Introduction to Artificial Intelligence (AI)

Computer Science cpsc502, Lecture 6

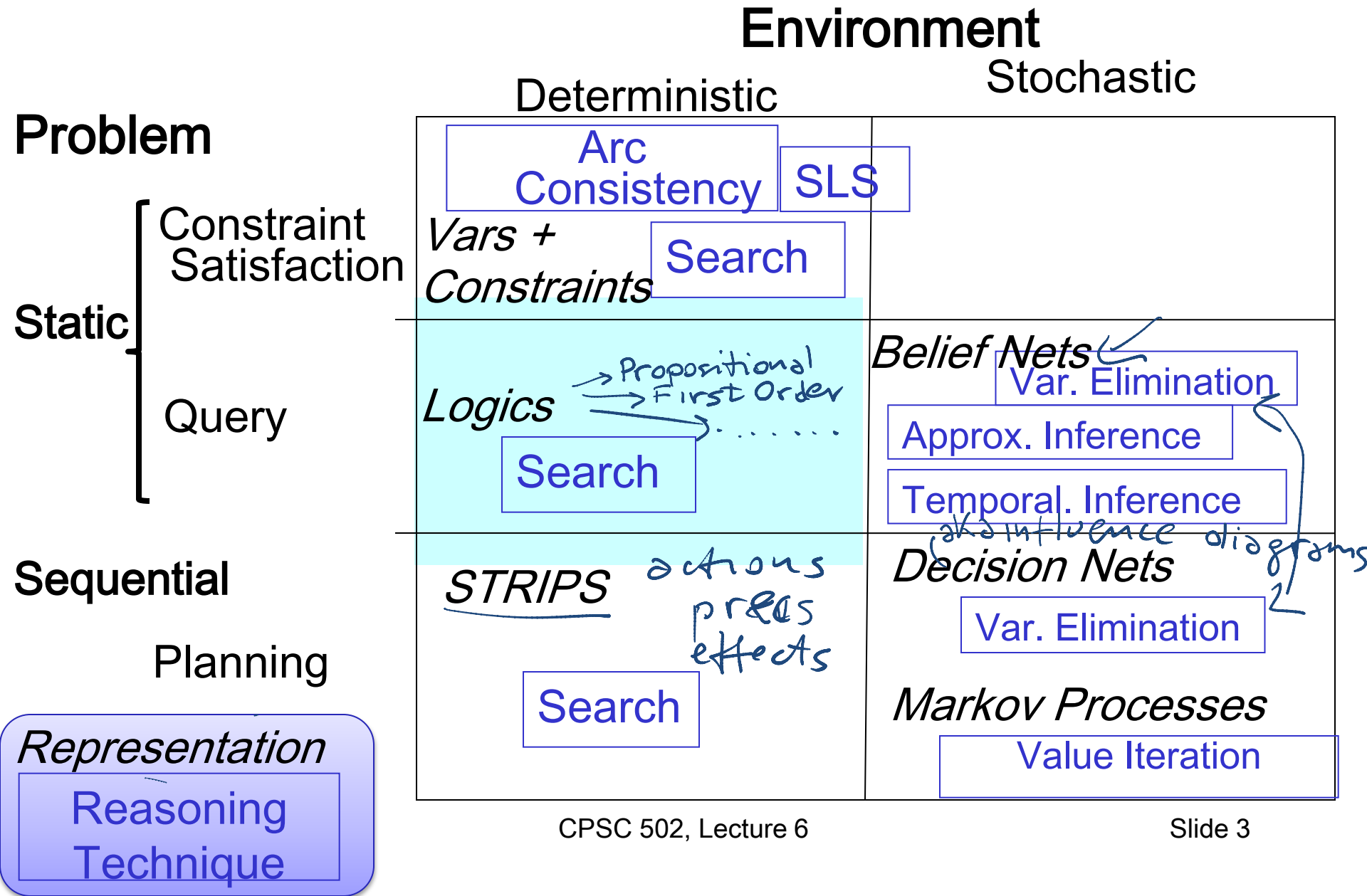
Sep, 27, 2011

Today Sept 27

Finish R&R systems in deterministic environments

- Logics
  - **Propositional, Definite Clause Logic: Syntax Semantics + Two different proof Procedures**
  - Reasoning with individuals and relations
  - Full Propositional Logics and First-Order Logics

R&Rsys we'll cover in this course



What you already know for sure about logic...

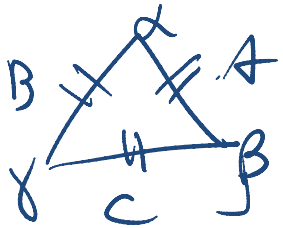
From programming: Some logical operators

```
If ((amount > 0) && (amount < 1000)) || !(age < 30)
...
```

AND ∧ OR ∨ NOT ~, ¬

You know what they mean in a "procedural" way

Logic is the language of Mathematics. To define formal structures (e.g., sets, graphs) and to proof statements about those



$\forall x \; triangle(x) \; A=B=C \iff \alpha=\beta=\gamma$

We are going to look at Logic as a **Representation and Reasoning System** that can be used to **formalize a domain (e.g., an electrical system, an organization)** and to **reason about it**

Why Logics?

- **"Natural"** to express **knowledge** about the world

(more natural than a "flat" set of variables & constraints)

"Every 502 student will pass the final"

Final (f_1)

Course(c_1)

Name-of(c_1, 502)

Course-of(f_1, c_1)

$\forall z$ Student(z) $\wedge$ Registred(z, c_1)

$\Rightarrow$ pass(f_1, z)

- It is easy to incrementally add knowledge
- It is easy to check and debug knowledge
- Provide language for asking complex queries
- Well understood formal properties

Ambitious Plan for today

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

Propositional Logic

A very simple form of Logics: **Propositional**

- The primitive elements are **propositions**: Boolean variables that can be {*true, false*}

p_1 p_2

- The goal is to illustrate the basic ideas: syntax, semantics, proof procedure

- This is a starting point for more complex logics (e.g., first-order logic)

- Boolean nature can be exploited for efficiency. So more complex logics are often mapped in propositional form for inference

Propositional logic: Complete Language

The **proposition** symbols p_1, p_2 … etc are sentences

- If S is a sentence, **¬S** is a sentence (negation)
- If S_1 and S_2 are sentences, $\mathbf{S_1 \wedge S_2}$ is a sentence (conjunction)
- If S_1 and S_2 are sentences, $\mathbf{S_1 \vee S_2}$ is a sentence (disjunction)
- If S_1 and S_2 are sentences, $\mathbf{S_1 \Rightarrow S_2}$ is a sentence (implication)
- If S_1 and S_2 are sentences, $\mathbf{S_1 \Leftrightarrow S_2}$ is a sentence (biconditional)

Sample Formula

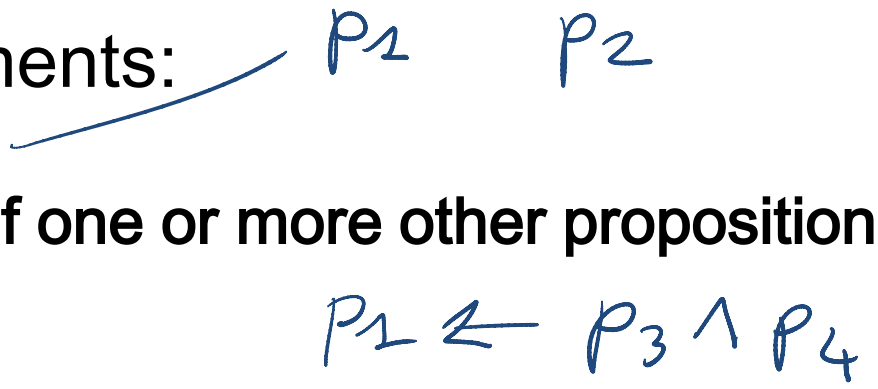
$$((p_1 \vee p_2) \wedge p_3) \Leftrightarrow ((p_2 \Rightarrow \neg p_4) \vee p_5)$$

Propositional Definite Clauses Logic

- **Propositional Definite Clauses:** our first logical representation and reasoning system.

