Contents Chapters 2 & 3

Chapters 2 & 3: A Representation and Reasoning System

- Lecture 1 Representation and Reasoning Systems. Datalog.
- Lecture 2 Semantics.
- Lecture 3 Variables, queries and answers, limitations.
- Lecture 4 Proofs. Soundness and completeness.
- Lecture 5 SLD resolution.
- Lecture 6 Proofs with variables. Function Symbols.

Representation and Reasoning System

A Representation and Reasoning System (RRS) is made up of:

- **formal language:** specifies the legal sentences
- **semantics:** specifies the meaning of the symbols
- **reasoning theory or proof procedure:** nondeterministic specification of how an answer can be produced.

Implementation of an RRS

An implementation of an RRS consists of

- **language parser:** maps sentences of the language into data structures.
- **reasoning procedure:** implementation of reasoning theory + search strategy.

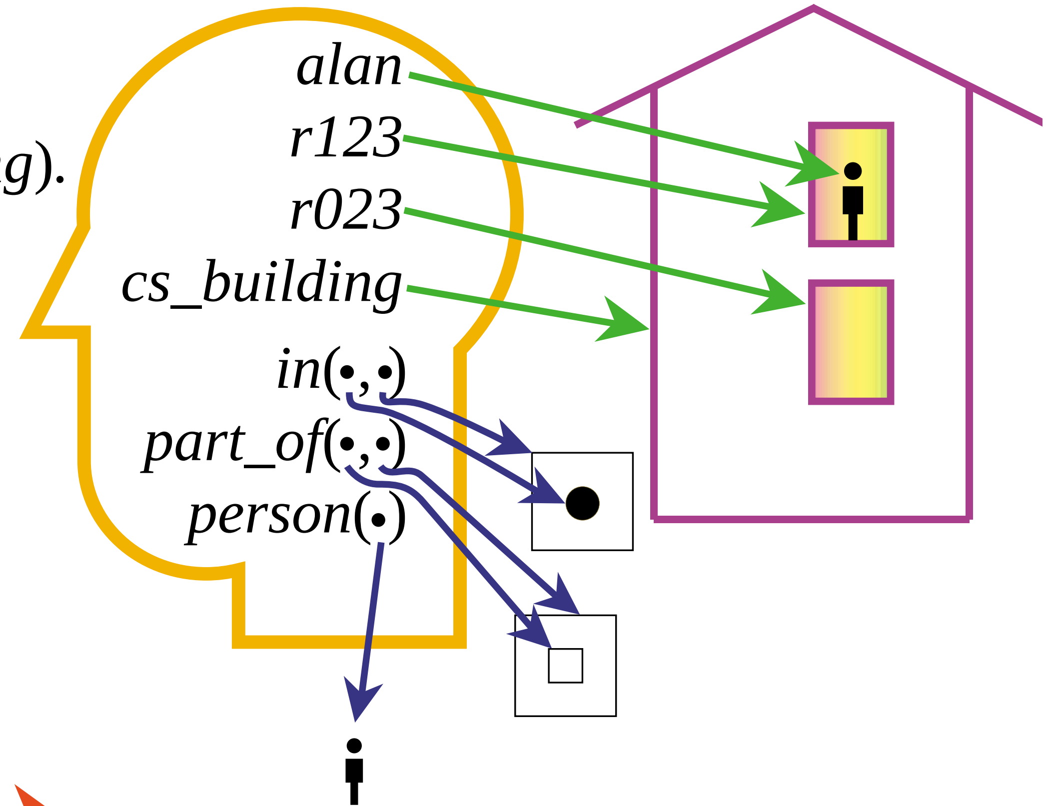
Note: the semantics aren't reflected in the implementation!

