I_1	F	F	F	T	F	T	no
I_2	F	F	T	T	F	T	no
I_3	F	T	F	T	F	F	no
I_4	F	T	T	T	F	F	no
I_5	T	F	F	F	T	T	no
I_6	T	F	T	T	T	T	yes
I_7	T	T	F	F	T	T	no
I_8	T	T	T	T	T	T	yes

Which atoms are entailed?

Logical consequence and BU proofs

Definition (logical consequence)
If KB is a set of clauses and g is a conjunction of atoms,
g is a logical consequence of KB, written KB ⊧ g,
if g is true in every model of KB

Example: KB = {h ← a, a, a ← c}. Then KB ⊧ ?

	a	c	h	h ← a	a	a ← c	Model of KB
I_1	F	F	F	T	F	T	no
I_2	F	F	T	T	F	T	no
I_3	F	T	F	T	F	F	no
I_4	F	T	T	T	F	F	no
I_5	T	F	F	F	T	T	no
I_6	T	F	T	T	T	T	yes
I_7	T	T	F	F	T	T	no
I_8	T	T	T	T	T	T	yes

Which atoms are entailed?

KB ⊧ a and
KB ⊧ h

Logical consequence and BU proofs

Definition (logical consequence)
If KB is a set of clauses and g is a conjunction of atoms,
g is a logical consequence of KB, written KB $\models$ g,
if g is true in every model of KB

Example: KB = {h $\leftarrow$ a, a, a $\leftarrow$ c}. Then KB $\models$ a and KB $\models$ h.

BU proof procedure

C := {};
repeat
select clause h $\leftarrow b_1 \wedge \ldots \wedge b_m$ in KB
such that $b_i \in$ C for all i, and h $\notin$ C;
C := C $\cup$ {h}
until no more clauses can be selected. KB $\vdash_{BU}$ g if and only if g $\in$ C

What does BU derive for the KB above?

Logical consequence and BU proofs

Definition (logical consequence)
If KB is a set of clauses and g is a conjunction of atoms,
g is a logical consequence of KB, written KB $\models$ g,
if g is true in every model of KB

Example: KB = {$h \leftarrow a$, a, $a \leftarrow c$}. Then KB $\models$ a and KB $\models$ h.

BU proof procedure

C := {};
repeat
select clause $h \leftarrow b_1 \wedge \ldots \wedge b_m$ in KB
such that $b_i \in C$ for all i, and $h \notin C$;
C := C $\cup$ {h}
until no more clauses can be selected. KB $\vdash_{BU}$ g if and only if g $\in$ C

What does BU derive for the KB above?
Trace: {a}, {a,h}. Thus KB $\vdash_{BU}$ a and KB $\vdash_{BU}$ h.
Exactly the logical consequences!

Summary for bottom-up proof procedure BU

- Proved last time
  - BU is sound:
    it derives only atoms that logically follow from KB
  - BU is complete:
    it derives all atoms that logically follow from KB
- Together:
  it derives exactly the atoms that logically follow from KB !
  - That's why the results for $\models$ and $\vdash_{BU}$ matched for the example above
- And, it is efficient!
  - Outer loop linear in the number of clauses in KB
    - Each clause is used maximally once by BU

Learning Goals Up To Here

- PDCL syntax & 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 knowledge base
- 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

Lecture Overview

- Recap: Bottom-up proof procedure is sound and complete
- Top-down Proof Procedure
- Datalog
- Logics: Big Picture

Bottom-up vs. Top-down

Bottom-up

g is proved if $g \in C$

When does BU look at the query g?

In every loop iteration | Never

At the end | At the beginning

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*.

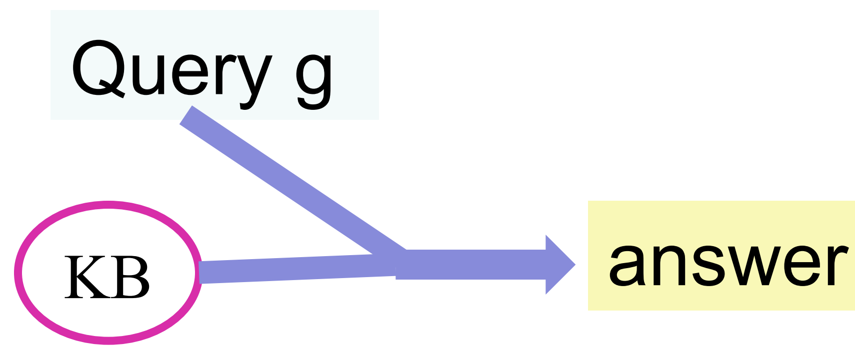
Bottom-up

g is proved if $g \in C$

When does BU look at the query g?

