Knowledge and Inference

- Recall basic concepts of logic
- Logical inference
  - deduction
  - *abduction*
  - *induction*
- Clausal Logic
- Deductive Inference (e.g. resolution)
- Recap of SLD and SLDNF

Logic: another thing that penguins aren't very good at.

Logic (a recap)

- Humans capable of manipulating logical information and making logical inference

The red block is on the green block.
The green block is somewhere above the blue block.
The green block is not on the blue block.
The yellow block is on the green block or the blue block.
There is some block on the black block.

Facts

There can be only one block on another.
A block cannot be two colors at once.

Background knowledge

- Logic is a mechanism for expressing a particular world as a knowledge base, and computing logical consequences of a knowledge base.

$on(red, green) \wedge \neg on(green, blue)$
$\exists X\ [\ block(X) \wedge on(green, X) \wedge on(X, blue)\ ]$
$on(yellow, green) \vee on(yellow, blue)$
$\exists X\ [\ block(X) \wedge on(X, black)\ ]$

$block(red) \wedge block(yellow) \wedge block(blue) \wedge block(back) \wedge block(green)$
$\forall X,Y,Z\ [\ on(X, Y) \wedge on(Z, Y) \rightarrow X = Z)\ ]$

Logic (a recap)

- **Propositional Logic**
  - » propositional constants $p, q, r, s, \ldots..$
  - » connectives $\neg\,, \wedge\,, \vee\,, \rightarrow$

Syntax

  - » sentences $((p \wedge q) \vee r) \rightarrow (p \wedge r))$

  - » propositional interpretation assigns each propositional constant a unique true value $p^i = T,\ q^i = F,\ r^i = T$

  - » interpretation of sentences is constructed from propositional interpretation and truth tables $((p \wedge q) \vee r) \rightarrow (p \wedge r))^i = T$
  - » logical entailment of a sentence from a set of sentences, given as premises, is when the sentence is true in all interpretations that satisfy the premises $\{p, p \rightarrow q\} \models q$

Logic (a recap)

❑ Predicate Logic

- » propositional letters — raining, snowing, wet…..
- » constants — table, block1, block2, etc.
- » variables — X, X_1, Y, Y_1, etc.
- » functions — size, color, etc.
- » predicates — on, above, clear, block etc.

Terms

» sentences

$\neg$block(table)

$\forall X$ block(X) $\leftrightarrow$ X= block1 $\vee$ X=block2 $\vee$ X= block3

$\forall X,Y$ (block(X) $\wedge$ block(Y) $\wedge$ size(X) = size(Y) $\rightarrow$ sameSize(X,Y))

$\forall X$ (clear(X) $\leftrightarrow$ (block(X) $\wedge$ $\neg\exists Y$ on(Y, X)))

$\forall X,Y$ (on(X,Y) $\leftrightarrow$ (block(X) $\wedge$ block(Y)) $\vee$ Y= table)

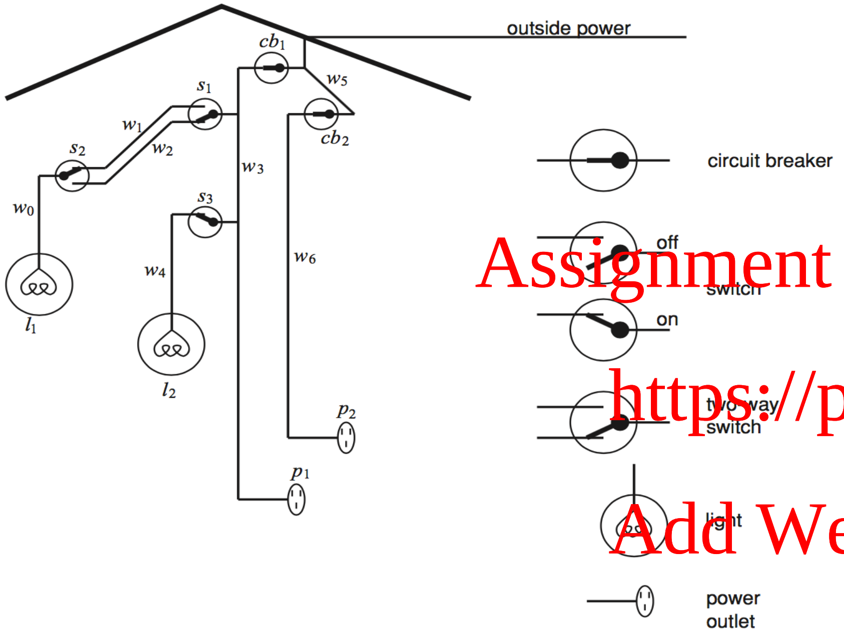
» interpretation I = <D, i > where D is a universe of discourse and i maps:

- ➢ constants to objects in D
- ➢ functions to functions over D
- ➢ predicates to tuples over D

» an interpretation and variable assignment satisfies a sentence if given the assignment the sentence is interpreted to be true.

» a sentence is satisfied if there is an interpretation and variable assignment that satisfy it.

Example: Electric Environment

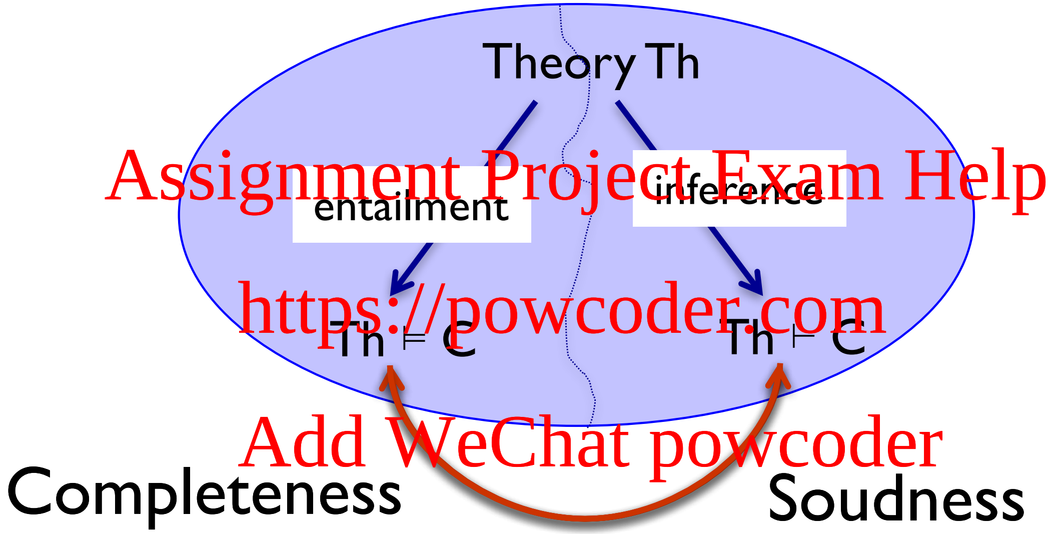


light(l1).
light(l2).
down(s1).
up(s2).
up(s3).
ok(cb1).
ok(outside).
connectedTo(l1, w0).
connectedTo(l2,w4).
live(outside).
connectedTo(w0,w1) ← up(s2).
connectedTo(w0,w2) ← down(s2).
connectedTo(w1,w3) ←up(s1).
connectedTo(w4,w3) ←up(s3).
connectedTo(w3, w5) ← ok(cb1).
connectedTo(w5, outside) ←ok(outside).
lit(L) ←light(L), live(L), ok(L).
live(X) ←connectedTo(X,Y), live(Y).

Query: ? lit(L)

Computational Logic

Predicate Logic helps modeling human reasoning



Make computation logical

Expresses relations between things using logic. Programs describe *what* to compute instead of *how* to compute

Make logic computational

Develop practical algorithms for a subset of logic that is computationally tractable.

Three forms of knowledge inference

Deductive

Reasoning from the general to reach the particular:
what follow *necessarily* from given premises.

Inductive

Reasoning from the specifics to reach the general:
process of deriving reliable *generalisations* from observations.

Abductive

Reasoning from observations to explanations:
process of using given general rules to establish *causal* relationships between existing knowledge and observations.

