

Computational Logic

Maurice Bruynooghe

email Maurice.Bruynooghe@cs.kuleuven.be

- Part I Foundations of Logic Programming
 - Definite Programs
 - Normal programs

- Part II Analysis and Transformation
 - Termination analysis
 - Abstract interpretation
 - Program specialisation

literature list Part I

J.W. Lloyd, *Foundations of Logic Programming*, Springer Verlag.

K.R. Apt, *Logic Programming*, Ch. 10 in *Handbook of Theoretical Computer Science*, Ed. J. Van Leeuwen, Elsevier (1990).

K.R. Apt, *From Logic Programming to Prolog*, Prentice Hall (1997)

K.R. Apt and R.N. Bol, *Logic Programming and Negation: A Survey*, *Journal of Logic Programming*, **19/20**: 9-71, 1994. Special Issue: Ten Years of Logic Programming, Eds. M. Bruynooghe, S. Debray, M. Hermenegildo and M. Maher.

A. Van Gelder, K.A. Ross and J.S. Schlipf, *The well-founded semantics for general logic programs*, *J. ACM*, **38**(3): 620–650, July 1991.

M. Gelfond and V. Lifschitz, *The stable model semantics for logic programming*, *Proc. Fifth Int. Conf. and Symp. on Logic Programming*, eds. R.A. Kowalsi and K.A. Bowen, pp. 1070–1080, MIT press 1988.

M. Denecker, M. Bruynooghe and V. Marek, *Logic programming revisited: logic programs as inductive definitions*, *ACM Transactions on Computational Logic* **2**(4): 623–654, October 2001.

DEFINITE PROGRAMS

Proof Procedures and Semantics

1. First order Languages

- language independent part
 - variables: x, y, z, L, Ls, \dots
lower case letters x, y, z and names starting with an upper case
 - propositional constants: $true, false$
 - punctuation symbols: $() ,$
 - connectives: $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$
- language dependent/specific part
 - constants: a, b, c, d, \dots
(functors with arity 0)
 - function symbols/functors: f, g, h, \dots
(arity > 0)
 - predicate symbols: p, q, r (arity ≥ 0)

- terms
 - variables: x, y, \dots
 - constants: a, b, \dots
 - compound terms: built from n-ary functor and n-ary terms
E.g. $f(a, x), g(f(a, x)), \dots$
- formulas:
 - atoms: built from predicate and terms
E.g. $p(x, a), q(f(y)), r$
 - compound formulas: built from connectors and formulas. E.g.

formulas	schemas
$\neg p(a)$	$\neg F$
$q(b) \vee r$	$F \vee G$
$q(b) \wedge p(x)$	$F \wedge G$
$q(b) \rightarrow p(x)$	$F \rightarrow G$
$q(b) \leftrightarrow p(x)$	$F \leftrightarrow G$
$\forall x : p(x, a)$	$\forall x : F$
$\exists x : p(x, a)$	$\exists x : F$

usually predicates and functors disjoint
 not essential, is not the case in meta-programs
 $solve(p(x)) \leftarrow p(x)$

2. Logic Programs

Origin in Theorem proving

positive literal e.g. $parent(x, y)$

negative literal e.g. $\neg parent(x, y)$

- *clause*: a disjunction of literals, e.g.
 $grandpar(x, y) \vee \neg parent(x, z) \vee \neg parent(z, y)$
 $father(x, y) \vee mother(x, y) \vee \neg parent(x, y)$
 $blue(x) \vee red(x) \vee yellow(x) \vee \neg maincolor(x)$
Implicit universal quantification (\forall) over all variables
- Theory: conjunction (\wedge) of clauses
- LP uses special notation: *clausal form*
 $grandpar(x, y) \leftarrow parent(x, z), parent(z, y)$
 $father(x, y), mother(x, y) \leftarrow parent(x, y)$
 $blue(x), red(x), yellow(x) \leftarrow maincolor(x)$
clausal form: $head \leftarrow body$
head: disjunction (\vee) of atoms
the positive literals of the clause
body: conjunction (\wedge) of atoms
the negative literals of the clause

- *definite* (program) clause: clausal form
ONE (pos) literal in the head e.g.
 $grandpar(x, y) \leftarrow parent(x, z), parent(z, y)$
- *definite* goal or negative clause
NO literal in the head e.g.
 $\leftarrow grandpar(x, floris)$
query: conjunction in body of goal
- extension:
normal (general) clause/program/goal:
also negative literal allowed in body e.g.
 $father(x, y) \leftarrow parent(x, y), \neg mother(x, y)$
- extension:
disjunctive clause/program:
more than one literal in head e.g.
 $father(x, y), mother(x, y) \leftarrow parent(x, y)$
- empty body: fact
- one literal: unit clause