- Never
- It derives the same C regardless of the query

Top-down



We'll see how g is proved

TD performs a backward search starting at g

Top-down Ground Proof Procedure

Idea: search backward from a query

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

We express the query as an answer clause

- E.g. query $q_1 \wedge \ldots \wedge q_k$ is expressed as $yes \leftarrow q_1 \wedge \ldots \wedge q_k$

Basic operation: SLD Resolution of an answer clause

$$yes \leftarrow c_1 \wedge \ldots \wedge c_{i-1} \wedge c_i \wedge c_{i+1} \ldots \wedge c_m$$

on an atom c_i with another clause

$$c_i \leftarrow b_1 \wedge \ldots \wedge b_p$$

yields the clause

$$yes \leftarrow c_1 \wedge \ldots \wedge c_{i-1} \wedge b_1 \wedge \ldots \wedge b_p \wedge c_{i+1} \ldots \wedge c_m$$

Rules of derivation in top-down and bottom-up

Top-down:

SLD Resolution

$$\frac{yes \leftarrow c_1 \wedge \ldots \wedge c_{i-1} \wedge c_i \wedge c_{i+1} \ldots \wedge c_m \qquad c_i \leftarrow b_1 \wedge \ldots \wedge b_p}{yes \leftarrow c_1 \wedge \ldots \wedge c_{i-1} \wedge b_1 \wedge \ldots \wedge b_p \wedge c_{i+1} \ldots \wedge c_m}$$

Bottom-up:

Generalized modus ponens

$$\frac{h \leftarrow b_1 \wedge \ldots \wedge b_m \qquad b_1 \wedge \ldots \wedge b_m}{h}$$

Example for (successful) SLD derivation

$a \leftarrow b \wedge c.$

1 $\boxed{a \leftarrow e \wedge f.}$

$b \leftarrow f \wedge k.$

$c \leftarrow e.$

$d \leftarrow k$

3 $\boxed{e.}$

$f \leftarrow j \wedge e.$

2 $\boxed{f.}$

$j \leftarrow c.$

Query: ?a

γ_0: yes $\leftarrow$ a

γ_1: yes $\leftarrow$ e $\wedge$ f

γ_2: yes $\leftarrow$ e

γ_3: yes $\leftarrow$

Done. The question was "Can we derive a?"

The answer is "Yes, we can"

SLD Derivations

- An answer is an answer clause with m = 0.

  yes $\leftarrow$.

- A successful derivation from KB of query $?q_1 \wedge \ldots \wedge q_k$ is a sequence of answer clauses $\gamma_0, \gamma_1, \ldots, \gamma_n$ such that
  - γ_0 is the answer clause yes $\leftarrow q_1 \wedge \ldots \wedge q_k$.
  - γ_i is obtained by resolving γ_{i-1} with a clause in KB, and
  - γ_n is an answer yes $\leftarrow$

- An unsuccessful derivation from KB of query $?q_1 \wedge \ldots \wedge q_k$
  - We get to something like yes $\leftarrow b_1 \wedge \ldots \wedge b_k$, where there is no clause in KB with any of the b_i as its head

Top-down Proof Procedure for PDCL

To solve the query $? \, q_1 \wedge \ldots \wedge q_k$:

ac:= yes $\leftarrow$ body, where body is $q_1 \wedge \ldots \wedge q_k$

repeat

select $q_i \in$ body;

choose clause $C \in KB$, C is $q_i \leftarrow b_c$;

replace q_i in body by b_c

until ac is an answer (fail if no clause with q_i as head)

select: any choice will work

(“Don’t care non-determinism”)

choose: truly non-deterministic, must pick the right one

(“Don’t know non-determinism”)