(very simple!)

- Only two kinds of statements:
  - that **a proposition is true**
  - that **a proposition is true if one or more other propositions are true**



- Why still useful?
  - Adequate in many domains (with some adjustments)
  - Reasoning steps easy to follow by humans
  - Inference linear in size of your set of statements
  - Similar formalisms used in cognitive architectures

Propositional Definite Clauses: Syntax

Definition (atom)

An **atom** is a symbol starting with a lower case letter

Definition (body)

A **body** is an atom or is of the form $b_1 \wedge b_2$ where b_1 and b_2 are bodies.

Definition (definite clause)

A **definite clause** is an atom or is a rule of the form $h \leftarrow b$ where h is an atom and b is a body. (Read this as ``h if b.")

Definition (KB)

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

Propositional Definite Clauses Semantics: Interpretation

Semantics allows you to relate the symbols in the logic to the domain you're trying to model. An **atom** can be….. T F

Definition (interpretation)
An interpretation I assigns a truth value to each atom.

If your domain can be represented by four atoms (propositions):

q p s r
T T F F

2^4

So an interpretation is just a…possible world

PDC Semantics: Body

We can use the **interpretation** to determine the truth value of **clauses** and **knowledge bases**:

Definition (truth values of statements): A body $b_1 \wedge b_2$ is true in I if and only if b_1 is true in I and b_2 is true in I.

	p	q	r	s
I_1	*true*	*true*	*true*	*true*
I_2	*false*	*false*	*false*	*false*
I_3	*true*	*true*	*false*	*false*
I_4	*true*	*true*	*true*	*false*
I_5	*true*	*true*	*false*	*true*

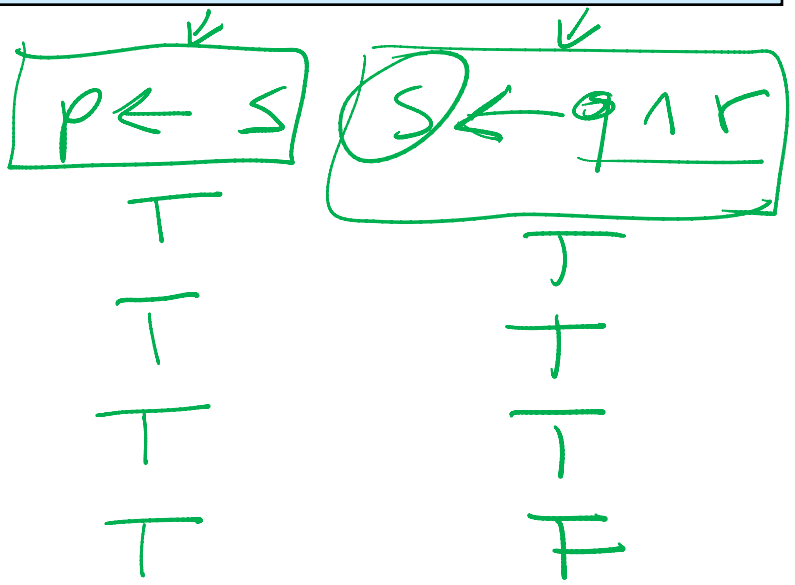
$p \wedge r$: T, F, F, T, F

$p \wedge r \wedge s$: T, F, F, F, F

PDC Semantics: definite clause

Definition (truth values of statements cont'): A rule $h \leftarrow b$ is false in I if and only if b is true in I and h is false in I.

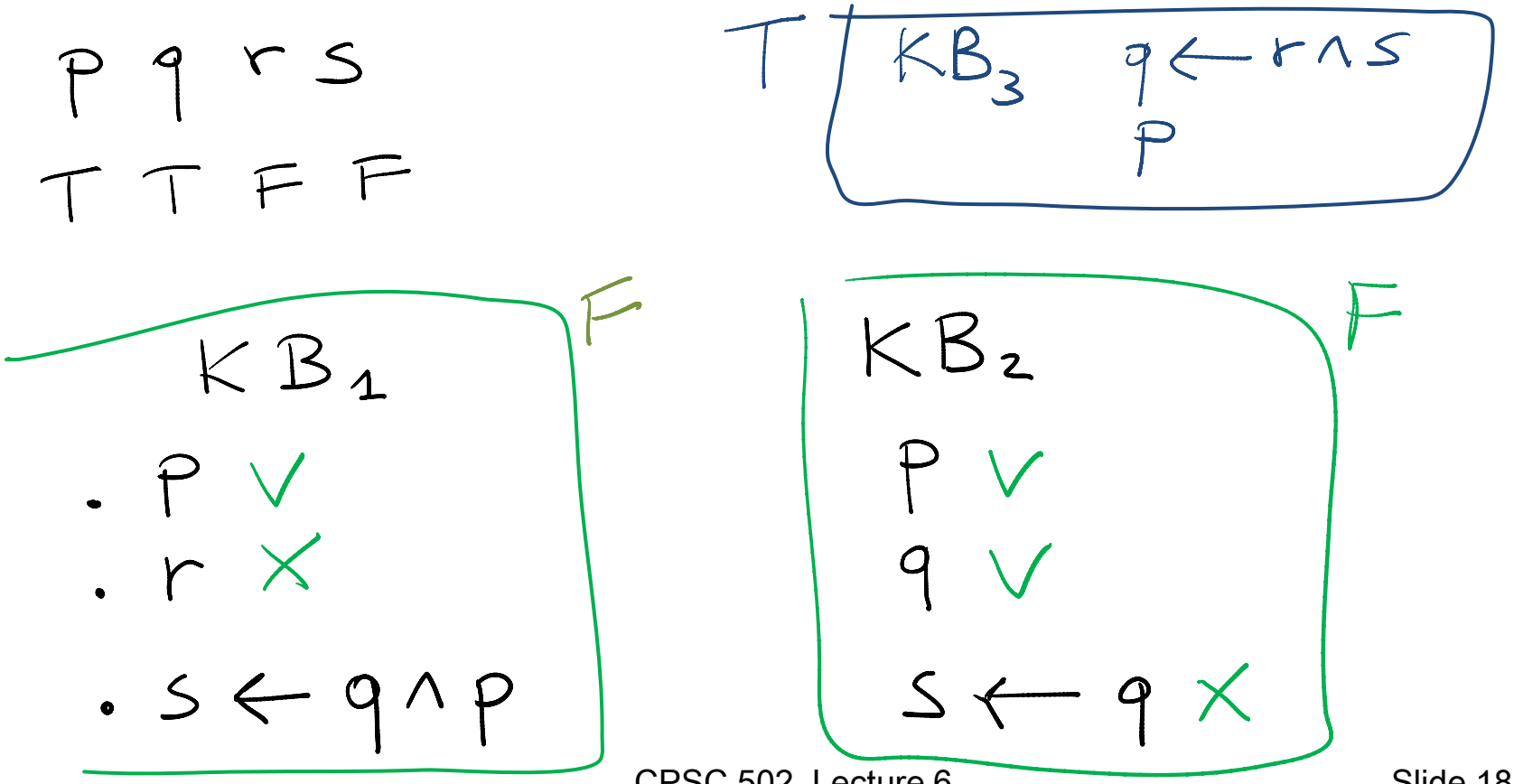
	p	q	r	s
I_1	*true*	*true*	*true*	*true*
I_2	*false*	*false*	*false*	*false*
I_3	*true*	*true*	*false*	*false*
I_4	*true*	*true*	*true*	*false*
…..	….	…..	….	….



In other words: *"if b is true I am claiming that h must be true, otherwise I am not making any claim"*

PDC Semantics: Knowledge Base

Definition (truth values of statements cont'): A knowledge base *KB* is true in I if and only if every clause in *KB* is true in I.



Models and Satisfiability

Definition (model)
A model of a set of clauses (a KB) is an interpretation in which all the clauses are *true*.