Three forms of knowledge inference

Deduction

Rule	*All beans in this bag are white*
Case	*These beans are from this bag*
Results	*These beans are white*

Induction

Case	*These beans are from this bag*
Results	*These beans are white*
Rule	*All beans in this bag are white*

Abduction

Rule	*All beans in this bag are white*
Results	*These beans are white*
Case	*These beans are from this bag*

Example: Electrical Environment

light(l1).
light(l2).
down(s1).
up(s2).
up(s3).
ok(cb1).
ok(outside).
connectedTo(l1, w0).
connectedTo(l2,w4).
live(outside).
connectedTo(w0,w1) ← up(s2).
connectedTo(w0,w2) ← down(s2).
connectedTo(w1,w3) ←up(s1).
connectedTo(w4,w3) ←up(s3).
connectedTo(w3, w5) ← ok(cb1).
connectedTo(w5, outside) ←ok(outside).
lit(L) ←light(L), live(L), ok(L).
live(X) ←connectedTo(X,Y), live(Y).

Deduction

Rule
Case
___
Results

live(X) ← connectedTo(X,Y), live(Y).
connectedTo(w5, outside) ← ok(outside).

live(outside).
ok(outside).
___

live(w5)

Example: Electrical Environment

light(l1).
light(l2).
down(s1).
up(s2).
up(s3).
ok(cb1).
ok(outside).
connectedTo(l1, w0).
connectedTo(l2,w4).
live(outside).
connectedTo(w0,w1) ← up(s2).
connectedTo(w0,w2) ← down(s2).
connectedTo(w1,w3) ←up(s1).
connectedTo(w4,w3) ←up(s3).
connectedTo(w3, w5) ← ok(cb1).
connectedTo(w5, outside) ←ok(outside).
lit(L) ←light(L), live(L), ok(L).
live(X) ←connectedTo(X,Y), live(Y).

Case
Results
Rules

Inductive

ok(outside).
connected(w5, outside) ← ok(outside).
live(outside).
live(w5).

live(X) ← connectedTo(X,Y), live(Y).

Example: Electrical Environment

light(l1).
light(l2).
down(s1).
up(s2).
up(s3).
ok(cb1).
ok(outside).
connectedTo(l1, w0).
connectedTo(l2,w4).
live(outside).
connectedTo(w0,w1) ← up(s2).
connectedTo(w0,w2) ← down(s2).
connectedTo(w1,w3) ←up(s1).
connectedTo(w4,w3) ←up(s3).
connectedTo(w3, w5) ← ok(cb1).
connectedTo(w5, outside) ←ok(outside).
lit(L) ←light(L), live(L), ok(L).
live(X) ←connectedTo(X,Y), live(Y).

Abduction

Rule
Results
_____
Case

live(X) ← connectedTo(X,Y), live(Y).
connected(w5, outside) ← ok(outside).
live(w5).
_____
ok(outside).
live(outside).

Clausal Representation

- Formulae in special form
  - Theory: set (conjunction) of clauses $\{p \lor \neg q;\ r;\ s\}$
  - Clause: disjuction of literals $p \lor \neg q$
  - Literal: atomic sentence or its negation p $\neg q$

- Every formula can be converted into a clausal theory

$(p \lor q) \rightarrow \neg p$	
$\neg(p \lor q) \lor \neg p$	eliminate $\rightarrow$
$(\neg p \land \neg q) \lor \neg p$	push the $\neg$ inwards
$(\neg p \lor \neg p) \land (\neg q \lor \neg p)$	distribute $\lor$ over $\land$
$\neg p \land (\neg q \lor \neg p)$	collect terms: $\neg p \lor \neg p$ gives $\neg p$

CNF

What about formulae in Predicate Logic?

Clausal Representation

- Atomic sentences may have terms with variables
  - Theory $\{p(X) \vee \neg r(a,f(b,X))\ ;\ q(X,Y)\}$
    - All variables are understood to be universally quantified

$$\forall X\,[\,(r(a,f(b,X)) \rightarrow p(X))\,] \wedge \forall X, Y\; q(X,Y)$$

- Substitution $\theta = \{v_1/t_1, v_2/t_2, v_3/t_3, \ldots.\}$

  if l is a literal, $l\theta$ is the resulting literal after substitution

$$\theta = \{X/a,\ Y/g(b,Z)\} \qquad p(X,Y)\theta = p(a,\ g(b,Z))$$

- A literal is *ground* if it contains no variables

- A literal l' is *an instance of l*, if for some θ, $l' = l\theta$

Clausal Representation

- Conversion in CNF
  - Skolemisation $\exists X\ p(X) \Rightarrow p(c)$ new constant
    $\forall X\ \exists Y\ p(X,Y) \Rightarrow \forall X\ p(X, f(X))$
  - Remove universal quantifiers