Example for failing SLD derivation

$a \leftarrow b \wedge c.$

1 $\boxed{a \leftarrow e \wedge f.}$

$b \leftarrow f \wedge k.$

$c \leftarrow e.$

$d \leftarrow k$

3 $\boxed{e.}$

2 $\boxed{f \leftarrow k.}$

$f\,.$

$j \leftarrow c.$

Query: ?a

γ_0: yes $\leftarrow$ a

γ_1: yes $\leftarrow e \wedge f$

γ_2: yes $\leftarrow e \wedge k$

γ_3: yes $\leftarrow k$

"Can we derive a?"
"This time we failed"

There is no rule with k as its head, thus … fail

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 ← a.

yes ← b ∧ f.

yes ← b ∧ g ∧ h.

yes ← c ∧ d ∧ g ∧ h.

yes ← d ∧ g ∧ h.

yes ← g ∧ h.

yes ← h.

yes ← .

BU derivation

{}

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	BU derivation
yes ← a.	{}
yes ← b ∧ f.	{h}
yes ← b ∧ g ∧ h.	{g,h}
yes ← c ∧ d ∧ g ∧ h.	{d,g,h}
yes ← d ∧ g ∧ h.	{c,d,g,h}
yes ← g ∧ h.	{b,c,d,g,h}
yes ← h.	{b,c,d,f,g,h}
yes ← .	{a,b,c,d,f,g,h}

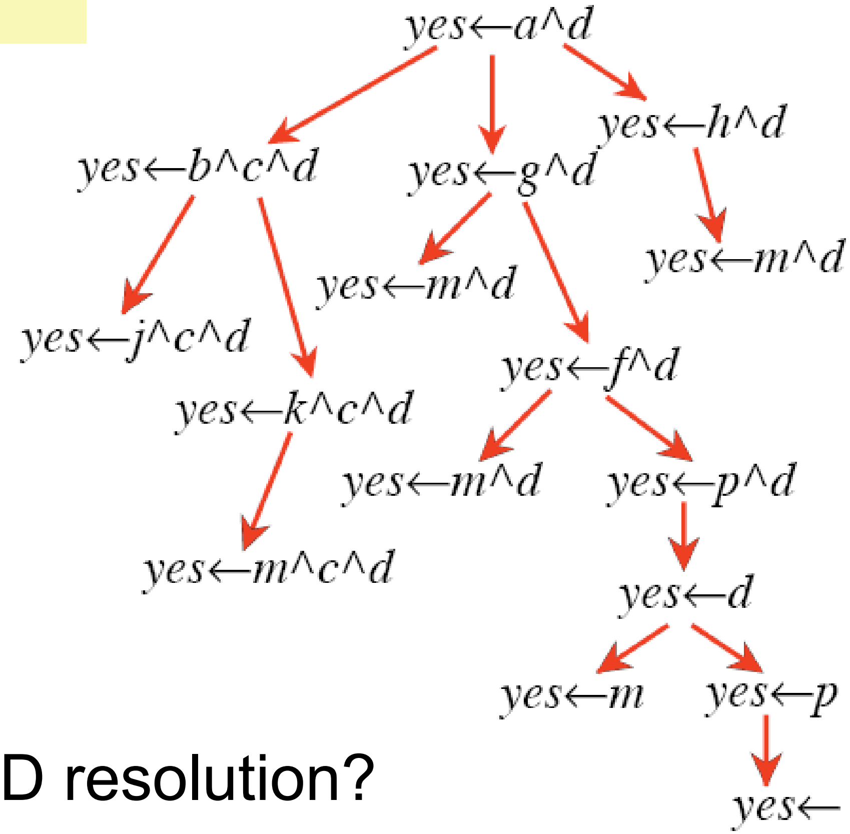
Is the Top-down procedure sound and complete?

- 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)

Search Graph for Top-down proofs

Query: ?$a \wedge d$.

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



What kind of search is SLD resolution?