Definition (satisfiability)
A set of clauses (a KB) is satisfiable if it has at least one model

Example: Models

$$KB = \begin{cases} p \leftarrow q. \\ q. \\ r \leftarrow s. \end{cases}$$

	p	q	r	s	
I_1	*true*	*true*	*true*	*true*	M
I_2	*false*	*false*	*false*	*false*	×
I_3	*true*	*true*	*false*	*false*	M
I_4	*true*	*true*	*true*	*false*	M
I_5	*true*	*true*	*false*	*true*	×

Which interpretations are models?

Logical Consequence

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*.

- we also say that *G* logically follows from *KB*, or that *KB* entails *G*.
- In other words, $KB \models G$ if there is no interpretation in which *KB* is *true* and *G* is *false*.

Example: Logical Consequences

	p	q	r	s
I_1	*true*	*true*	*true*	*true*
I_2	*true*	*true*	*true*	*false*
I_3	*true*	*true*	*false*	*false*
I_4	*true*	*true*	*false*	*true*
I_5	*false*	*true*	*true*	*true*
I_6	*false*	*true*	*true*	*false*
I_7	*false*	*true*	*false*	*false*
I_8	*false*	*true*	*false*	*true*
...	....	...	...	...

Models

$$KB = \begin{cases} p \leftarrow q. \\ q. \\ r \leftarrow s. \end{cases}$$

$2^4 = 16$ interpretations in total, only 3 are models

remaining 8 cannot be models because q is false

F

Which of the following is true?

- $KB \models q$, $KB \models p$, $KB \not\models s$, $KB \not\models r$

One simple way to prove that G logically follows from a KB

- Collect all the models of the KB
- Verify that G is true in all those models

a set of atoms

P1, P2, ...

Any problem with this approach?

intractable time complexity

you have to check all the 2^n interpretations

- The goal of proof theory is to find proof procedures that allow us to prove that a logical formula follows form a KB avoiding the above

is logically entailed by

Soundness and Completeness

- If I tell you I have a **proof procedure for PDCL**
- What do I need to show you in order for you to trust my procedure?
  - KB $\vdash$ G means *G* can be derived by my proof procedure from *KB*.
  - Recall *KB* $\models$ *G* means *G* is true in all models of *KB*.

Definition (soundness)

A proof procedure is sound if *KB* $\vdash$ *G* implies *KB* $\models$ *G*.

Definition (completeness)

A proof procedure is complete if *KB* $\models$ *G* implies *KB* $\vdash$ *G*.

Bottom-up Ground Proof Procedure

One rule of derivation, a generalized form of ***modus ponens***:

If "$h \leftarrow b_1 \wedge \ldots \wedge b_m$" is a clause in the knowledge base, and each b_i has been derived, then h can be derived.

You are forward chaining on this clause.

(This rule also covers the case when $m=0$.)

Bottom-up proof procedure

$KB \vdash G$ if $G \subseteq C$ at the end of this 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.

Bottom-up proof procedure: Example

BU

$z \leftarrow f \wedge e$

$q \leftarrow f \wedge g \wedge z$

$e \leftarrow a \wedge b$

a

b

r

f

$C = \{t, r, b, a, e, z\}$

```
C :={};
repeat
    select clause "h ← b1 ∧ … ∧ bm" in KB such
                that bi ∈ C for all i, and h ∉ C;
    C := C ∪ { h }
until no more clauses can be selected.
```

BU can derive $r \wedge z$

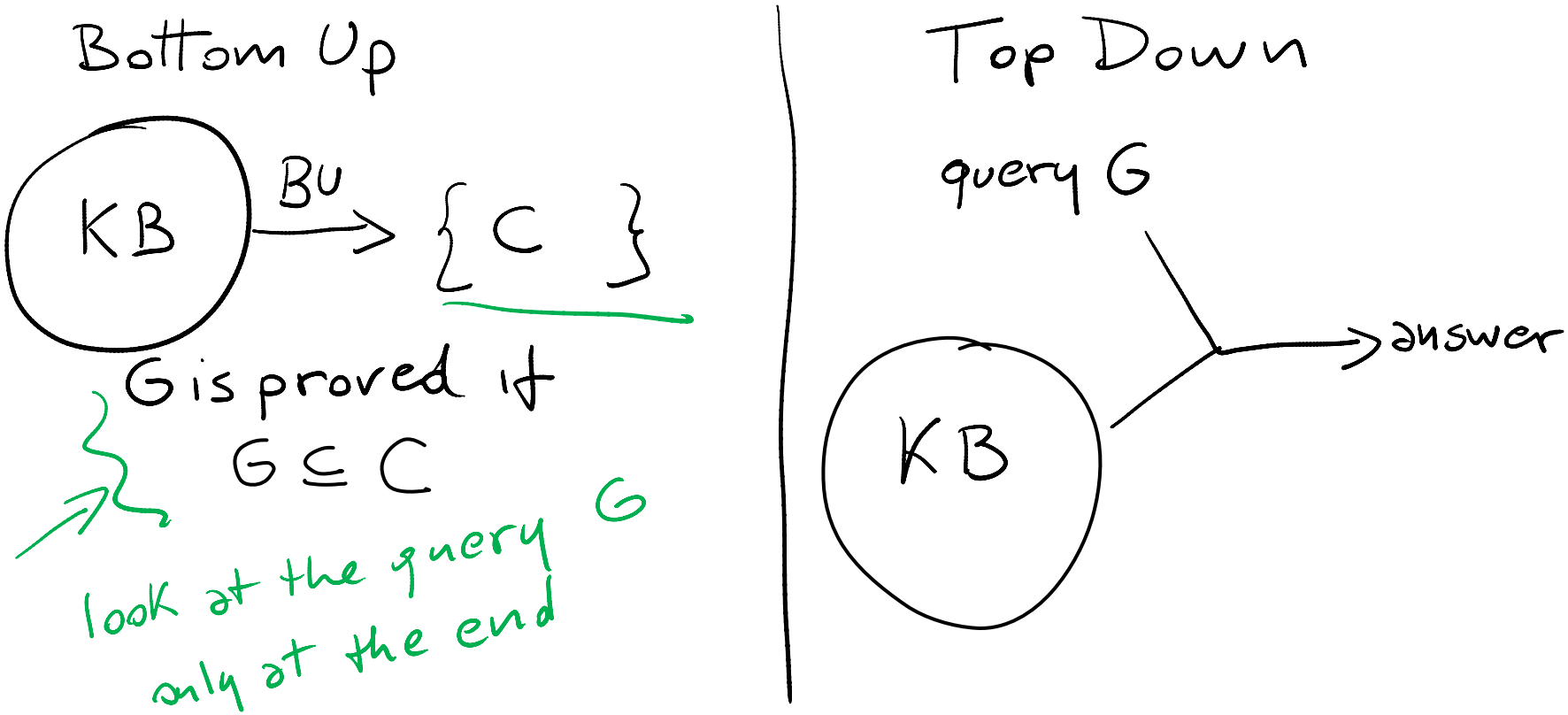
$KB \vdash_{BU} r?\ q?\ z?$

BU cannot derive

$KB \not\vdash_{BU} q$

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 $a_1 \wedge a_2 \wedge \ldots \wedge a_m$)

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

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 clause:

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

You can generate the answer clause

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

Rule of inference: Examples

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

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

$yes \leftarrow b \wedge c.$ $\quad b \leftarrow k \wedge f.$ $\Rightarrow yes \leftarrow k \wedge f \wedge c$

$yes \leftarrow e \wedge f.$ $\quad e \leftarrow$ $\Rightarrow yes \leftarrow f$

(successful) Derivations

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

- A (successful) derivation of query "$?q_1 \wedge \ldots \wedge q_k$" from KB 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…..