$\forall X(\neg literate(X) \rightarrow (\neg write(X) \wedge \neg \exists Y(book(Y) \wedge read(X,Y))))$

$\forall X(literate(X) \vee (\neg write(X) \wedge \neg \exists Y(book(Y) \wedge read(X,Y))))$ eliminate $\rightarrow$

$\forall X(literate(X) \vee (\neg write(X) \wedge \forall Y(\neg(book(Y) \wedge read(X,Y)))))$ push the $\neg$ inwards

$\forall X(literate(X) \vee (\neg write(X) \wedge \forall Y(\neg book(Y) \vee \neg read(X,Y))))$

$\forall X,Y(literate(X) \vee (\neg write(X) \wedge (\neg book(Y) \vee \neg read(X,Y))))$ remove $\forall$ quantifier

$literate(X) \vee (\neg write(X) \wedge (\neg book(Y) \vee \neg read(X,Y)))$

$(literate(X) \vee \neg write(X)) \wedge (literate(X) \vee \neg book(Y) \vee \neg read(X,Y)))$ distribute $\vee$

$\neg write(X)) \vee literate(X)$

$\neg book(Y) \vee \neg read(X,Y) \vee literate(X)$

Propositional resolution

- Given two clauses of the form $p \vee C_1$ and $\neg p \vee C_2$, then $C_1 \vee C_2$ is the inferred clause, called resolvent.

$w \vee r \vee q$ $\quad$ $w \vee s \vee \neg r$

$w \vee q \vee s$ $\quad$ resolvent

- Resolution is refutation complete

$Th \vDash [\,]$ iff $Th \vdash [\,]$

KB

fg
$fg \rightarrow c$
$c \wedge m \rightarrow b$
$c \wedge f \rightarrow g$
f

$\vDash g$

$KB \cup \{\neg g\} \vDash [\,]$

fg $\quad$ $\neg fg \vee c$

c $\quad$ $\neg c \vee \neg f \vee g$

$\neg f \vee g$ $\quad$ f

g $\quad$ $\neg g$

[ ]

Predicate logic resolution

Main idea: a literal (with variables) stands for all of its instances; so we can allow to infer all such instances in principle.

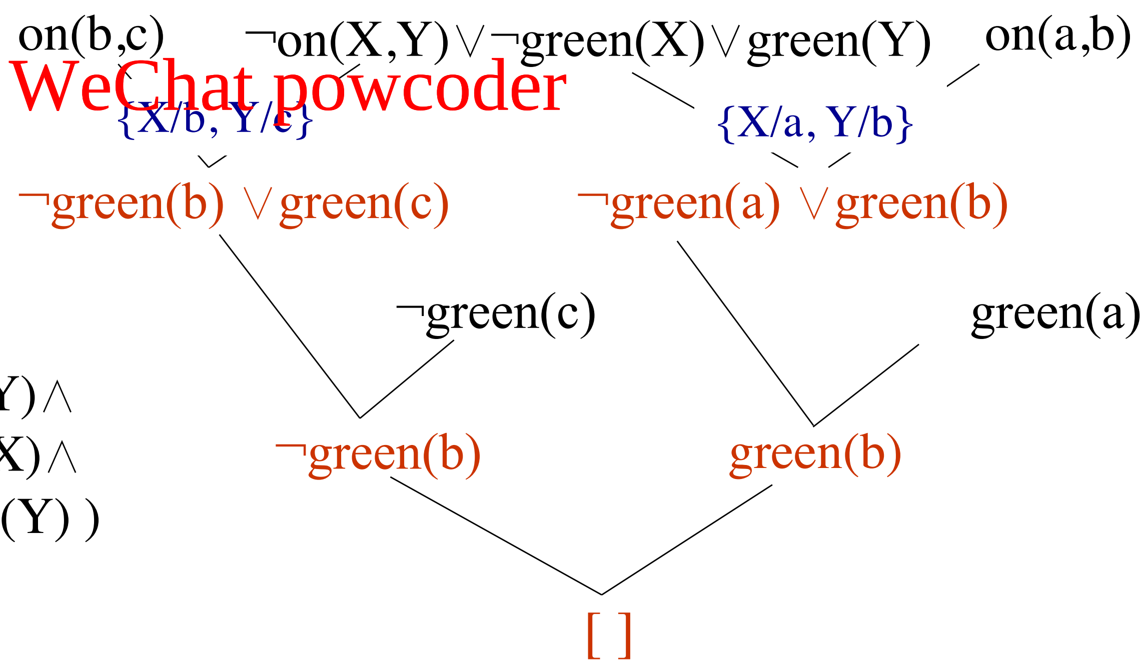
- Given two clauses of the form $\varphi_1 \vee C_1$ and $\neg\varphi_2 \vee C_2$, then
  - rename variables so that they are distinct in the two clauses φ_1 and $\neg\varphi_2$
  - for any θ such that $\varphi_1\theta = \varphi_2\theta$, then infer $(C_1 \vee C_2)\theta$ as resolvent clause
    - φ_1 unifies with φ_2 and θ is the unifier of the two literals