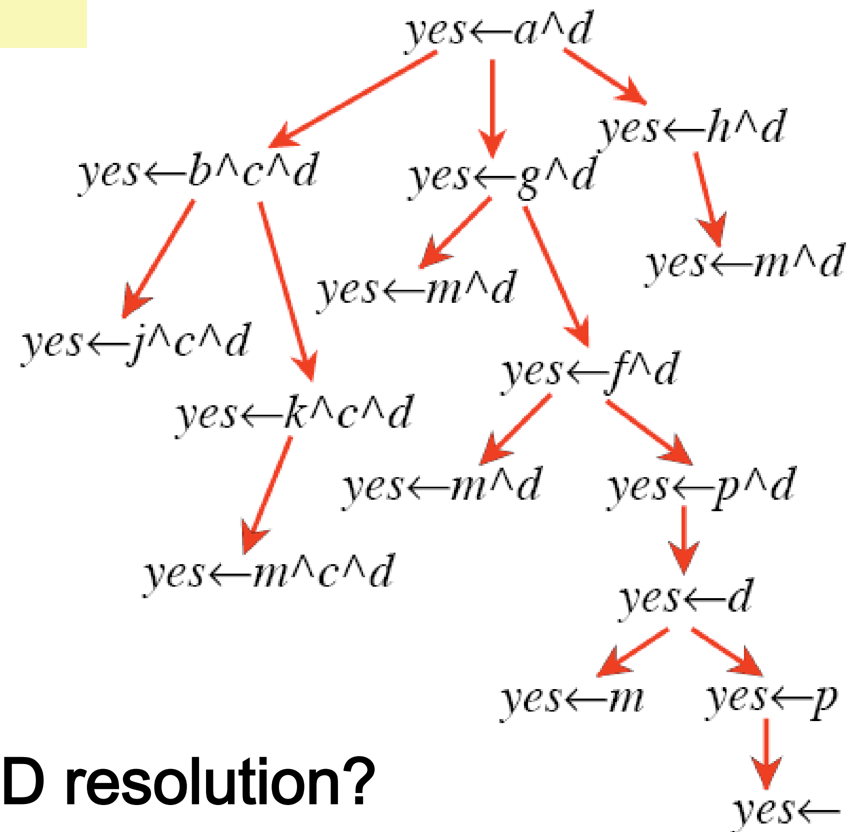
Breadth-first search

Depth-first-search

Search Graph for Top-down proofs

Query: ?$a \wedge d$.

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



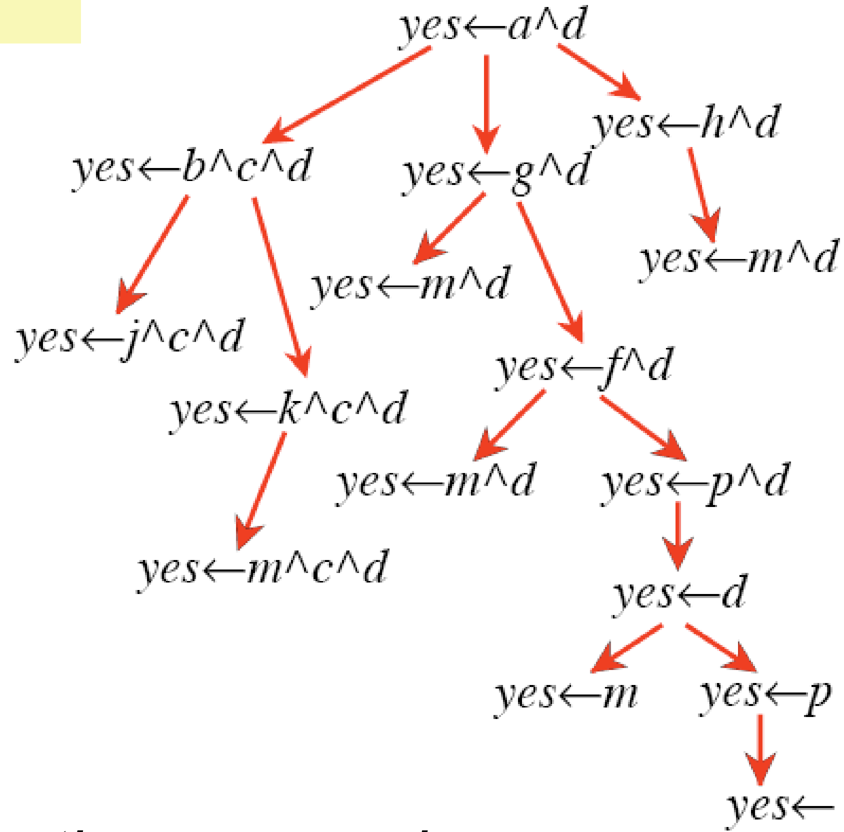
What kind of search is SLD resolution?