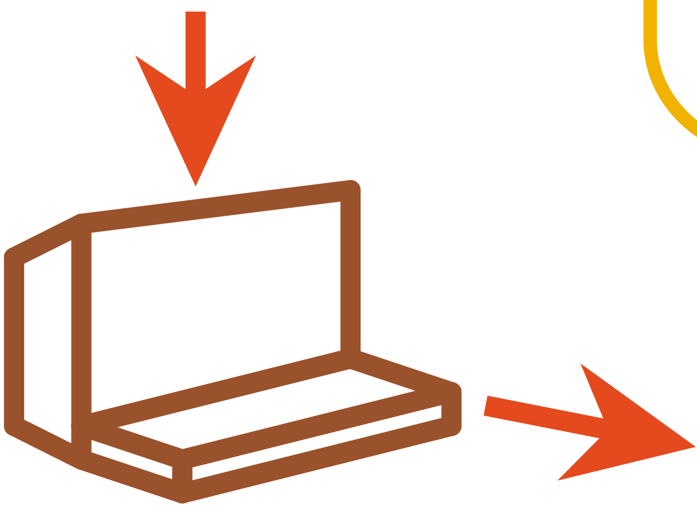
Using an RRS

1. Begin with a task domain.
2. Distinguish those things you want to talk about (the ontology).
3. Choose symbols in the computer to denote objects and relations.
4. Tell the system knowledge about the domain.
5. Ask the system questions.

Role of Semantics in an RRS

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





in(*alan*,*cs_building*)

Simplifying Assumptions of Initial RRS

An agent's knowledge can be usefully described in terms of *individuals* and *relations* among individuals.

An agent's knowledge base consists of *definite* and *positive* statements.

The environment is *static*.

There are only a finite number of individuals of interest in the domain. Each individual can be given a unique name.

$\Longrightarrow$ Datalog

Syntax of Datalog

variable starts with upper-case letter.

constant starts with lower-case letter or is a sequence of digits (numeral).

predicate symbol starts with lower-case letter.

term is either a variable or a constant.

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

Syntax of Datalog (cont)

definite clause is either an atomic symbol (a fact) or of the form:

$$\underbrace{a}_{\text{head}} \quad \leftarrow \quad \underbrace{b_1 \wedge \cdots \wedge b_m}_{\text{body}}$$

where a and b_i are atomic symbols.

query is of the form $?b_1 \wedge \cdots \wedge b_m$.

knowledge base is a set of definite clauses.