Example

KB

on(a,b)
on(b,c)
green(a)
¬green(c)

$\models \exists X \exists Y\, (\, on(X,Y) \wedge green(X) \wedge \neg green(Y)\,)$



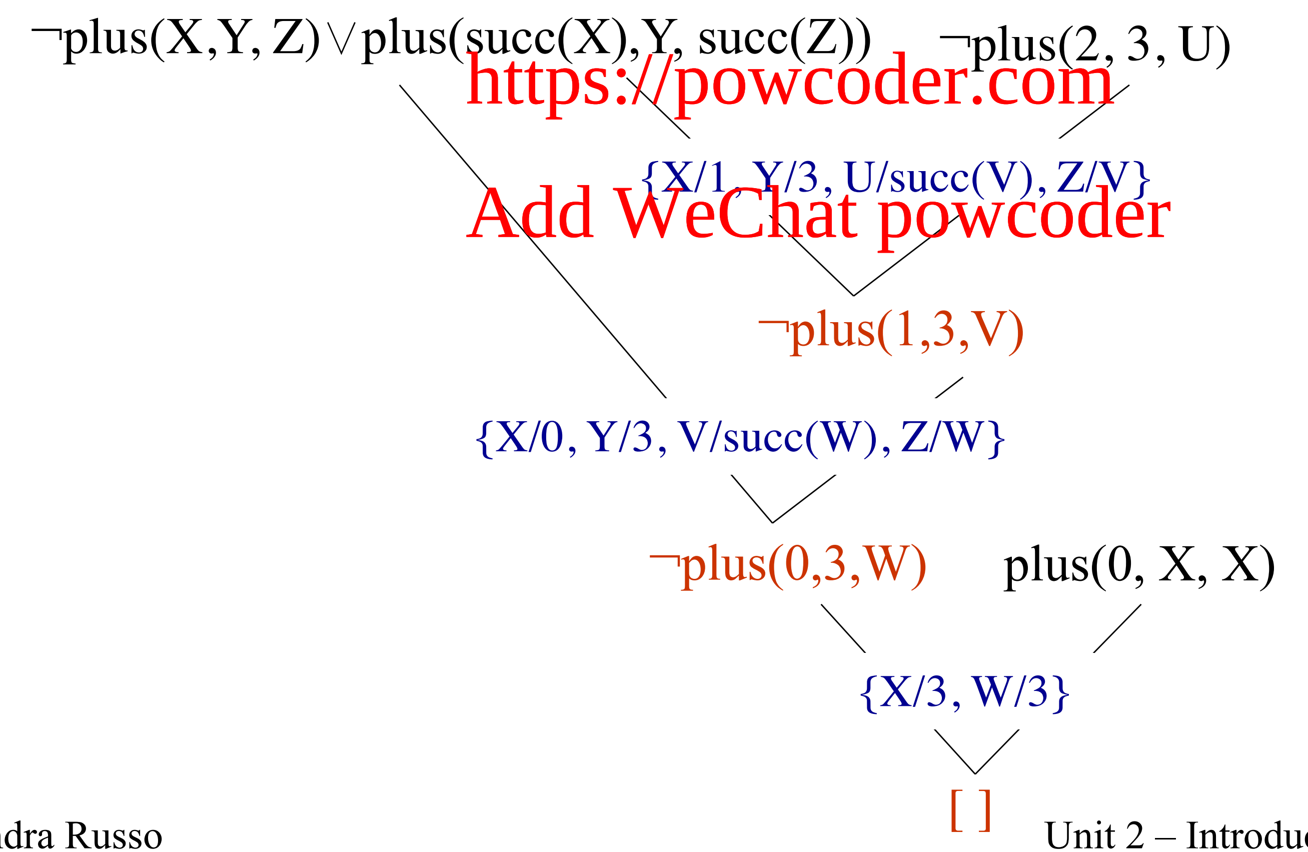
Predicate logic resolution

Answering queries may return unification values as well

KB

plus(0,X,X)
plus(X,Y,Z) $\rightarrow$ plus(succ(X), Y, succ(Z))

✓ $\models$ $\exists$U plus(2, 3, U) U = 5



Horn Clauses

Particular types of clauses with at most one positive literal.

- definite clauses exactly one positive literals $\neg b_1 \vee \neg b_2 \vee \ldots \vee \neg b_n \vee h$
- denials no positive literals $\neg b_1 \vee \neg b_2 \vee \ldots \vee \neg b_n$

Definite clauses can be represented as rules/facts, and denials as constraints:

$\neg b_1 \vee \neg b_2 \vee \ldots \vee \neg b_n \vee h$ — $h \leftarrow b_1, b_2, \ldots, b_n$ (rule)

h — h (fact)

$\neg b_1 \vee \neg b_2 \vee \ldots \vee \neg b_n$ — $\leftarrow b_1, b_2, \ldots, b_n$ (constraint)

A set of definite clauses forms a knowledge based.

A query is of the form of a denial. It can also be written as the following clause, where $X_1, \ldots, X_n$ are the variables occurring in the body literals:

$$ask(X_1,\ldots,X_n) \leftarrow b_1, b_2, \ldots, b_n \quad \text{(query)}$$

SLD derivation

SLD inference rule

$$\leftarrow \varphi_1,\ldots\varphi_n \qquad\qquad \varphi_1' \leftarrow \beta_1,\ldots,\beta_n$$

$$\leftarrow \beta_1\theta,\ldots,\beta_n\theta,\varphi_2\theta,\ldots,\varphi_n\theta$$

where θ is the mgu(φ_1, φ_1')

φ_i and β_j are atoms

SLD derivation

Given a denial (goal) G_0 and a KB of definite clauses, an SLD-derivation of G_0 from KB is a (possibly infinite) sequence of denials

$$G_0 \overset{C_0}{\Rightarrow} G_1 \quad \cdots \quad G_{n-1} \overset{C_{n-1}}{\Rightarrow} G_n$$

where G_{i+1} is derived directly from G_i and a clause C_i in the KB with variables appropriately renamed.

The composition $\theta = \theta_1\theta_2 \cdots \theta_n$ of mgus, defined in each step, gives the substitution computed by the whole derivation.

Example of SLD derivation

KB

proud(X) ← parent(X,Y), newborn(Y)
parent(X,Y) ← father(X,Y)
parent(X,Y) ← mother(X,Y)
father(adam, mary).
newborn(mary).