$yes \leftarrow a \wedge b$

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

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

KB

Query: a (two ways)

yes ← *a.*

$\leftarrow e \wedge f$

$\leftarrow f$

$\leftarrow c$

$\leftarrow e$

$\leftarrow$

yes ← *a.*

$\leftarrow b \wedge c$

$\leftarrow k \wedge f \wedge c$

$\leftarrow$

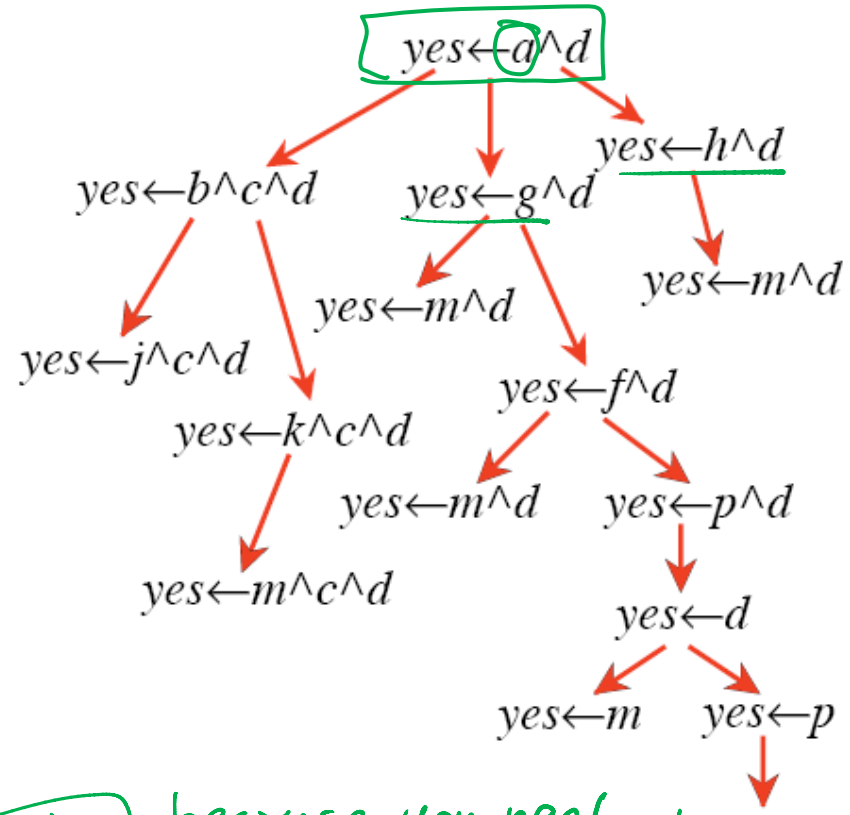
Fail

Search Graph

Prove: ? ← $a \wedge d$.

KB

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



Admissible

because you need at least that number of resolution steps to obtain yes←

Heuristics?

# of atoms in answer clause

i.e. the goal state

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 number of atoms in given state

Today Sept 27

Finish R&R systems in deterministic environments

- Logics
  - Propositional Definite Clause Logic: Syntax Semantics + Two different proof Procedures
  - **Reasoning with individual and relations**
  - Full Propositional Logics and First-Order Logics

Ambitious Plan for today

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

$p \leftarrow s \wedge f$

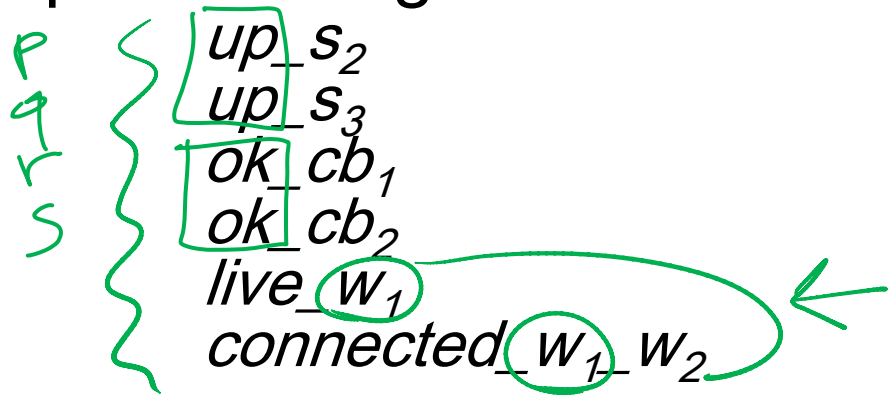
$r \leftarrow s \wedge q \wedge p$

r

p

Representation and Reasoning in Complex domains

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



- 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: a relational rule language

It expands the syntax of PDCL….

A variable is a symbol starting with an upper case letter

X Y

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

alan w1

A term is either a variable or a constant.

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

in part-of live

Datalog Syntax (cont')

so it includes propositions

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

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.")

in(X, Y) ← in(X, Z) ∧ part_of(Z, Y)

A knowledge base is a set of definite clauses

Datalog: Top Down Proof

Extension of TD for PDCL.

How do you deal with variables?

Example:

in(alan, r123).

part_of(r123,cs_building).

in(X,Y) <- part_of(Z,Y) & in(X,Z).

Query: in(alan, cs_building).

yes <- in(alan, cs_building).

Datalog: queries with variables

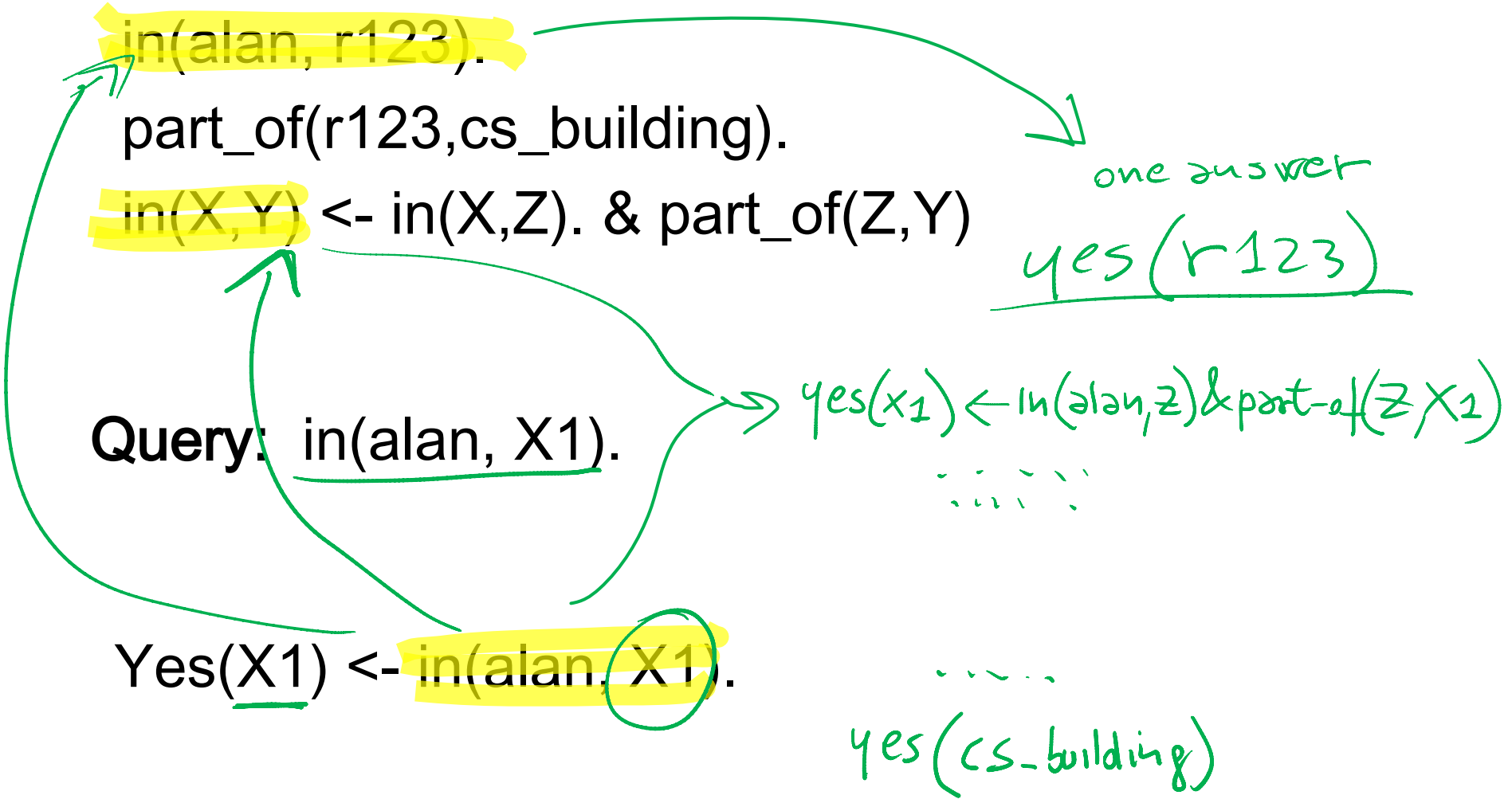
in(alan, r123).

part_of(r123,cs_building).

in(X,Y) <- in(X,Z). & part_of(Z,Y)

Query: in(alan, X1).