3. Semantics of First order Logic

- Language independent part ($\neg, \exists, true \dots$): fixed meaning
- language specific part (constants, functors, predicates):
interpretation provides mapping between symbols and meaning: terms to objects, atoms to truth
- theory: model of some world: intended interpretation maps symbols to modelled world. Hence theory should be true in the intended interpretation.
- If theory also true in other (non isomorphic) interpretations, then the formalisation is incomplete (too weak).

interpretation:

- a domain of objects (objects in the world)
- functions over the domain
- relations over the domain

Formally: interpretation I of L :

- a pre-interpretation
 - a non-empty domain D
 - for each constant c in L an assignment c_I of an element in D
 - for each n -ary functor f/n in L an assignment $f_I : D^n \rightarrow D$
- for each n -ary predicate p/n in L : an assignment p_I : a n -ary relation over D^n

with $n = 0$ there are two relations, the empty relation (*false*) and the relation with the empty tuple (*true*)

What about variables?

a *state* or *variable assignment*

a function V from the variables to D

Variable assignment V^v can be extended into *term assignment* V :

$V(t) =$ if $Var(t)$ then $V^v(t)$
else if $constant(t)$ then t_I its pre-interpretation
else with $t = f(t_1, \dots, t_n)$: $f_I(V(t_1), \dots, V(t_n))$

truth with respect to V and I :

- of an atom $p(t_1, \dots, t_n)$:
it is true, i.e. $I \models_V p(t_1, \dots, t_n)$ if
 $\langle V(t_1), \dots, V(t_n) \rangle$ is in the relation p_I
- of a compound formula:
- $I \models_V \neg F$ iff $\neg(I \models_V F)$
- $I \models_V F \vee G$ iff $I \models_V F \vee I \models_V G$
- $I \models_V \forall x : F$ iff $I \models_{V[x/d]} F$ for all $d \in D$

$V[x/d]$ same variable assignments as V except for x :

$V[x/d](y) =$ if $y = x$ then d else $V(y)$

other forms can be reduced:

- $I \models_V F \wedge G$ iff $I \models_V \neg(\neg F \vee \neg G)$
- $I \models_V F \rightarrow G$ iff $I \models_V \neg F \vee G$
- $I \models_V F \leftrightarrow G$ iff $I \models_V (F \rightarrow G) \wedge (G \rightarrow F)$
- $I \models_V \exists x : F$ iff $I \models_V \neg \forall x : \neg F$

some can also be defined directly

- $I \models_V F \wedge G$ iff $I \models_V F \wedge I \models_V G$
- $I \models_V \exists x : F$ iff $I \models_{V[x/d]} F$ for at least one $d \in D$

about empty formula \square :

formula: $head \leftarrow body$

$head$: empty disjunction: *false*

$body$: empty conjunction: *true*

false \leftarrow *true* is *false*

formula F is *true* in interpretation I iff
 F is *true* for all variable assignments V
notation: $I \models F$

with S a set of formulas (program)
 I is a model of S iff $I \models F$ for all F in S

S is *satisfiable/consistent*: S has a model

S is *UNsatisfiable/INconsistent*: S has NO
model

S is *valid*: S has ONLY models (every I is a
model)

$S \models F$: F is true for all interpretations for
which S is true, i.e. every model of S is also
a model of F

Examples

L : symbols of F

F : $father(maurice, floris)$

pre-interpretation J :

- domain $D = \{MB, FB\}$
- $J(maurice) = MB$
- $J(floris) = FB$

interpretation I_1 : $J +$

$I_1(father/2) = \{\langle MB, FB \rangle\}$

$J(\langle maurice, floris \rangle) = \langle MB, FB \rangle \in I_1(father/2)$

thus I_1 is model of F

interpretation I_2 : $J +$

$I_2(father/2) = \{\langle FB, FB \rangle\}$

I_2 is not a model of F

F is satisfiable

L : constant a , functor $s/1$, predicate $r/2$

$F : r(s(x), x)$

pre-interpretation J :

$D = N$ (the natural numbers)