$\models \exists Z.\ \text{proud}(Z)$

G_0 ← proud(Z) | proud(X_0) ← parent(X_0,Y_0), newborn(Y_0) | C1

{X_0/Z}

G_1 ← parent(Z,Y_0), newborn(Y_0) | parent(X_1,Y_1) ← father(X_1,Y_1) | C2

{X_1/Z, Y_1/Y_0}

G_2 ← father(Z,Y_0), newborn(Y_0) | father(adam, mary) | C3

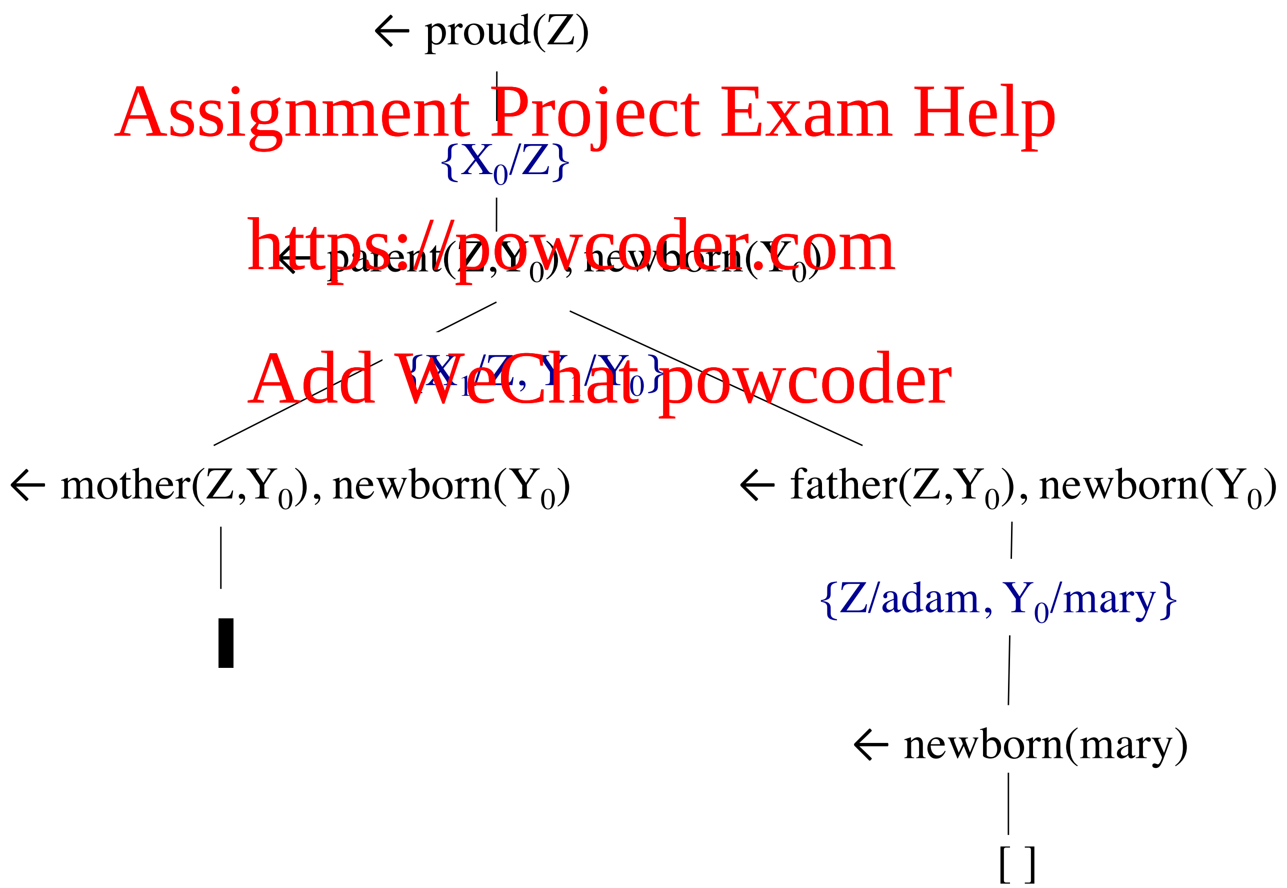
{Z/adam, Y_0/mary}

G_3 ← newborn(mary) | newborn(mary) | C4

[ ]

SLD Trees

A denial can unify with more than one clause. So multiple SLD derivations could be computed:



Normal Clausal Logic

It extends Horn Clauses by permitting atoms in the body of rules or in the denials to be prefixed with a special operator *not* (read as "fail").

Normal clauses	$h \leftarrow b_1, \ldots, b_n$, *not* $b_{n+1}, \ldots,$ *not* b_m
Normal denials	$\leftarrow b_1, b_2, \ldots, b_n,$ *not* $b_{n+1}, \ldots,$ *not* b_m

- *not* operator is the \+ used in Prolog.
- computational meaning of *not* p
  - *not* p succeeds   if and only if   p fails finitely
  - *not* p fails   if and only if   p succeeds