Yes(X1) <- in(alan, X1).



Ambitious Plan for today

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

Today Sept 27

Finish R&R systems in deterministic environments

- Logics
  - Propositional Definite Clause Logic: Syntax Semantics + Two different proof Procedures
  - Reasoning with individual and relations
  - Full Propositional Logics and First-Order Logics

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

INFERENCE: $KB \stackrel{?}{\models} \alpha$ formula $(p) \wedge (q \vee \neg r) \wedge (\neg q \vee p)$

- 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 $KB \not\vdash \alpha$
  - 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?

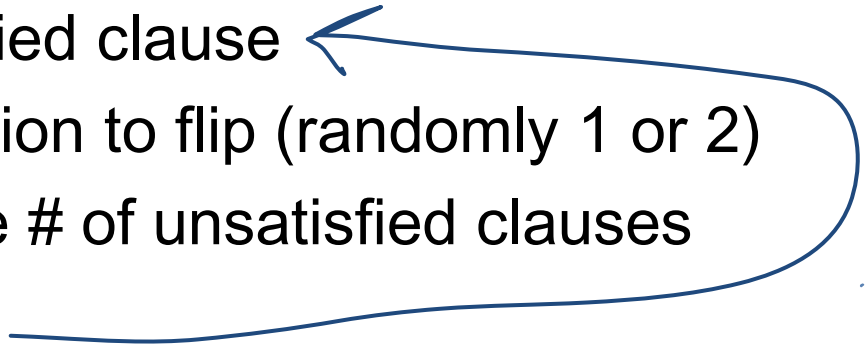
(Stochastic) Local Search Algorithms can be used for this task**!**

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

TODO for next Tue

Thurs class is canceled – I'll be visiting SAP labs in France for a workshop on Business Intelligence

This week Focus on

- Complete the assignment!
- Study carefully handout on Probability (critical for the rest of the lectures!)

Also Do exercises 5.A and 12.A,B,C

http://www.aispace.org/exercises.shtml

Expressiveness of the language

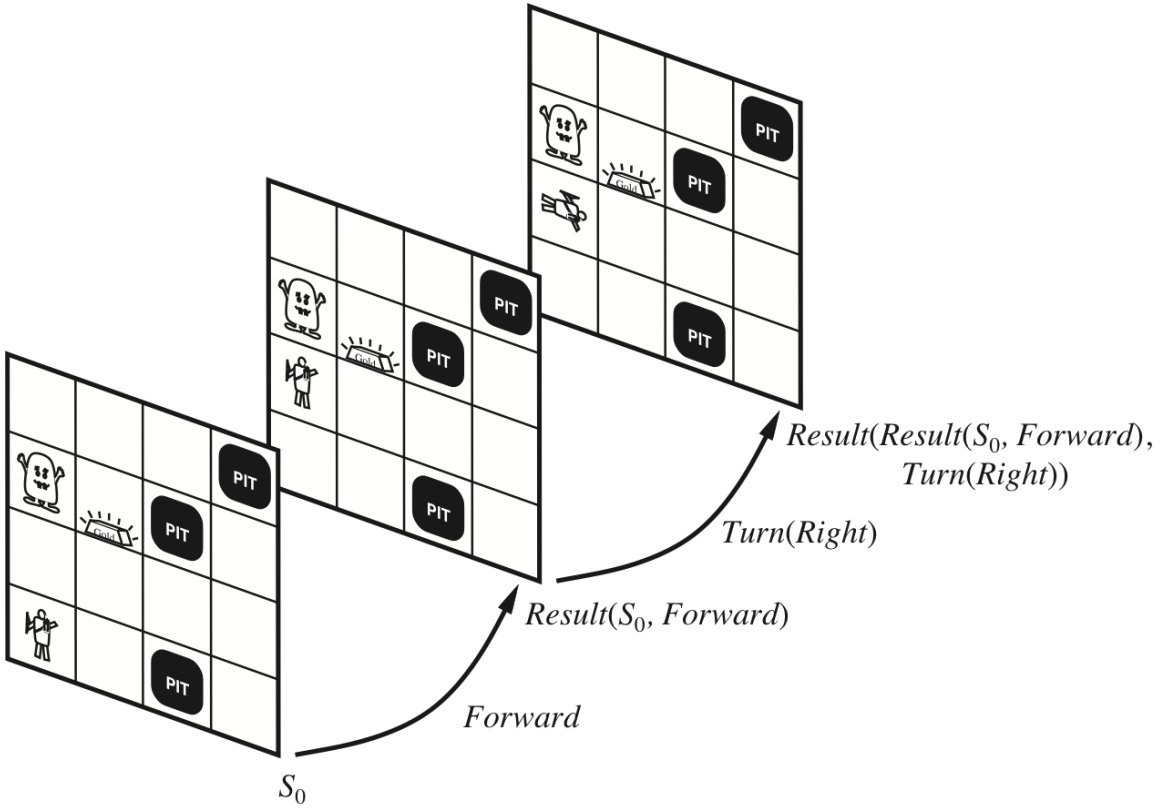
$Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK)$
$\quad \land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2)$
$\quad \land Airport(JFK) \land Airport(SFO))$
$Goal(At(C_1, JFK) \land At(C_2, SFO))$
$Action(Load(c, p, a),$
$\quad$ PRECOND: $At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
$\quad$ EFFECT: $\neg At(c, a) \land In(c, p))$
$Action(Unload(c, p, a),$
$\quad$ PRECOND: $In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
$\quad$ EFFECT: $At(c, a) \land \neg In(c, p))$
$Action(Fly(p, from, to),$
$\quad$ PRECOND: $At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)$
$\quad$ EFFECT: $\neg At(p, from) \land At(p, to))$

Figure 10.1 A PDDL description of an air cargo transportation planning problem.

Planning with Structured Rep

Situation Calculus

First-order Logic



Situation Calculus

- Convenient to have more expressive lang. “Move all the cargo from SFO to JFK”
- Use existing mechanisms for logical proof
- Strong foundation for studying planning
- Still, less used in practice than other techniques

Situation Calculus

- Possibility Axioms (for each action)
  - $SomeFormula(s) \Rightarrow Poss(a, s)$
  - $Alive(Agent, s) \wedge Have(Agent, Arrow, s) \Rightarrow Poss(Shoot, s)$
- Successor-state Axiom (for each fluent)
  - $Poss(a, s) \Rightarrow$ (*fluent is true* $\Leftrightarrow$ *a made it true* $\vee$ *it was true and a left it alone*)
  - $Poss(a, s) \Rightarrow (Holding(Agent, g, Result(s, a)) \Leftrightarrow a = Grab(g) \vee (Holding(Agent, g, s) \wedge a \neq Release(g)))$

CS 221: Artificial Intelligence

(Some) Slide credit: Peter Norvig and Sebastian Thrun

Dan Klein, Stuart Russell, Andrew Moore

Soundness & completeness of proof procedures

- A proof procedure X is sound …

$KB \vdash_X G \Rightarrow KB \models G$

- A proof procedure X is complete….

$KB \models G \Rightarrow KB \vdash_X G$

- BottomUp for PDCL is ← sound & complete

- We proved this in general even for domains represented by **thousands of propositions** and corresponding **KB with millions of definite clauses** !