$J(a) = 0$

$J(s(n)) = n + 1$

interpretation I_1 : $I_1(r/2) = \{\langle n, m \rangle \mid m = n - 1\}$

Consider $V(x) = m$ (a natural number):

$J(\langle s(x), x \rangle) = \langle m + 1, m \rangle \in I_1(r/2)$

F is true for all variable assignments V thus I_1 is a model

interpretation I_2 : $I_2(r/2) = \{\langle n, m \rangle \mid m = n + 1\}$

Consider $V(x) = m$:

$J(\langle s(x), x \rangle) = \langle m + 1, m \rangle \notin I_2(r/2)$

F is false for all variable assignments V thus I_2 is not a model

F is satisfiable

inconsistent formulas: $p \wedge \neg p$, \square , ...

valid formulas: $p \vee \neg p$, $p \rightarrow p$, ...

what are valid and inconsistent formulas good for?

typical question: $T \models F$?

i.e. Is F true in all models of T ?

If so then $T \rightarrow F$ is valid

and $T \wedge \neg F$ is inconsistent

e.g. $T = \text{father}(m, f)$, $F = \exists x : \text{father}(m, x)$

When neither $T \rightarrow F$ nor $T \rightarrow \neg F$ are valid:
the description/knowledge expressed by T is
not sufficient (“incomplete”) to settle the
truth of F

same T is *complete* (enough) for some F , incomplete for others.

non trivial questions, FOL is expressive!

e.g. T : axiomatisation of natural numbers

is $T \rightarrow \exists a, b, c, n : n > 2 \wedge a^n + b^n = c^n$ valid?

You need a good understanding of how interpretations are constructed, hence work out some examples:

1. Given the formula F : $\text{equal}(\text{plus}(X,Y),\text{plus}(Y,X))$

Find an interpretation I based on a finite domain and variable assignments V_1 and V_2 such that:

a. $I \models_{V_1} F$

b. $I \not\models_{V_2} F$

Find interpretations I_1 and I_2 based on an infinite domain such that:

a. $I_1 \models \forall F$

b. $I_2 \not\models \forall F$

2. Given the clausal theory T :

$\text{even}(0) \leftarrow$

$\text{even}(s(X)) \leftarrow \text{odd}(X)$

$\text{odd}(s(X)) \leftarrow \text{even}(X)$

$\text{false} \leftarrow \text{even}(X), \text{odd}(X)$

Find interpretations I_1 and I_2 based on a finite domain such that:

a. $I_1 \models T$

b. $I_2 \not\models T$

Is the theory inconsistent, valid or satisfiable?

4. Substitutions

What: function mapping expressions (terms, formulas,...) to expressions by replacing variables by terms

Representation: $\theta = \{x_1/t_1, \dots, x_n/t_n\}$

x_i : variable to be replaced

t_i : term replacing the variable

So: all variables different

$$x_i \neq t_i$$

Result: *simultaneous* replacing all variable occurrences by corresponding term occurrences.

Notation $E\theta$ (rather than $\theta(E)$)

$E\theta$ is an *instance* of E

Examples

$$f(x, y, z)\{x/y, y/a\} = f(y, a, z)$$

$$f(x, y, z)\{x/y\}\{y/a\} = f(y, y, z)\{y/a\} = f(a, a, z)$$

Renaming Substitutions

purpose change names of variables in reversible way

motivation names often irrelevant

e.g. $\forall x : p(x)$ and $\forall y : p(y)$

different formalisations possible

Apt:

Def.: a renaming substitution maps variables to a permutation of the variables

e.g. $\theta = \{y/y', z/z', y'/y, z'/z\}$

$f(x, y, z)\theta = f(x, y', z')$

$g(y', y, z)\theta = g(y, y', z')$

Def.: $E\theta$ is a *variant* of E (θ a renaming)

Lemma: E variant of F iff E instance of F and
 F instance of E

Lloyd:

Def.: E and F variants if instances of each other.