Example Knowledge Base

$$in(alan, R) \leftarrow$$

$$teaches(alan, cs322) \wedge$$

$$in(cs322, R).$$

$$grandfather(william, X) \leftarrow$$

$$father(william, Y) \wedge$$

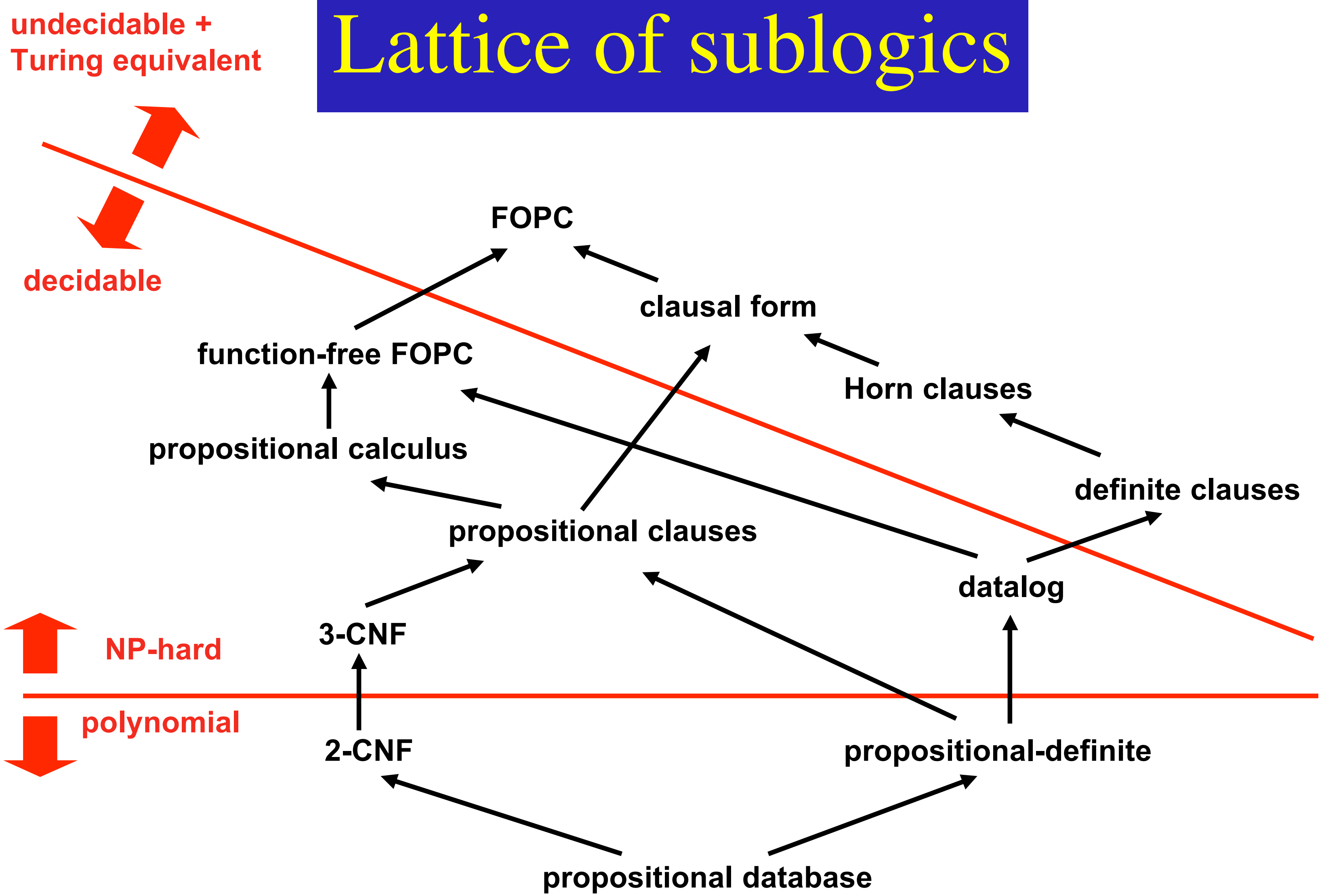
$$parent(Y, X).$$

$$slithy(toves) \leftarrow$$

$$mimsy \wedge borogroves \wedge$$

$$outgrabe(mome, Raths).$$

Lattice of sublogics



Characteristics of sublogics

FOPC	first-order predicate calculus
clausal form	conjunctions of disjunctions of literals (CNF)
Horn clauses	definite clauses and integrity contraints
definite clauses	cluases with function symbols
function-free FOPC	FOPC without functions or existentially quantified variables in the scope of universally quantified variables
datalog	definite clauses without function symbols
propositional calculus	FOPC without variables or function symbols
propositional clauses	clausal form without variables or function symbols
3-CNF	propositional clauses with at most 3 disjuncts in a clause
2-CNF	propositional clauses with at most 2 disjuncts in a clause
propositional definite	definite clauses without variables or function symbols
propositional database	facts without variables or function symbols (no rules)

Semantics: General Idea

A semantics specifies the meaning of sentences in the language.

An interpretation specifies:

- what objects (individuals) are in the world
- the correspondence between symbols in the computer and objects & relations in world
  - constants denote individuals
  - predicate symbols denote relations

Formal Semantics

An interpretation is a triple $I = \langle D, \phi, \pi \rangle$, where

- D, the domain, is a nonempty set. Elements of D are individuals.
- ϕ is a mapping that assigns to each constant an element of D. Constant c denotes individual $\phi(c)$.
- π is a mapping that assigns to each n-ary predicate symbol a relation: a function from D^n into $\{\textit{TRUE}, \textit{FALSE}\}$.

Example Interpretation

Constants: *phone*, *pencil*, *telephone*.

Predicate Symbol: *noisy* (unary), *left_of* (binary).

- $D = \{$✂, ☎, ✎$\}$.
- $\phi(phone) =$ ☎, $\phi(pencil) =$ ✎, $\phi(telephone) =$ ☎.
- $\pi(noisy)$:

⟨✂⟩	*FALSE*	⟨☎⟩	*TRUE*	⟨✎⟩	*FALSE*

$\pi(left\_of)$:

⟨✂, ✂⟩	*FALSE*	⟨✂, ☎⟩	*TRUE*	⟨✂, ✎⟩	*TRUE*
⟨☎, ✂⟩	*FALSE*	⟨☎, ☎⟩	*FALSE*	⟨☎, ✎⟩	*TRUE*
⟨✎, ✂⟩	*FALSE*	⟨✎, ☎⟩	*FALSE*	⟨✎, ✎⟩	*FALSE*

Important points to note

- The domain D can contain real objects. (e.g., a person, a room, a course). D can't necessarily be stored in a computer.