Can you think of a proof procedure for PDCL….

A: C_A = {all clauses with empty bodies}

$KB \vdash_A G$ $G \in C_A \subseteq C_{BU}$

B: C_B = {all atoms of KB}

$KB \vdash_B G$ $G \in C_B$ $C_{BU} \subseteq C_B$

KB

$a \leftarrow e \wedge g.$
$b \leftarrow f \wedge g.$
$c \leftarrow e.$
$f \leftarrow c \wedge e.$
$e.$
$d.$

C_A

- That is sound but not complete?
- That is complete but not sound?

$KB \vdash_A G \Rightarrow KB \models G$

$\Rightarrow G \in C_A \subseteq C_{BU} \Rightarrow KB \vdash_{BU} G$

C_B {a b c d e f g}

$\Rightarrow KB \models G$

soundness of BU

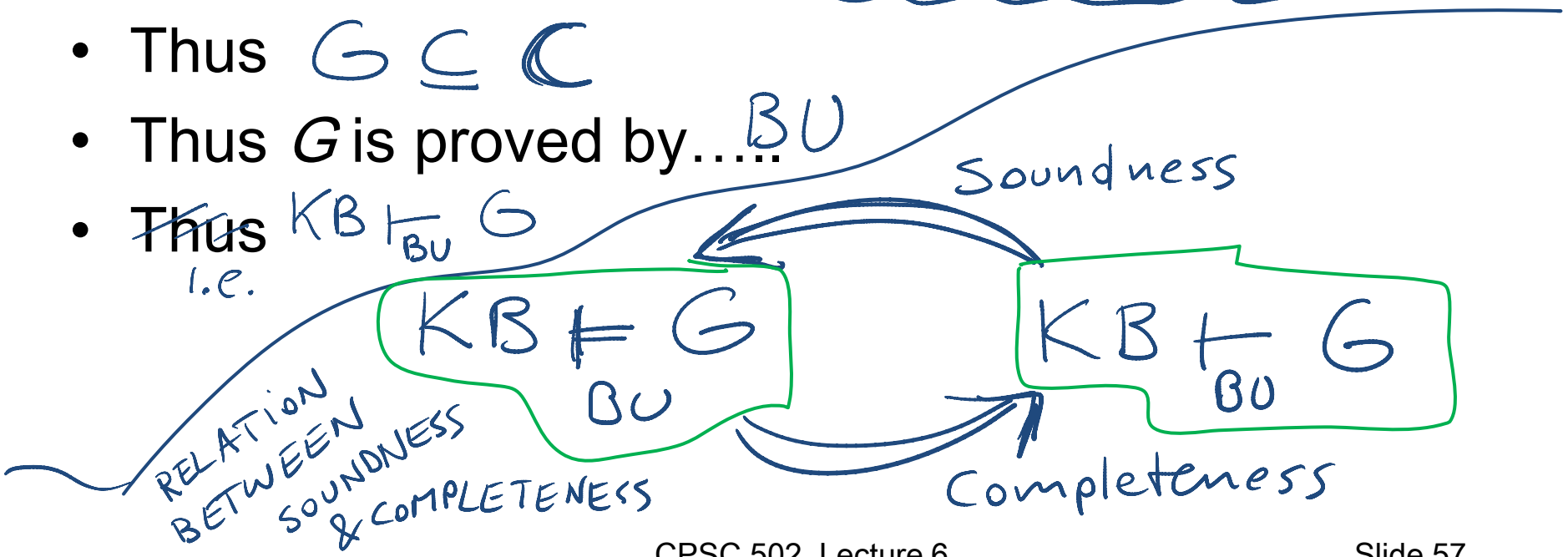
$KB \models G \Rightarrow KB \vdash_B G$

$KB \models G \Rightarrow KB \vdash_{BU} G \Rightarrow G \in C_{BU} \Rightarrow G \in C_B \Rightarrow KB \vdash_B G$

Completeness of Bottom Up
(proof summary)

If $KB \models G$ then $KB \vdash_{BU} G$

- Suppose $KB \models G$.
- Then G is true in all the models
- Thus G is true in the minimal model
- Thus $G \subseteq \mathbb{C}$
- Thus G is proved by ... BU
- ~~Thus~~ $KB \vdash_{BU} G$ i.e.

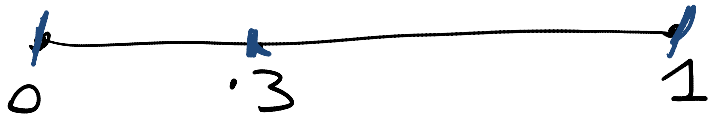


Sampling a discrete probability distribution

e.g. Sim. Annealing. Select n' with probability P

P = .3

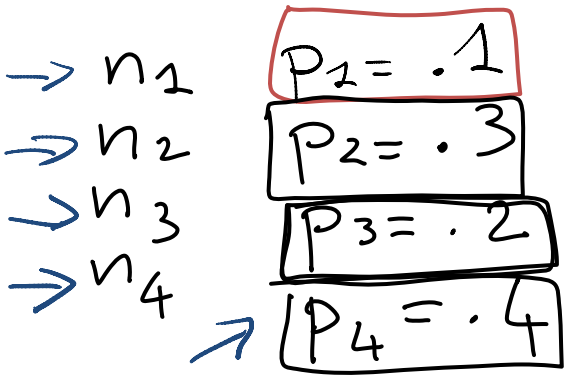
generate random number in [0,1]

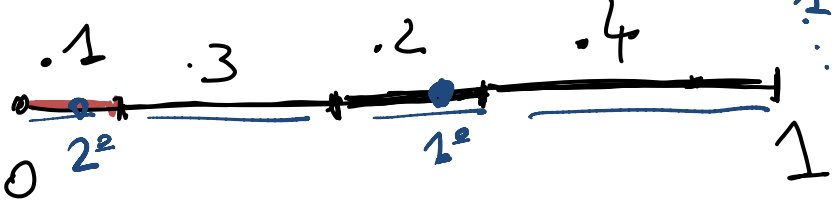


If < .3 accept n'

e.g. Beam Search: Select K individuals. Probability of selection proportional to their value

