CPSC 322
Introduction to Artificial Intelligence

September 20, 2004

Five Simple Steps to World Domination

(or how to build the black box)

1. Begin with a task domain that you want to characterize
2. Distinguish the things you want to talk about in the domain (the ontology)
3. Use symbols in the computer to represent objects and relations in the domain. (Symbols *denote* objects...they're not really the objects.)
4. Tell the computer the knowledge about the domain.
5. Ask the RRS a question which prompts the RRS to reason with its knowledge to solve a problem, produce answers, or generate actions

Five Simple Steps to Semantics

(or how to build the black box)

1. Begin with a task domain that you want to characterize. You must have an intended interpretation of that domain. Figure out the individuals that go into D.
2. Associate constants in the language with individuals in D. That mapping is ϕ.
3. For each relation you want to represent, associate a predicate symbol from the language. Each n-ary predicate is a mapping from D^n to TRUE or FALSE. That mapping is π.
4. Tell the computer the statements that are true in the intended interpretation. This is called axiomatizing the domain, where the definite clauses are axioms.
5. Ask questions about the intended interpretation, and interpret answers using meanings you supplied.

Why do we care about semantics?

Giving serious thought to your intended interpretation of your chosen domain helps you see whether the conclusions Generated by your RRS are right or wrong

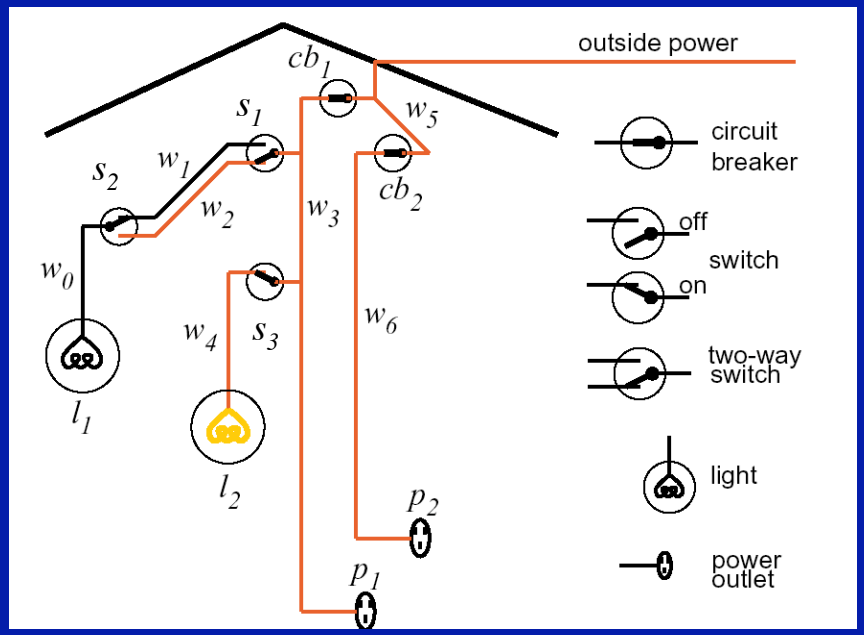
Your RRS doesn’t know...it’s just manipulating symbols

Formal semantics gives you a defined process for specifying the intended interpretation and maintaining the integrity of your RRS

So what exactly happens when we ask?

```
light(l1).
light(l2).
down(s1).
up(s2).
up(s3).
ok(l1).
ok(l2).
ok(cb1).
ok(cb2).
connected_to(l1, w0).
connected_to(w0, w1) <- up(s2).
connected_to(w0, w2) <- down(s2).
connected_to(w1, w3) <- up(s1).
connected_to(w2, w3) <- down(s1).
connected_to(l2, w4).
connected_to(w4, w3) <- up(s3).
connected_to(p1, w3).
connected_to(w3, w5) <- ok(cb1).
connected_to(p2, w6).
connected_to(w6, w5) <- ok(cb2).
connected_to(w5, outside).

ask connected_to(w3,w5) &
    connected_to(w2,w3).
```



Top-down ground proof query procedure

Start with a generalization of modus ponens, the basic rule of inference:

> if "h <- b_1 ^ b_2 ^ ... ^ b_m" is a clause in the knowledge base, and each b_i has been derived, then h can be derived

Derived means it can be computed from the knowledge base. We write

$KB \vdash g$

if g can be derived from KB.