- $\pi(p)$ specifies whether the relation denoted by the n-ary predicate symbol p is true or false for each n-tuple of individuals.

- If predicate symbol p has no arguments, then $\pi(p)$ is either *TRUE* or *FALSE*.

Truth in an interpretation

A constant c **denotes in I** the individual $\phi(c)$.

Ground (variable-free) atom $p(t_1, \ldots, t_n)$ is

- **true in interpretation I** if $\pi(p)(t'_1, \ldots, t'_n) = \textit{TRUE}$, where t_i denotes t'_i in interpretation I and
- **false in interpretation I** if $\pi(p)(t'_1, \ldots, t'_n) = \textit{FALSE}$.

Ground clause $h \leftarrow b_1 \wedge \ldots \wedge b_m$ is **false in interpretation I** if h is false in I and each b_i is true in I, and is **true in interpretation I** otherwise.

Example Truths

In the interpretation given before:

$noisy(phone)$	true
$noisy(telephone)$	true
$noisy(pencil)$	false
$left\_of(phone, pencil)$	true
$left\_of(phone, telephone)$	false
$noisy(pencil) \leftarrow left\_of(phone, telephone)$	true
$noisy(pencil) \leftarrow left\_of(phone, pencil)$	false
$noisy(phone) \leftarrow noisy(telephone) \wedge noisy(pencil)$	true

Models and logical consequences

- A knowledge base, *KB*, is true in interpretation I if and only if every clause in *KB* is true in I.
- A **model** of a set of clauses is an interpretation in which all the clauses are true.
- 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*.
- That is, $KB \models g$ if there is no interpretation in which *KB* is true and g is false.

Simple Example

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

	$\pi(p)$	$\pi(q)$	$\pi(r)$	$\pi(s)$	
I_1	*TRUE*	*TRUE*	*TRUE*	*TRUE*	is a model of KB
I_2	*FALSE*	*FALSE*	*FALSE*	*FALSE*	not a model of KB
I_3	*TRUE*	*TRUE*	*FALSE*	*FALSE*	is a model of KB
I_4	*TRUE*	*TRUE*	*TRUE*	*FALSE*	is a model of KB
I_5	*TRUE*	*TRUE*	*FALSE*	*TRUE*	not a model of KB

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

User's view of Semantics

1. Choose a task domain: intended interpretation.
2. Associate constants with individuals you want to name.
3. For each relation you want to represent, associate a predicate symbol in the language.
4. Tell the system clauses that are true in the intended interpretation: axiomatizing the domain.
5. Ask questions about the intended interpretation.
6. If $KB \models g$, then g must be true in the intended interpretation.

Computer's view of semantics

- The computer doesn't have access to the intended interpretation.
- All it knows is the knowledge base.
- The computer can determine if a formula is a logical consequence of KB.
- If $KB \models g$ then g must be true in the intended interpretation.
- If $KB \not\models g$ then there is a model of KB in which g is false. This could be the intended interpretation.

Variables

- Variables are universally quantified in the scope of a clause.
- A variable assignment is a function from variables into the domain.
- Given an interpretation and a variable assignment, each term denotes an individual and each clause is either true or false.
- A clause containing variables is true in an interpretation if it is true for all variable assignments.

Queries and Answers

A query is a way to ask if a body is a logical consequence of the knowledge base:

$$?b_1 \wedge \cdots \wedge b_m .$$

An answer is either

- an instance of the query that is a logical consequence of the knowledge base KB, or
- no if no instance is a logical consequence of KB.

Example Queries

$$KB = \begin{cases} in(alan, r123). \\ part\_of(r123, cs\_building). \\ in(X, Y) \leftarrow part\_of(Z, Y) \wedge in(X, Z). \end{cases}$$

Query	Answer
$?part\_of(r123, B).$	$part\_of(r123, cs\_building)$
$?part\_of(r023, cs\_building).$	no
$?in(alan, r023).$	no
$?in(alan, B).$	$in(alan, r123)$
	$in(alan, cs\_building)$

Logical Consequence

Atom g is a logical consequence of KB if and only if:

- g is a fact in KB, or
- there is a rule

  $$g \leftarrow b_1 \wedge \ldots \wedge b_k$$

  in KB such that each b_i is a logical consequence of KB.