Def.: renaming substitution *for* E : of the form $\{x_1/y_1, \dots, x_n/y_n\}$ such that:

$x_i \in \text{Var}(E)$ and

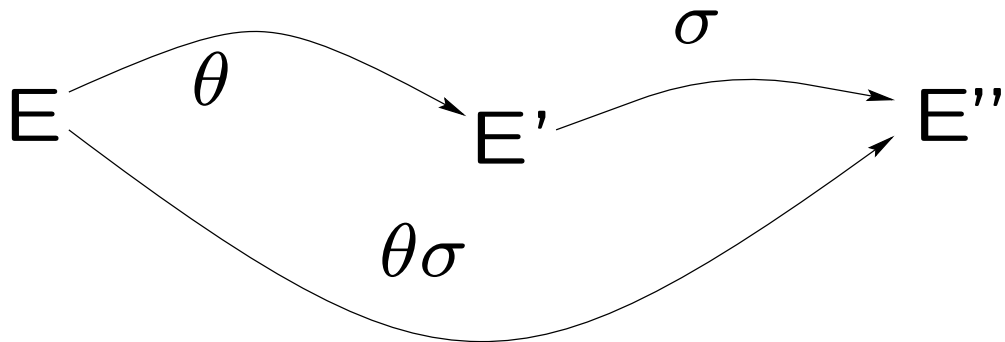
to preserve reversibility:

$y_i \notin \text{Var}(E) \setminus \{x_1, \dots, x_n\}$ and all distinct

E.g. $\{y/y', z/z'\}$ is renaming for $f(x, y, z)$ but not for $g(y', y, z)$

Proposition: if E and F are variants, then a renaming substitution exists.

composition



$$\theta = \{x_1/t_1, \dots, x_n/t_n\}, \quad \sigma = \{y_1/s_1, \dots, y_m/s_m\}$$

$$\theta\sigma =$$

$$\{x_1/t_1\sigma, \dots, x_n/t_n\sigma, y_1/s_1, \dots, y_m/s_m\} \setminus \dots$$

$\setminus \dots$: removal of:

$$x_i/t_i\sigma \text{ if } x_i = t_i\sigma$$

$$y_i/s_i \text{ if } y_i \in \{x_1, \dots, x_n\}$$

Lemma: associativity

$$(E\theta)\sigma = E(\theta\sigma)$$

$$(\theta\sigma)\gamma = \theta(\sigma\gamma)$$

θ more general than σ if $\exists \gamma : \sigma = \theta\gamma$

some authors use \geq other \leq

about *more general*

Is $\{x/f(y)\}$ more general than $\{x/f(a)\}$?

NO $\{x/f(y)\}\{y/a\} = \{x/f(a), y/a\}$

Counter intuitive! origin of many subtle errors

One solution: use ordering between equivalence classes of terms

$t \geq s$ iff $s = t\theta$ for some θ e.g. $f(x) \geq f(a)$

transitive, reflexive: pre-order

if $t \geq s$ and $s \geq t$ then t and s are *variants*

variants: transitive, reflexive, symmetric: equivalence relation

notation for the equivalence class of t : $[t]_{\equiv}$

e.g. $[f(x)]_{\equiv} = \{f(x), f(y), \dots\}$

partial order (transitive, reflexive, anti-symmetric)

between equivalence classes:

$[t]_{\equiv} \succeq [s]_{\equiv}$ iff $t \geq s$ e.g. $[f(y)]_{\equiv} \succeq [f(a)]_{\equiv}$

Other solution: use a reference domain

$\theta \geq_V \sigma$ iff $\exists \gamma : \theta\gamma \upharpoonright_V = \sigma \upharpoonright_V$

where $\sigma \upharpoonright_V$ is the projection on V

i.e. $\{x_i/t_i \mid x_i/t_i \in \sigma \text{ and } x_i \in V\}$

e.g. $\{x/f(y)\} \geq_{\{x,z\}} \{x/f(a)\}$

5. Unifiers

θ is unifier of A and B iff $A\theta = B\theta$

Most general unifier: more general than any other.

e.g. $\{x/y\}$ and $\{y/x\}$ are most general unifiers of x and y .

$\{x/z, y/z\}$ is not most general.

Some Unification Algorithms (for two terms A and B)

1. Robinson's algorithm (1963)

$\sigma \leftarrow \epsilon$ (empty substitution)

$D \leftarrow \text{disagreementset}(A, B)$

while $D \neq \emptyset$ **do**

if $D = \{x, t\}$, x does not occur in t

then $\sigma \leftarrow \sigma\{x/t\}$

else stop, no unifier

$A \leftarrow A\{x/t\}; B \leftarrow B\{x/t\}$

$D \leftarrow \text{disagreementset}(A, B)$

— σ is the mgu

The *disagreement set* is the “first” pair of sub-terms in A and B which are different.