Top-down ground proof query procedure

When we say to CILOG “ask ...”, we’re really saying “Here’s a theorem, go prove it.”

So “ask a_1 & a_2 & ... & a_m.” in CILOG or
“?a_1 ^ a_2 ^ ... ^ a_m.” in Datalog gets converted to an answer clause:

yes <- a_1 ^ a_2 ^ ... ^ a_m.

Top-down ground proof query procedure

Miraculously, the proof procedure selects an atom or conjunct from the right hand side of:

$$yes <- a_1 \wedge a_2 \wedge ... \wedge a_m.$$

Let's say the procedure selected a_i.
Then the proof procedure chooses a clause from the knowledge base whose head matches a_i. For example:

$$a_i <- b_1 \wedge ... \wedge b_x.$$

select vs. choose

select indicates “don’t-care nondeterminism”
if the selection made doesn’t lead to a solution (i.e., it’s not ultimately true) then there’s no reason to try any alternatives

so when the proof procedure selects an atom from the body of the answer clause, if that atom ultimately isn’t true then there’s no point in selecting some other atom in the body because it’s one big conjunction

select vs. choose

choose indicates “don’t-know nondeterminism”
if the selection made doesn’t lead to a solution (i.e., it’s not ultimately true) then it may be worthwhile to try other choices

so when the proof procedure selects a clause from the KB to resolve with the selected atom from the body of the answer clause, if that resolution doesn’t ultimately lead to a solution then the proof procedure should choose another clause from the KB...it just might be the one that leads to a solution

Top-down ground proof query procedure

The proof procedure then resolves the answer clause:

$$yes <- a_1 \wedge \ldots \wedge a_i \wedge \ldots \wedge a_m.$$

with the chosen clause from the knowledge base:

$$a_i <- b_1 \wedge \ldots \wedge b_x.$$

yielding:

$$yes <- a_1 \wedge \ldots \wedge b_1 \wedge \ldots \wedge b_x \wedge \ldots a_m.$$

Top-down ground proof query procedure

Keep doing this until all the atoms in the body of the answer clause are true:

yes <- a_1 ^ a_2 ^ a_3 ^ ... ^ a_m.
yes <- a_1 ^ true ^ a_3 ^ ... ^ a_m.
yes <- a_1 ^ a_3 ^ ... ^ a_m.
yes <- a_1 ^ true ^ ... ^ a_m.
yes <- a_1 ^ ... ^ a_m.
:
yes <- .

A sequence of answer clauses that ends with "yes <- ." is called a derivation. The process is called definite clause resolution.

Top-down definite clause interpreter (without variables)

solve or prove: $?a_1 \wedge \ldots \wedge a_k$.

AC := yes <- $a_1 \wedge \ldots \wedge a_k$.
repeat
    select a conjunct a_i from the body of AC
    choose clause C from KB with a_i as head
    replace a_i in the body of AC by the body of C
until AC is an answer (i.e., yes <- .)

So what exactly happens when we ask?

```
light(l1).
light(l2).
down(s1).
up(s2).
up(s3).
ok(l1).
ok(l2).
ok(cb1).
ok(cb2).
connected_to(l1, w0).
connected_to(w0, w1) <- up(s2).
connected_to(w0, w2) <- down(s2).
connected_to(w1, w3) <- up(s1).
connected_to(w2, w3) <- down(s1).
connected_to(l2, w4).
connected_to(w4, w3) <- up(s3).
connected_to(p1, w3).
connected_to(w3, w5) <- ok(cb1).
connected_to(p2, w6).
connected_to(w6, w5) <- ok(cb2).
connected_to(w5, outside).

ask connected_to(w3,w5) &
    connected_to(w2,w3).
```