Debugging false conclusions

To debug answer g that is false in the intended interpretation:

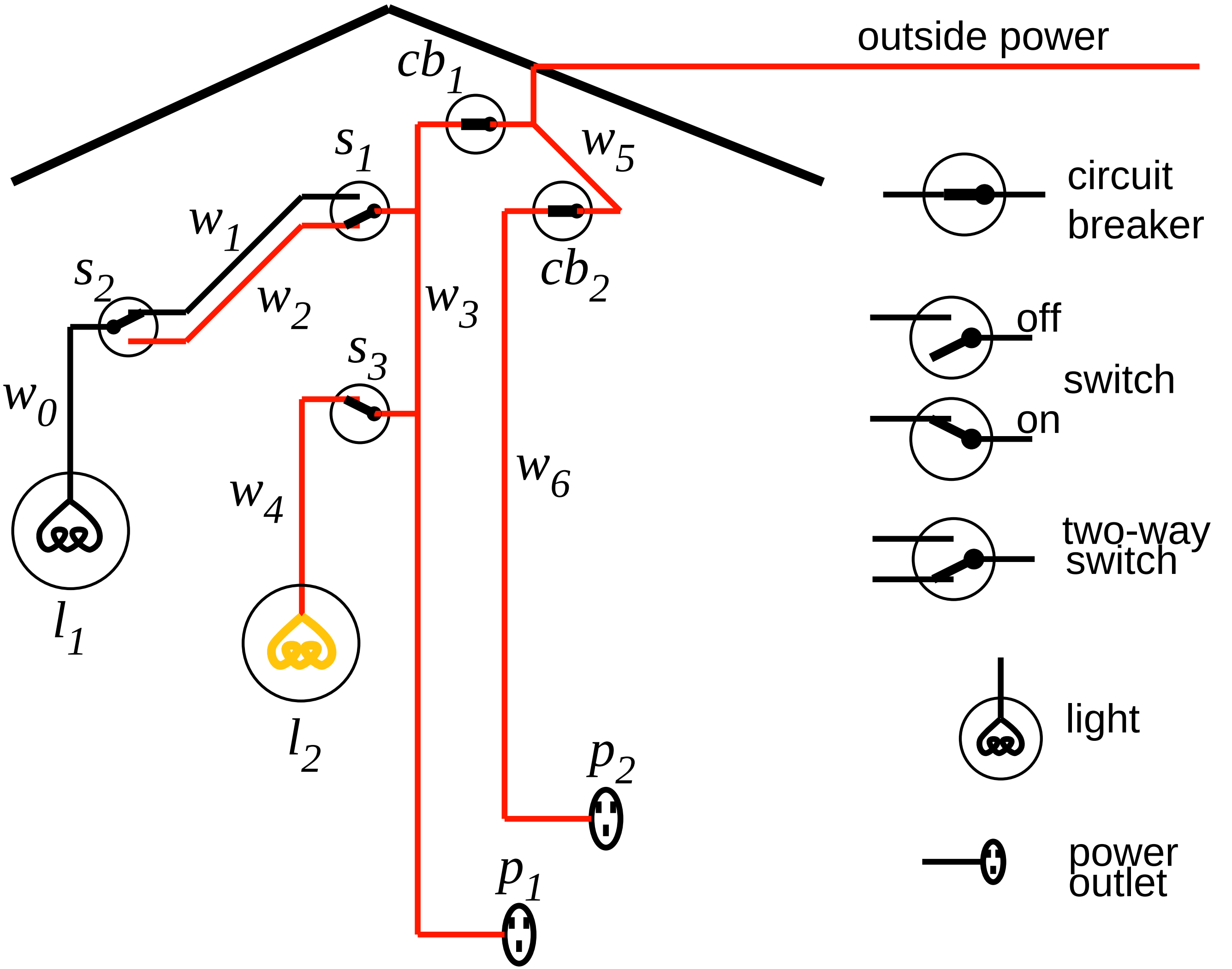
- If g is a fact in KB, this fact is wrong.
- Otherwise, suppose g was proved using the rule:

$$g \leftarrow b_1 \wedge \ldots \wedge b_k$$

  where each b_i is a logical consequence of KB.

  - If each b_i is true in the intended interpretation, this clause is false in the intended interpretation.
  - If some b_i is false in the intended interpretation, debug b_i.