Example:

$$\sigma = \epsilon$$

$$A = f(g(x), g(y)) \quad B = f(y, g(g(z)))$$

$$D = \{g(x), y\}$$

$$\sigma = \{y/g(x)\}$$

$$A = f(g(x), g(g(x))) \quad B = f(g(x), g(g(z)))$$

$$D = \{x, z\}$$

$$\sigma = \{y/g(z), x/z\}$$

$$A = f(g(z), g(g(z))) \quad B = f(g(z), g(g(z)))$$

$$D = \emptyset$$

Can be very expensive, e.g.:

unify $g(x_1, g(x_2, g(\dots, g(x_n, x_{n+1}) \dots)))$ and

$g(f(x_0, x_0), g(f(x_1, x_1), g(\dots, g(f(x_{n-1}, x_{n-1}), f(x_n, x_n)) \dots)))$

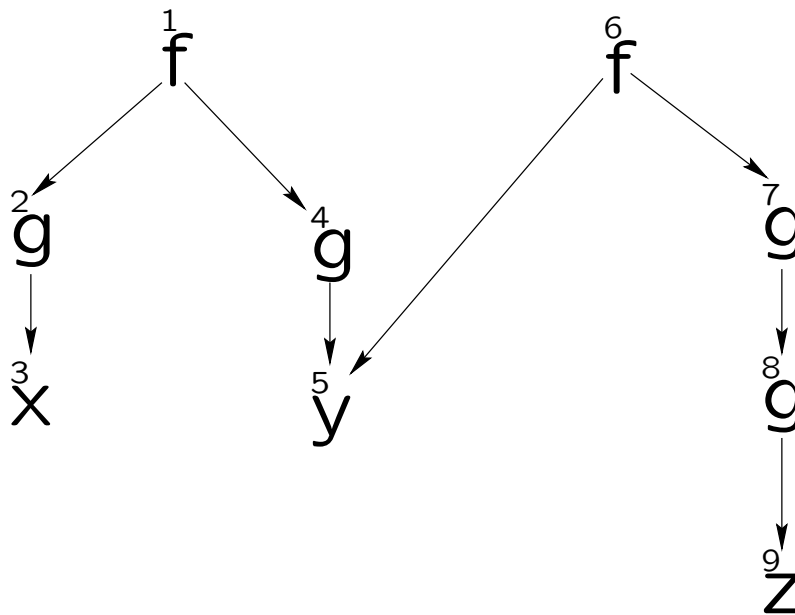
But is approximately the algorithm used in LP systems

Differences: terms as trees, with sharing of identical subterms possible

Sharing of variables obligatory (applying x/t is simply setting pointer from x to t)

No occur check, but prevention of infinite loops (unify x and $g(x)$)

2. Paterson and Wegman (1978)
 (Starting point of an algorithm linear in input size)
 Represents terms as *directed acyclic graphs* (dag)



Example:
 built equivalence classes: $1 \sim 6$
 propagate $\{1, 6\} : 2 \sim 5, 4 \sim 7$
 propagate $\{2, 5\} : \text{nothing}$
 propagate $\{4, 7\} : 5 \sim 8$
 propagate $\{2, 5, 8\} : 3 \sim 9$

reconsider arcs of dag as arcs between equivalence classes:

$\{1, 6\} \rightarrow \{2, 5, 8\}, \{1, 6\} \rightarrow \{4, 7\}$
 $\{2, 5, 8\} \rightarrow \{3, 9\}, \{4, 7\} \rightarrow \{2, 5, 8\}$
 NO cycle

algorithm

let u and v be roots of dags representing A and B

$u \sim v$ (in same equivalence class)

while there is a pair of nodes r, s such that $r \sim s$ but with corresponding children r' and s' such that $\text{not}(r' \sim s')$

do *propagation*: $r' \sim s'$

if classes *homogeneous* (no different functor labels in a equiv. class) and *no cycle* between classes

then extract mgu **else** not unifiable

example extraction: $\{2, 5, 8\} : \{y/g(x)\}, \{3, 9\} : \{z/x\}$

Optimisation:

each class is propagated only once:

select always *root* class for propagation: a class such that classes of parent nodes have been propagated

keep count of number of incoming arcs

amount of work linear in size of input

3. Equation solving (Herbrand 1930 Martelli-Montanari 1982)

Also start of a linear unification algorithm

Manipulates set of equations until set has a trivial mgu

Each step preserves the set of unifiers

Set with trivial mgu: *solved form*:

$$\{x_1 = t_1, \dots, x_n = t_n\}$$

all x_i different

no x_i does occur in right hand side

trivial mgu: $\{x_1/t_1, \dots, x_n/t_n\}$

example:

$$\{f(g(x), g(y)) = f(y, g(g(z)))\}$$

peel of first equation

$$\{g(x) = y, g(y) = g(g(z))\}$$

switch of first equation

$$\{y = g(x), g(y) = g(g(z))\}$$

substitute first equation in rest

$$\{y = g(x), g(g(x)) = g(g(z))\}$$

peel second equation

$$\{y = g(x), g(x) = g(z)\}$$

peel second equation

$$\{y = g(x), x = z\}$$

substitute second equation in rest

$$\{y = g(z), x = z\}: \text{ solved form}$$

algorithm

repeat select equation with applicable rule
until no rule applicable

Rules:

peel: $\{f(s_1, \dots, s_m) = g(t_1, \dots, t_n), e_2, \dots, e_k\} \rightarrow$

if $f/m = g/n$

then $\{s_1 = t_1, \dots, s_n = t_n, e_2, \dots, e_k\}$

else fail

remove: $\{x = x, e_2, \dots, e_k\} \rightarrow \{e_2, \dots, e_k\}$

switch:

$\{t = x, e_2, \dots, e_k\}$ and t is not a variable

$\rightarrow \{x = t, e_2, \dots, e_k\}$

substitute:

$\{x = t, e_2, \dots, e_k\}$ and $x \in Var(t) \rightarrow$ fail

$\{x = t, e_2, \dots, e_k\}$ and $x \notin Var(t), x \in Var(e_2, \dots, e_k)$

$\rightarrow \{x = t, e_2\{x/t\}, \dots, e_k\{x/t\}\}$

alternative: drop $x \in Var(e_2, \dots, e_k)$, repeat until *fixed point*

Terminates

With fail or solved form

Each step preserves all unifiers

Termination:

Consider tuple: $\langle \# \text{unsolved variables, } \# \text{functors, } \# \text{equations of the form } x = x \text{ or } t = x \rangle$
 $\langle n_1, n_2, n_3 \rangle < \langle m_1, m_2, m_3 \rangle$ if $n_1 < m_1$ or $n_1 = m_1, n_2 < m_2$ or $n_1 = m_1, n_2 = m_2, n_3 < m_3$

peel: $\langle n_1, n_2, n_3 \rangle \rightarrow \langle n_1 - ?, n_2 - 2, n_3 + ? \rangle$

remove: $\langle n_1, n_2, n_3 \rangle \rightarrow \langle n_1 - (0 \text{ or } 1), n_2, n_3 - 1 \rangle$

switch: $\langle n_1, n_2, n_3 \rangle \rightarrow \langle n_1 - (0 \text{ or } 1), n_2, n_3 - 1 \rangle$

substitute: $\langle n_1, n_2, n_3 \rangle \rightarrow \langle n_1 - 1, ?, ? \rangle$

No infinite decreasing sequence exists (*well founded order*)

preservation of unifiers: trivial for
peel, remove, switch.

substitute preserves unifiers

$$\left. \begin{array}{l} x = t \\ r = s \end{array} \right\} \rightarrow \left\{ \begin{array}{l} x = t \\ r\{x/t\} = s\{x/t\} \end{array} \right.$$

a. unifier initial system is unifier final system

Unifier θ of the initial system must be of the form

$$\theta = \{x/t\gamma\} \cup \gamma$$

Is θ unifier of the final system?

Of $x = t$? Yes, that equation was in the initial system

Of $r\{x/t\} = s\{x/t\}$?

$$r\{x/t\}\theta = r\{x/t\}(\{x/t\gamma\} \cup \gamma) = r(\{x/t\}(\{x/t\gamma\} \cup \gamma)) = r(\{x/t\gamma\} \cup \gamma) = r\theta$$

$$\text{similar } s\{x/t\}\theta = \dots = s\theta$$

Yes because the equation reduces to $r\theta = s\theta$ and θ is unifier of the initial system which has equation $r = s$

b. unifier final system is unifier initial system:

Unifier θ of the final system must be of the form $\theta =$

$$\{x/t\gamma\} \cup \gamma$$

Is θ unifier of the initial system?

Of $x = t$? Yes, that equation was in the final system

Of $r = s$?