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

Search Graph for Top-down proofs

Query: ?$a \wedge d$.

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



Q: Can we use heuristics?

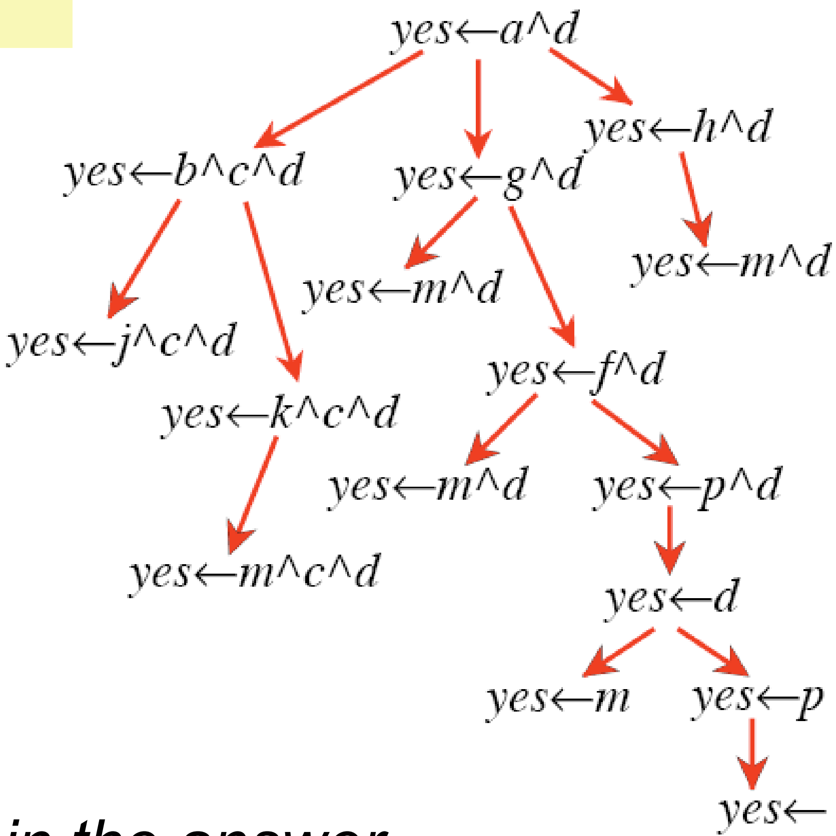
A: Yes! *E.g. number of atoms in the answer clause*

Admissible? Yes No

Search Graph for Top-down proofs

Query: ?$a \wedge d$.

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



Q: Can we use heuristics?

A: Yes! *E.g. number of atoms in the answer clause*

Admissible? *Yes, you need at least these many SLD steps to get an answer*

Inference as Standard Search

- 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 :
  - State: full assignment of values to features
  - Successor function: states reachable by applying valid actions
  - Goal test: partial assignment of values to features
  - Solution: a sequence of actions
  - Heuristic function: relaxed problem! E.g. “ignore delete lists”
- Inference (Top-down/SLD resolution)
  - State: answer clause of the form $yes \leftarrow q_1 \wedge \ldots \wedge q_k$
  - Successor function: all states resulting from substituting first atom a with $b_1 \wedge \ldots \wedge b_m$ if there is a clause $a \leftarrow b_1 \wedge \ldots \wedge b_m$
  - Goal test: is the answer clause empty (i.e. $yes \leftarrow$) ?
  - Solution: the proof, i.e. the sequence of SLD resolutions
  - Heuristic function: number of atoms in the query clause

Lecture Overview

- Recap: Bottom-up proof procedure is sound and complete
- Top-down Proof Procedure
- Datalog
- Logics: Big Picture

Representation and Reasoning in complex domains

- Expressing knowledge with **propositions** can be quite limiting

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

E.g. there is no notion that w_1 *is the same in* $live\_w_1$ and in $connected\_w_1\_w_2$

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

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

Now there is a notion that w_1 *is the same in* $live(w_1)$ and in $connected(w_1, w_2)$

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:

$$live(W) \leftarrow wire(W) \land connected\_to(W,W_1) \land wire(W_1) \land live(W_1).$$

  - 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 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 $p(t_1 \dots. t_n)$ 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:

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

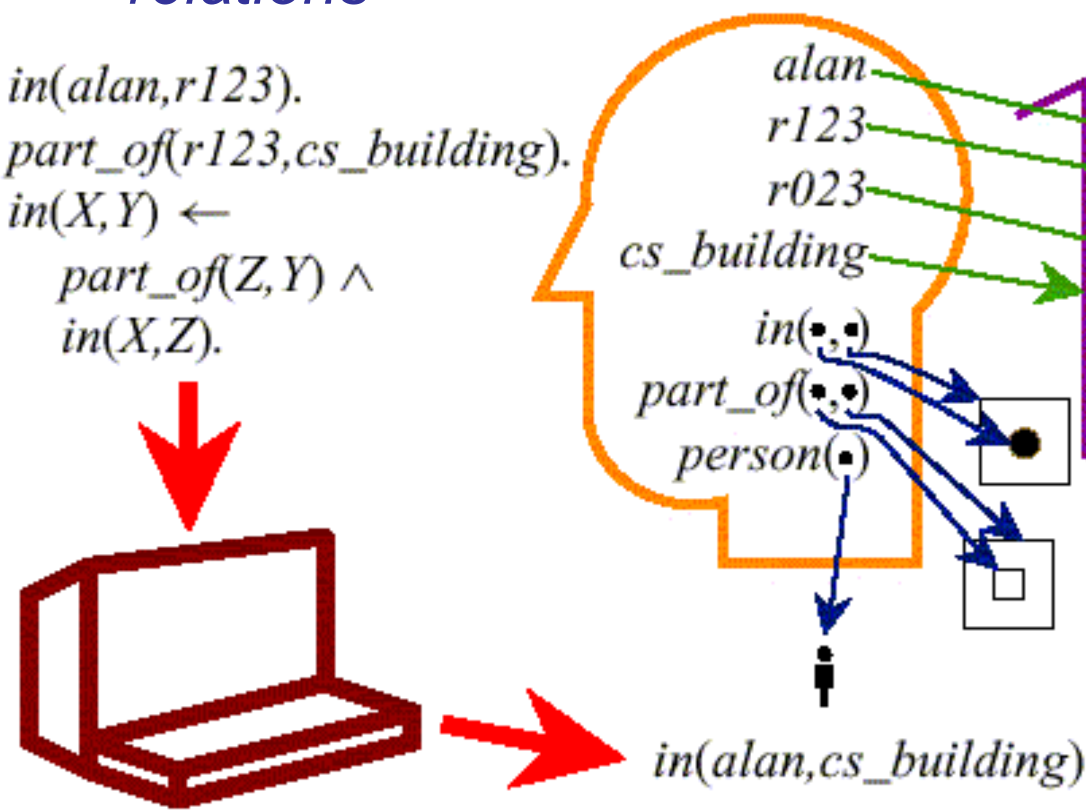
where h and the b_i are atoms (Read this as ``h if b.'')

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

A knowledge base is a set of definite clauses

Datalog Semantics

- Role of semantics is still to connect symbols and sentences in the language with the target domain. Main difference:
  - need to create correspondence both *between terms and individuals,* as well as *between predicate symbols and relations*



We won’t cover the formal definition of Datalog semantics, but if you are interested see 12.3.1 and 12.3.2 in textbook

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 clauses with heads that match the query
    - Substitute variable in the clause with the matching constant
  - Example:

Query: yes ← in(alan, cs_building).

↓

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

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

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).

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

yes ←.

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 http://cs.ubc.ca/~mack/CS322/in-part-of.pl

- 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).

You can trace the SLD derivation for this query in the AIspace Deduction Applet, using file http://cs.ubc.ca/~mack/CS322/in-part-of.pl

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)