Electrical Environment



Axiomatizing the Electrical Environment

% $light(L)$ is true if L is a light

$light(l_1)$. $light(l_2)$.

% $down(S)$ is true if switch S is down

$down(s_1)$. $up(s_2)$. $up(s_3)$.

% $ok(D)$ is true if D is not broken

$ok(l_1)$. $ok(l_2)$. $ok(cb_1)$. $ok(cb_2)$.

$?light(l_1)$. $\Longrightarrow$ yes

$?light(l_6)$. $\Longrightarrow$ no

$?up(X)$. $\Longrightarrow$ $up(s_2)$, $up(s_3)$

$connected\_to(X, Y)$ is true if component X is connected to Y

$$connected\_to(w_0, w_1) \leftarrow up(s_2).$$

$$connected\_to(w_0, w_2) \leftarrow down(s_2).$$

$$connected\_to(w_1, w_3) \leftarrow up(s_1).$$

$$connected\_to(w_2, w_3) \leftarrow down(s_1).$$

$$connected\_to(w_4, w_3) \leftarrow up(s_3).$$

$$connected\_to(p_1, w_3).$$

$?connected\_to(w_0, W).$ $\Longrightarrow$ $W = w_1$

$?connected\_to(w_1, W).$ $\Longrightarrow$ no

$?connected\_to(Y, w_3).$ $\Longrightarrow$ $Y = w_2, Y = w_4, Y = p_1$

$?connected\_to(X, W).$ $\Longrightarrow$ $X = w_0, W = w_1, \ldots$

% $lit(L)$ is true if the light L is lit

$$lit(L) \leftarrow light(L) \wedge ok(L) \wedge live(L).$$

% $live(C)$ is true if there is power coming into C

$$live(Y) \leftarrow$$
$$connected\_to(Y, Z) \wedge$$
$$live(Z).$$

$$live(outside).$$

This is a recursive definition of *live*.

Recursion and Mathematical Induction

$$above(X, Y) \leftarrow on(X, Y).$$

$$above(X, Y) \leftarrow on(X, Z) \wedge above(Z, Y).$$

This can be seen as:

- Recursive definition of *above*: prove *above* in terms of a base case (*on*) or a simpler instance of itself; or
- Way to prove *above* by mathematical induction: the base case is when there are no blocks between X and Y, and if you can prove *above* when there are n blocks between them, you can prove it when there are $n + 1$ blocks.

Limitations

Suppose you had a database using the relation:

$$enrolled(S, C)$$

which is true when student S is enrolled in course C.

You can't define the relation:

$$empty\_course(C)$$

which is true when course C has no students enrolled in it.

This is because $empty\_course(C)$ doesn't logically follow from a set of *enrolled* relations. There are always models where someone is enrolled in a course!

Proofs

- A **proof** is a mechanically derivable demonstration that a formula logically follows from a knowledge base.
- Given a proof procedure, $KB \vdash g$ means g can be derived from knowledge base KB.
- Recall $KB \models g$ means g is true in all models of KB.
- A proof procedure is **sound** if $KB \vdash g$ implies $KB \models g$.
- 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 $g \in C$ 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.

Example

$a \leftarrow b \wedge c.$

$a \leftarrow e \wedge f.$

$b \leftarrow f \wedge k.$

$c \leftarrow e.$

$d \leftarrow k.$

$e.$

$f \leftarrow j \wedge e.$

$f \leftarrow c.$

$j \leftarrow c.$

Soundness of bottom-up proof procedure

If $KB \vdash g$ then $KB \models g$.

Suppose there is a g such that $KB \vdash g$ and $KB \not\models g$.

Let h be the first atom added to C that's not true in every model of KB. Suppose h isn't true in model I of KB. There must be a clause in KB of form

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

Each b_i is true in I. h is false in I. So this clause is false in I. Therefore I isn't a model of KB.

Contradiction: thus no such g exists.

Fixed Point

The C generated at the end of the bottom-up algorithm is called a **fixed point.**

Let I be the interpretation in which every element of the fixed point is true and every other atom is false.

I is a model of KB.

Proof: suppose $h \leftarrow b_1 \wedge \ldots \wedge b_m$ in KB is false in I. Then h is false and each b_i is true in I. Thus h can be added to C. Contradiction to C being the fixed point.