- fundamental constraint:

  when executing *not* p, p must be ground

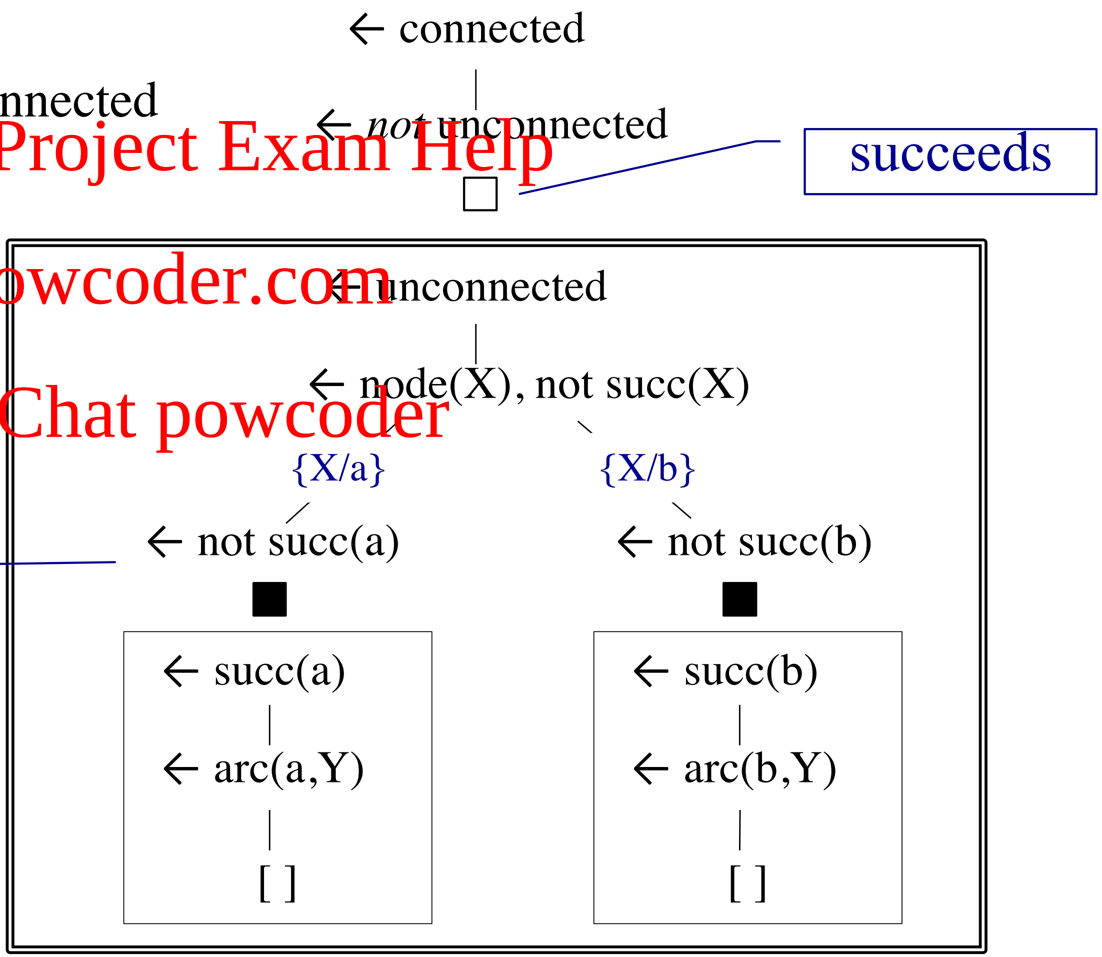
SLDNF derivation

We omit a formal definition of an SLDNF derivation

KB

connected ← *not* unconnected
unconnected ← node(X),
not succ(X)
succ(X) ← arc(X,Y)
node(a)
node(b)
arc(a, b)
arc(b, c)

$\models$ connected



SLDNF derivation

We omit a formal definition of an SLDNF derivation.

KB

connected ← *not* unconnected
unconnected ← node(X), *not* arc(X,Y)
succ(X) ← arc(X,Y)
node(a)
node(b)
arc(a, b)
arc(b, c)

⊨ connected

← connected

← *not* unconnected

floundered

← unconnected

← node(X), not arc(X,Y)

{X/a} ← not arc(a,Y)

floundered

{X/b} ← not arc(b,Y)

floundered

Any floundered branch in a tree containing no success branch (refutation) must flounder the node in the parent tree

Summary

- Propositional and predicate logic.
- Types of formal reasoning:
  deduction, abduction and induction
- Resolution: one of the main deductive proof procedures used in computational logic.
- Recap of Horn clauses and SLD resolution.
- Illustration of SLDNF for normal clauses