θ is unifier of $r\{x/t\} = s\{x/t\}$ i.e. $r\{x/t\}\theta = s\{x/t\}\theta$

The left hand side reduces to $r\theta$, the rhs to $s\theta$ (see above)

So θ is unifier of r and s

properties of mgu's

idempotent substitution: $\theta = \theta\theta$

A substitution $\{x_1/t_1, \dots, x_n/t_n\}$ is idempotent iff for all $i, j : x_i \notin \text{Var}(t_j)$

A unifier θ of term A, B is *relevant* iff it contains only variables from A and B .

Proposition: Unification algorithms produce an mgu which is *idempotent* and *relevant*

Lemma: $E \xrightarrow{\theta} E\theta$

An idempotent substitution $\theta (= \{\dots, x_i/t_i, \dots\})$ eliminates all variables from the domain of θ (\dots, x_i, \dots) from E :

$$\text{Vars}(E\theta) \cap \text{dom}(\theta) = \emptyset$$

Some exercises on substitutions and unification

1. Given the substitutions $\sigma = \{X/f(Z), Y/g(U)\}$ and $\tau = \{X/f(a), Y/g(W), U/W\}$.
 - a. Is σ more general than τ ?
 - b. Give another substitution similar to σ which is more general than τ .
 - c. Give another substitution similar to τ such that σ is more general than it.

2. Apply the three unification algorithms on the following pairs of terms:
 - a. $p(X,a)$ and $p(b,Y)$
 - b. $p(a)$ and $p(b)$
 - c. $p(X,f(g(X)))$ and $p(f(Y),f(Y))$

6. SLD-proof procedure (Definite programs)

SLD-resolution

Definite program P

Definite goal (negative clause) $N = \leftarrow A_1, \dots, A_n$

Clause $C = A \leftarrow B_1, \dots, B_k \in P$

$mgu(A, A_i) = \theta$

resolvent of N and C is:

$N' = \leftarrow (A_1, \dots, A_{i-1}, B_1, \dots, B_k, A_{i+1}, \dots, A_n)\theta$

four steps:

1. *Select* A_i
2. *unify* A and A_i
3. if success then *replace* A_i by B_1, \dots, B_k
4. *apply* θ

SLD-derivation

from def. program P and def. goal N

maximal sequence (if not, it is a *partial* SLD-derivation)

$$\begin{array}{lll} N_0 = N & C_0 & \theta_0 \\ N_1 & C_1 & \theta_1 \\ N_2 & C_2 & \theta_2 \\ \vdots & \vdots & \vdots \\ N_{i+1} & C_{i+1} & \theta_{i+1} \\ \vdots & \vdots & \vdots \end{array}$$

C_i : a *standardised apart* variant of a clause in P , i.e. its variables have not been used in the derivation so far.

N_{i+1} : the *resolvent* of N_i, C_i

θ_i : the mgu obtained from unifying head of C_i and selected atom in N_i

refutation:

finite SLD-derivation ending with \square

failure:

finite SLD-derivation, not a refutation so last goal is non empty but no clause head unifies with selected atom

Later we will show that SLD-derivation is a sound inference rule, i.e. derives formulas which are true in every model of the initial theory

refutation from P and N : $P \wedge N$ is an inconsistent theory i.e. (assuming P is consistent) N is false in every model of P hence $\neg N$ is true in every model of P : $P \models \neg N$.

$$\begin{aligned}
 N &= \leftarrow A_1, \dots, A_n \\
 &= \forall x_1, \dots, x_n (\text{false} \leftarrow A_1 \wedge \dots \wedge A_n) \\
 &= \forall x_1, \dots, x_n (\neg A_1 \vee \dots \vee \neg A_n) \\
 \neg N &= \neg (\forall x_1, \dots, x_n (\neg A_1 \vee \dots \vee \neg A_n)) \\
 &= \exists x_1, \dots, x_n \neg (\neg A_1 \vee \dots \vee \neg A_n) \\
 &= \exists x_1, \dots, x_n (A_1 \wedge \dots \wedge A_n)
 \end{aligned}$$

To construct \square , the SLD-derivation was applying $\theta_0, \theta_1, \dots$ on initial goal N :

The composition $\theta_0, \theta_1, \dots$ gives bindings for the variables x_1, \dots, x_n such that

$P \rightarrow (A_1, \dots, A_n)\theta_0\theta_1 \dots$ is a valid theory i.e.