I is called a **Minimal Model.**

Completeness

If $KB \models g$ then $KB \vdash g$.

Suppose $KB \models g$. Then g is true in all models of KB.

Thus g is true in the minimal model.

Thus g is generated by the bottom up algorithm.

Thus $KB \vdash g$.

Top-down Ground Proof Procedure

Idea: search backward from a query to determine if it is a logical consequence of KB.

An answer clause is of the form:

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

The SLD Resolution of this answer clause on atom a_i with the clause:

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

is the answer clause

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

Derivations

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

Top-down definite clause interpreter

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

> ac := "$yes \leftarrow q_1 \wedge \ldots \wedge q_k$"
>
> **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.

Nondeterministic Choice

- **Don't-care nondeterminism** If one selection doesn't lead to a solution, there is no point trying other alternatives. **select**
- **Don't-know nondeterminism** If one choice doesn't lead to a solution, other choices may. **choose**

Example: successful derivation

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

$c \leftarrow e.$ $d \leftarrow k.$ $e.$

$f \leftarrow j \wedge e.$ $f \leftarrow c.$ $j \leftarrow c.$

Query: $?a$

$\gamma_0 : \quad yes \leftarrow a$

$\gamma_1 : \quad yes \leftarrow e \wedge f$

$\gamma_2 : \quad yes \leftarrow f$

$\gamma_3 : \quad yes \leftarrow c$

$\gamma_4 : \quad yes \leftarrow e$

$\gamma_5 : \quad yes \leftarrow$

Example: failing derivation

$a \leftarrow b \wedge c.$ $\quad a \leftarrow e \wedge f.$ $\quad b \leftarrow f \wedge k.$

$c \leftarrow e.$ $\quad d \leftarrow k.$ $\quad e.$

$f \leftarrow j \wedge e.$ $\quad f \leftarrow c.$ $\quad j \leftarrow c.$

Query: $?a$

$\gamma_0 : \quad yes \leftarrow a$

$\gamma_1 : \quad yes \leftarrow b \wedge c$

$\gamma_2 : \quad yes \leftarrow f \wedge k \wedge c$

$\gamma_3 : \quad yes \leftarrow c \wedge k \wedge c$

$\gamma_4 : \quad yes \leftarrow e \wedge k \wedge c$

$\gamma_5 : \quad yes \leftarrow k \wedge c$

Reasoning with Variables

- An **instance** of an atom or a clause is obtained by uniformly substituting terms for variables.
- A **substitution** is a finite set of the form $\{V_1/t_1, \ldots, V_n/t_n\}$, where each V_i is a distinct variable and each t_i is a term.
- The **application** of a substitution $\sigma = \{V_1/t_1, \ldots, V_n/t_n\}$ to an atom or clause e, written $e\sigma$, is the instance of e with every occurrence of V_i replaced by t_i.

Application Examples

The following are substitutions:

- $\sigma_1 = \{X/A, Y/b, Z/C, D/e\}$
- $\sigma_2 = \{A/X, Y/b, C/Z, D/e\}$
- $\sigma_3 = \{A/V, X/V, Y/b, C/W, Z/W, D/e\}$

The following shows some applications:

- $p(A, b, C, D)\sigma_1 = p(A, b, C, e)$
- $p(X, Y, Z, e)\sigma_1 = p(A, b, C, e)$
- $p(A, b, C, D)\sigma_2 = p(X, b, Z, e)$
- $p(X, Y, Z, e)\sigma_2 = p(X, b, Z, e)$
- $p(A, b, C, D)\sigma_3 = p(V, b, W, e)$
- $p(X, Y, Z, e)\sigma_3 = p(V, b, W, e)$

Unifiers

- Substitution σ is a unifier of e_1 and e_2 if $e_1\sigma = e_2\sigma$.
- Substitution σ is a most general unifier (mgu) of e_1 and e_2 if
  - σ is a unifier of e_1 and e_2; and
  - if substitution σ' also unifies e_1 and e_2, then $e\sigma'$ is an instance of $e\sigma$ for all atoms e.
- If two atoms have a unifier, they have a most general unifier.