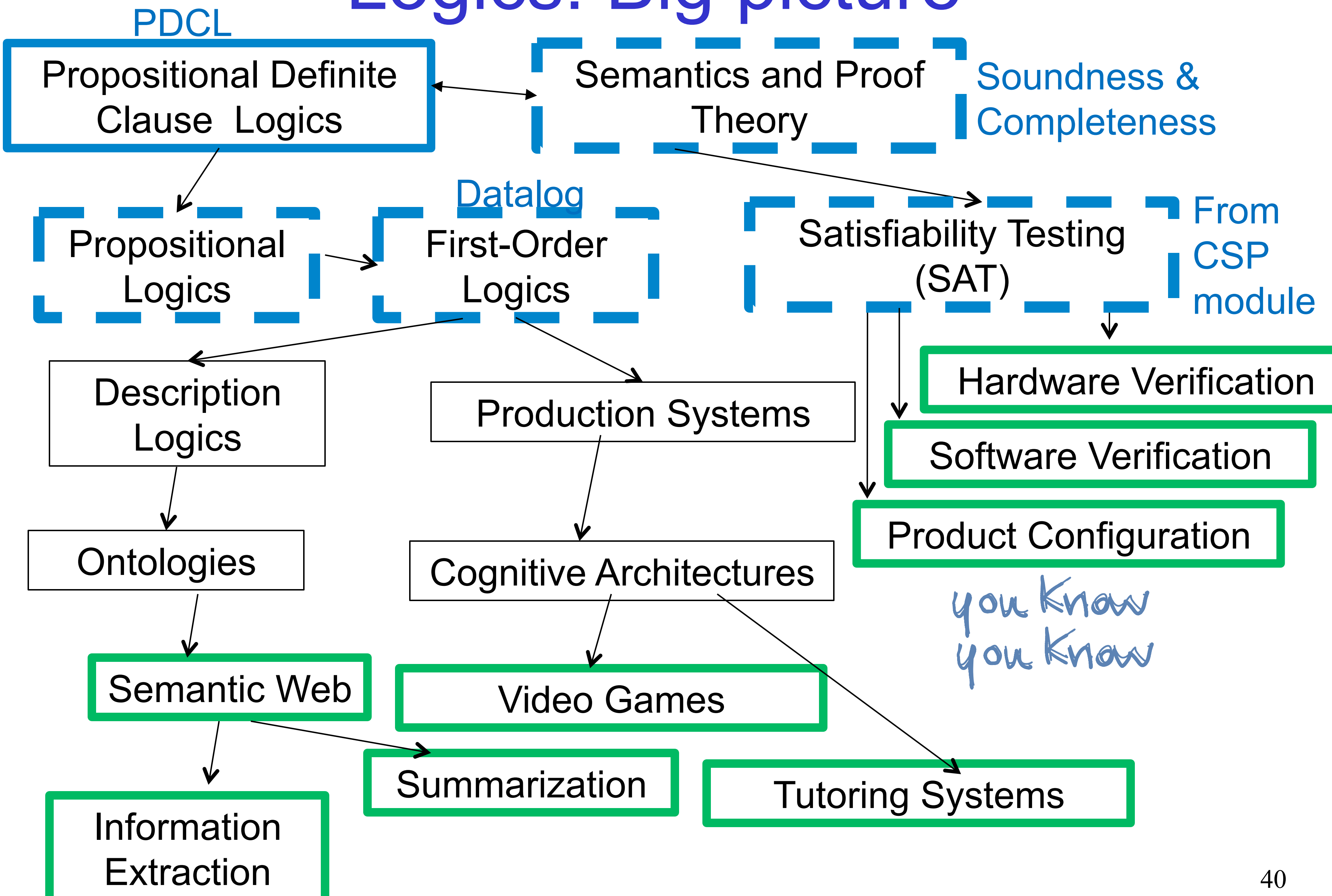
Learning Goals For Logic

- PDCL syntax & 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
- 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

Lecture Overview

- Recap: Bottom-up proof procedure is sound and complete
- Top-down Proof Procedure
- Datalog
- Logics: Big Picture

Logics: Big picture



Logics: Big picture

- 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, …
- There are many important applications of logic
  - For example, software agents roaming the web on our behalf
    - Based on a more structured representation: the semantic web
    - This is just one example for how logics are used

Example problem: 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
- If webpages are based on basic HTML
  - Humans need to scout for the information and integrate it
  - Computers are not reliable enough (yet?)
    - Natural language processing can be powerful (see Watson and Siri!)
    - But some information may be in pictures (beach), or implicit in the text, so simple approaches like Watson and Siri still don't get all the info.

More structured representation: the Semantic Web

- Beyond HTML pages only made for humans
- 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/
  (This is the best technical intro; Herman keeps it up to date.)

Examples of ontologies for the Semantic Web

- "Ontology": logic-based representation of the world
- eClassOwl: 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"

Course Overview