SAME HERE



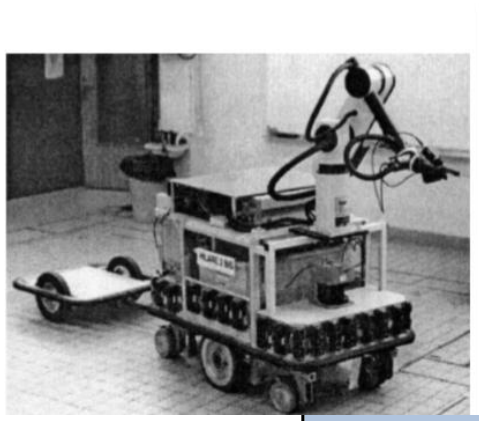


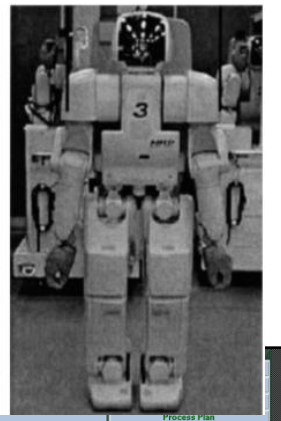
n3 first sample

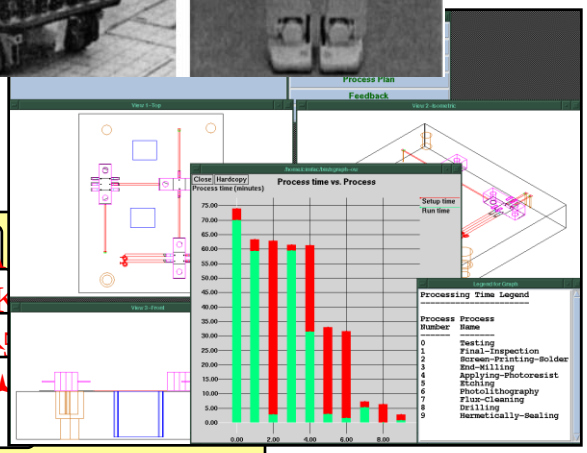
n1 second sample

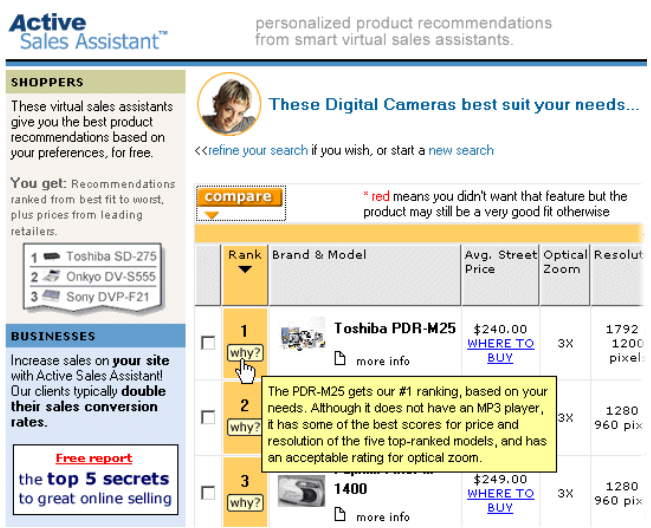
Now, do you know how to implement a planner for….

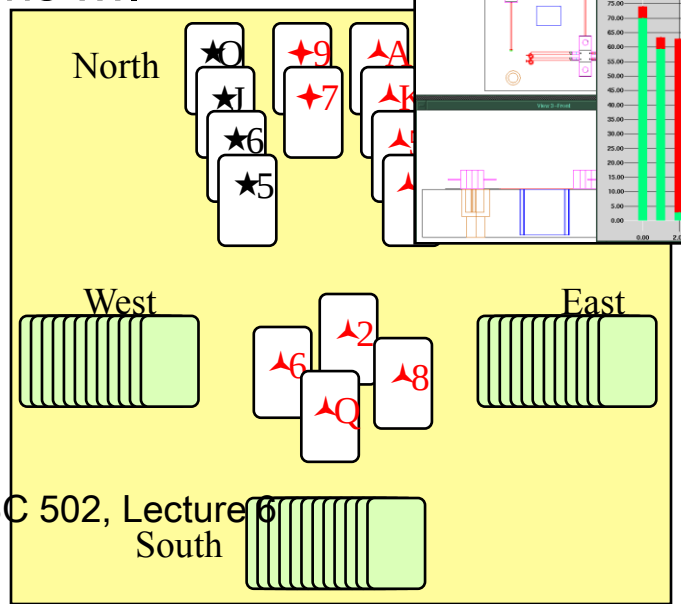
- Emergency Evacuation?
- Robotics?
- Space Exploration?
- Manufacturing Analysis?
- Games (e.g., Bridge)?
- Generating Natural language
  - Product Recommendations ….

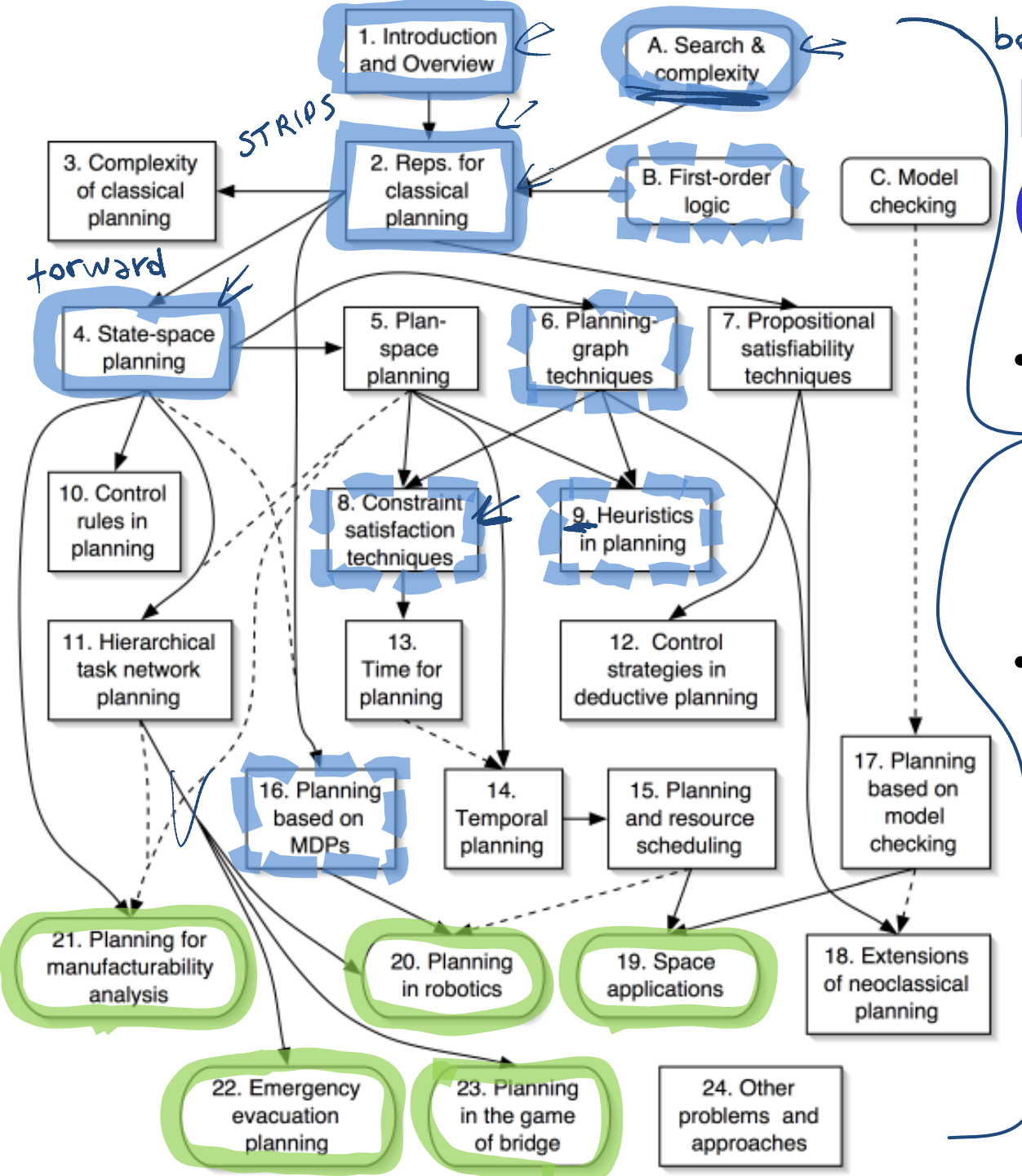












book chapters

No ☹, but you (will) know the key ideas ☺!