Unification Example

$p(A, b, C, D)$ and $p(X, Y, Z, e)$ have as unifiers:

- $\sigma_1 = \{X/A, Y/b, Z/C, D/e\}$
- $\sigma_2 = \{A/X, Y/b, C/Z, D/e\}$
- $\sigma_3 = \{A/V, X/V, Y/b, C/W, Z/W, D/e\}$
- $\sigma_4 = \{A/a, X/a, Y/b, C/c, Z/c, D/e\}$
- $\sigma_5 = \{X/A, Y/b, Z/A, C/A, D/e\}$
- $\sigma_6 = \{X/A, Y/b, Z/C, D/e, W/a\}$

The first three are most general unifiers.

The following substitutions are not unifiers:

- $\sigma_7 = \{Y/b, D/e\}$
- $\sigma_8 = \{X/a, Y/b, Z/c, D/e\}$

Bottom-up procedure

- You can carry out the bottom-up procedure on the ground instances of the clauses.

- Soundness is a direct corollary of the ground soundness.

- For completeness, we build a canonical minimal model. We need a denotation for constants:

  Herbrand interpretation: The domain is the set of constants (we invent one if the KB or query doesn't contain one). Each constant denotes itself.

Definite Resolution with Variables

A **generalized answer clause** is of the form

$$yes(t_1, \ldots, t_k) \leftarrow a_1 \wedge a_2 \wedge \ldots \wedge a_m,$$

where $t_1, \ldots, t_k$ are terms and $a_1, \ldots, a_m$ are atoms.

The **SLD resolution** of this generalized answer clause on a_i with the clause

$$a \leftarrow b_1 \wedge \ldots \wedge b_p,$$

where a_i and a have most general unifier θ, is

$$(yes(t_1, \ldots, t_k) \leftarrow$$
$$a_1 \wedge \ldots \wedge a_{i-1} \wedge b_1 \wedge \ldots \wedge b_p \wedge a_{i+1} \wedge \ldots \wedge a_m)\theta.$$

To solve query $?B$ with variables $V_1, \ldots, V_k$:

Set ac to generalized answer clause $yes(V_1, \ldots, V_k) \leftarrow B$;

While ac is not an answer do

Suppose ac is $yes(t_1, \ldots, t_k) \leftarrow a_1 \wedge a_2 \wedge \ldots \wedge a_m$

Select atom a_i in the body of ac;

Choose clause $a \leftarrow b_1 \wedge \ldots \wedge b_p$ in KB;

Rename all variables in $a \leftarrow b_1 \wedge \ldots \wedge b_p$;

Let θ be the most general unifier of a_i and a.

Fail if they don't unify;

Set ac to $(yes(t_1, \ldots, t_k) \leftarrow a_1 \wedge \ldots \wedge a_{i-1} \wedge$

$b_1 \wedge \ldots \wedge b_p \wedge a_{i+1} \wedge \ldots \wedge a_m)\theta$

end while.

Example

$live(Y) \leftarrow connected\_to(Y, Z) \wedge live(Z).\ \ live(outside).$

$connected\_to(w_6, w_5).\quad connected\_to(w_5, outside).$

$?live(A).$

$$yes(A) \leftarrow live(A).$$

$$yes(A) \leftarrow connected\_to(A, Z_1) \wedge live(Z_1).$$

$$yes(w_6) \leftarrow live(w_5).$$

$$yes(w_6) \leftarrow connected\_to(w_5, Z_2) \wedge live(Z_2).$$

$$yes(w_6) \leftarrow live(outside).$$

$$yes(w_6) \leftarrow .$$

Function Symbols

Often we want to refer to individuals in terms of components.

Examples: 4:55 p.m. English sentences. A classlist.

We extend the notion of term. So that a term can be $f(t_1, \ldots, t_n)$ where f is a function symbol and the t_i are terms.

In an interpretation and with a variable assignment, term $f(t_1, \ldots, t_n)$ denotes an individual in the domain.

With one function symbol and one constant we can refer to infinitely many individuals.

Lists

A list is an ordered sequence of elements.

Let's use the constant nil to denote the empty list, and the function $cons(H, T)$ to denote the list with first element H and rest-of-list T. These are not built-in.

The list containing $david$, $alan$ and $randy$ is

$$cons(david, cons(alan, cons(randy, nil)))$$

$append(X, Y, Z)$ is true if list Z contains the elements of X followed by the elements of Y

$$append(nil, Z, Z).$$

$$append(cons(A, X), Y, cons(A, Z)) \leftarrow append(X, Y, Z).$$