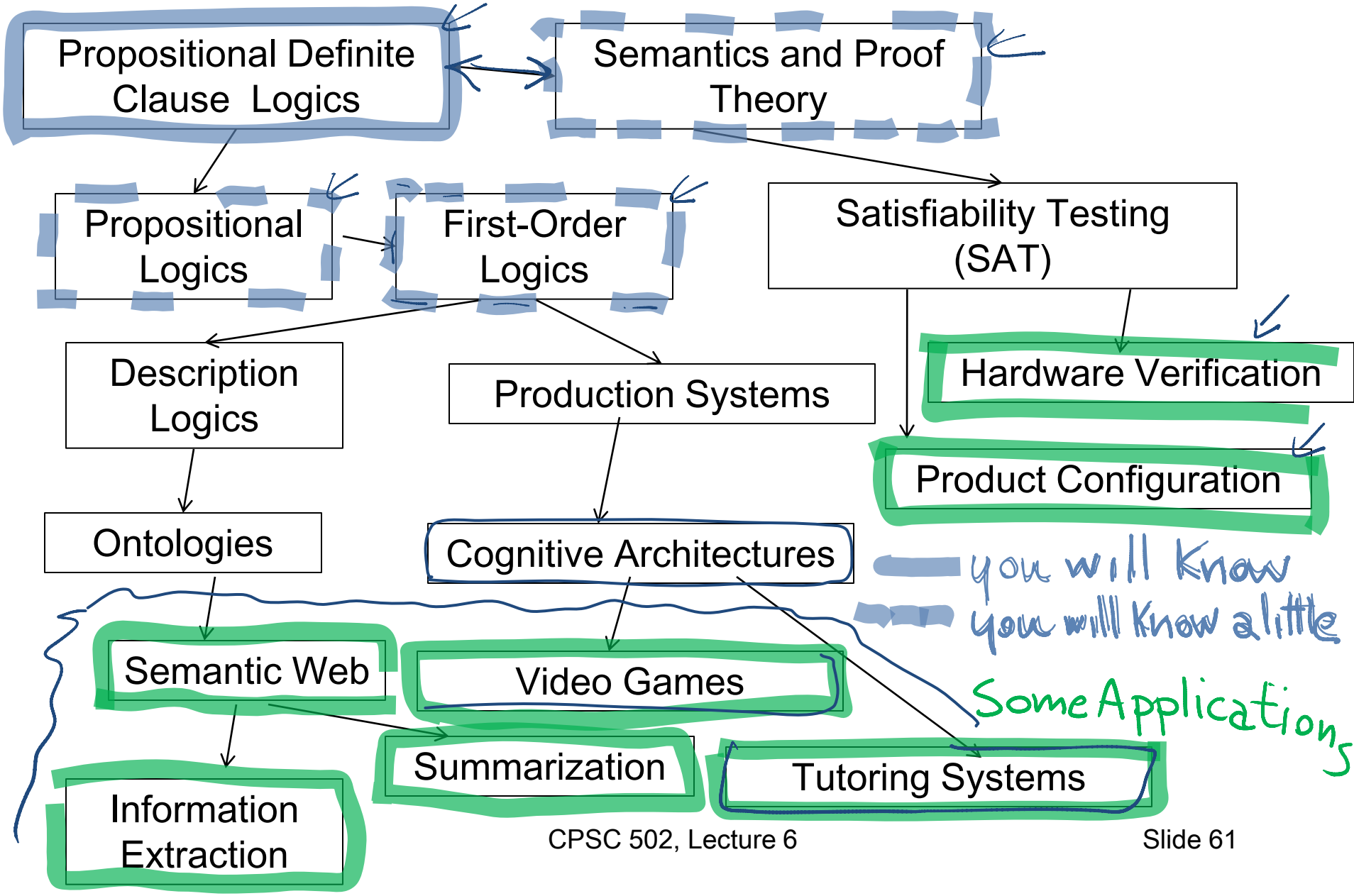
- Ghallab, Nau, and Traverso
  Automated Planning: Theory and Practice
  Morgan Kaufmann, May 2004
  ISBN 1-55860-856-7
- Web site:
  - ✓ http://www.laas.fr/planning

you know

you know a little

Applications

Logics in AI: Similar slide to the one for planning



Logics as a R&R system

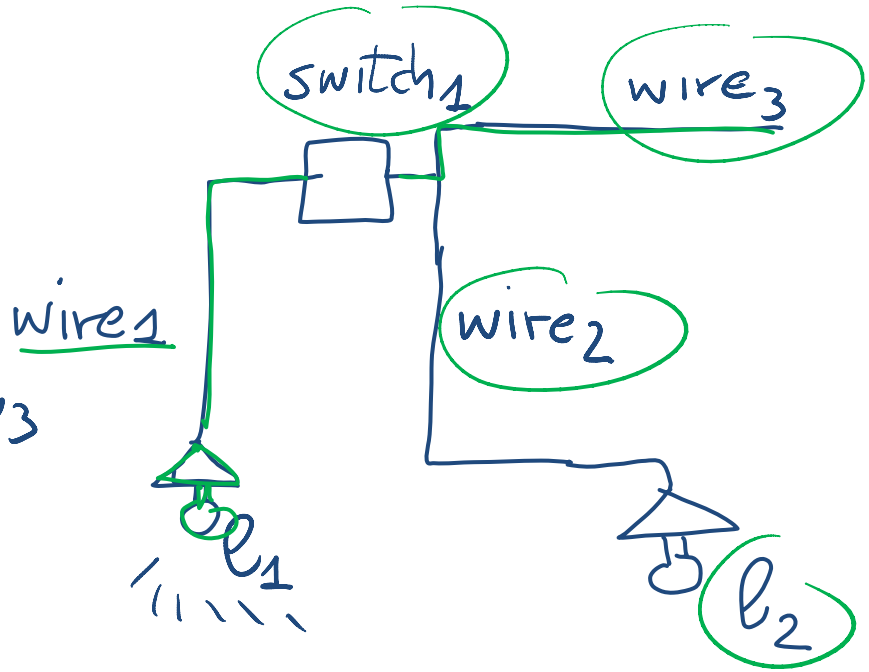
Represent

- formalize a domain

$on\_l_1 \leftarrow_{IF} live\_w_1$

$live\_w_1 \leftarrow_{IF} on\_sw_1 \wedge_{AND} live\_w_3$

.....



- reason about it

If the agent knows on_sw_1 and $live\_w_3$

it should be able to infer on_l_1

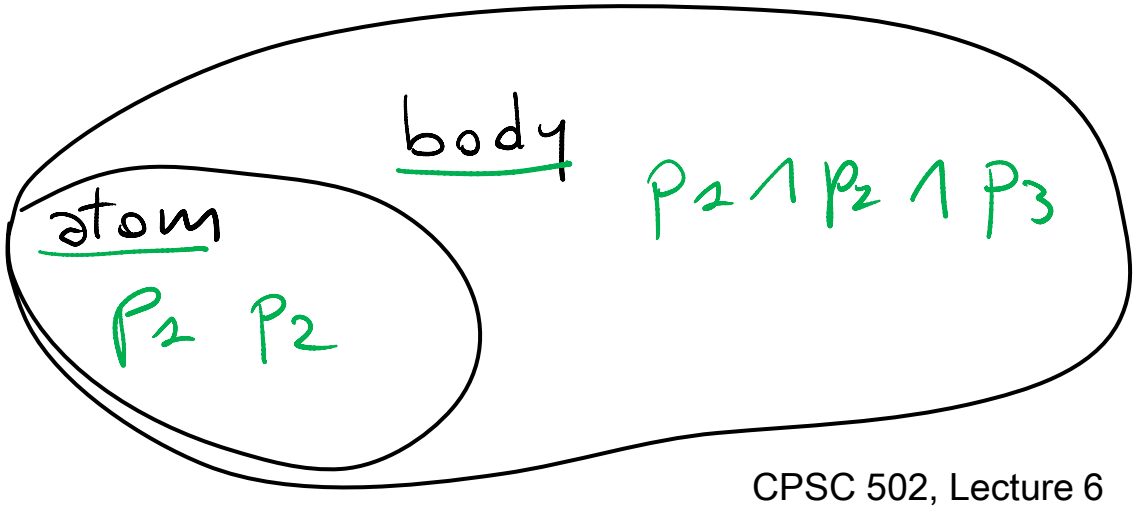
Propositional (Definite Clauses) Logic: Syntax

We start from a restricted form of Prop. Logic:

$\neg (p_1 \vee p_2) \Leftrightarrow (p_3 \vee \neg p_5)$

Only two kinds of statements

- that a proposition is true
- that a proposition is true if one or more other propositions are true



definite clause is either an atom or atom ← body

$p_3 \leftarrow p_1 \wedge p_2 \ldots$

Bottom-up proof procedure: Example

BU

$z \leftarrow f \wedge e$

$q \leftarrow f \wedge g \wedge z$

$e \leftarrow a \wedge b$

a

b

r

f

$C = \{t, r, b, a, c, z\}$

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.

BU can derive $r \wedge z$

$KB \vdash_{BU} r?$ ~~$q?$~~ $z?$

BU cannot derive

$KB \not\vdash_{BU} q$