$P \models (A_1, \dots, A_n)\theta_0\theta_1 \dots$ as we will show later

computed answer substitution (cas):
restriction of $\theta_0\theta_1 \dots$ to $Var(N_0)$

history: partial derivation up to current resolvent, say N_k

computation rule or *selection rule* R

Function: input: history (assume N_k last)

result: an atom from N_k

only N_k as input is sometimes not flexible enough -
given $N_k = N_i$ one may wish to select a different atom
because the history is different (e.g. to prevent infinite
derivation)

SLD-derivation via R: using R as computation rule

S: with Selection rule

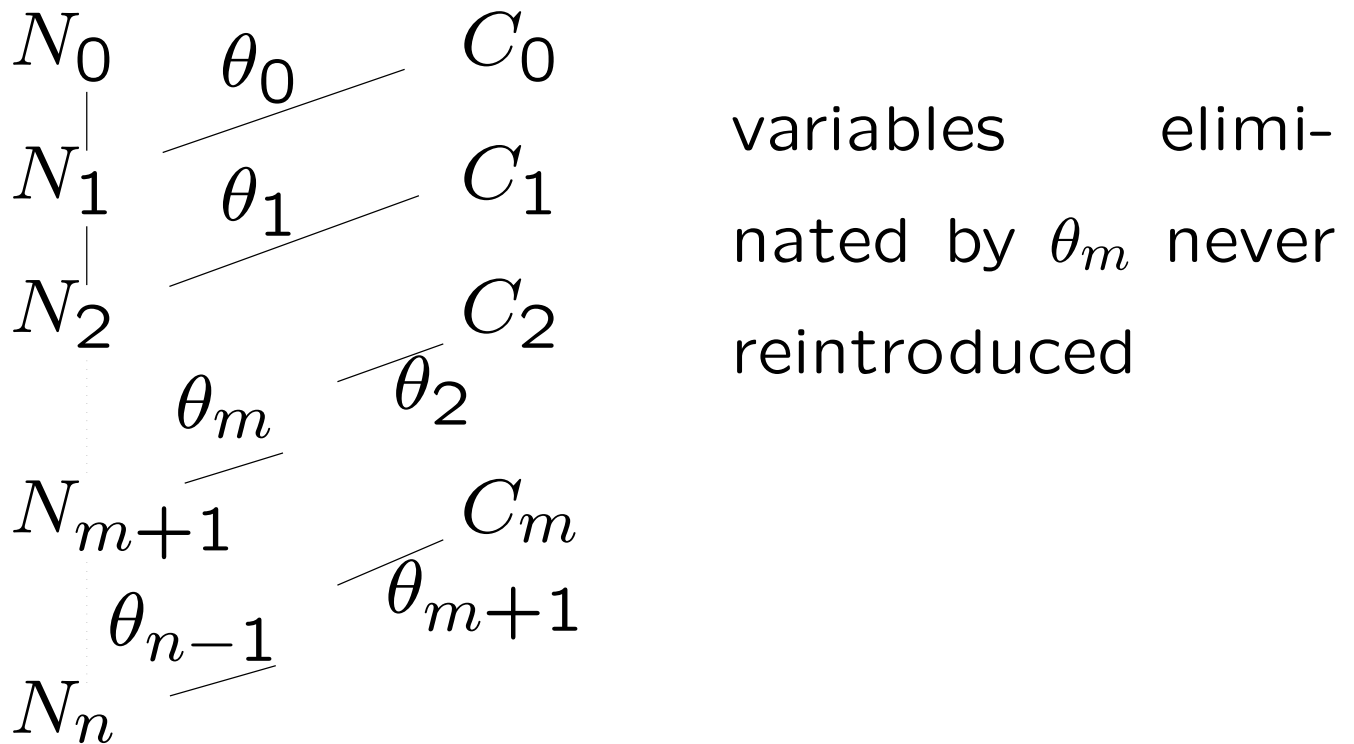
L: Linear, i.e. derivation has a *linear* structure: every goal N_k is derived from the previous goal N_{k-1} and a program clause (not so for resolution in general)

D: for Definite programs

Selection rule driven *Linear* resolution for *Definite* clauses

properties of SLD-derivations

Lemma



With $n > m$:

$$\text{Var}(N_n) \cap \text{dom}(\theta_m) = \emptyset$$

$$\text{Var}(N_n\theta_n) \cap \text{dom}(\theta_m) = \emptyset$$

Proof by induction (see Apt Lemma 2.7) Idea: θ_m is idempotent and relevant, so variables of $\text{dom}(\theta_m)$ are eliminated by applying θ_m i.e.

$$\text{Var}(N_{m+1}) \cap \text{dom}(\theta_m) = \emptyset$$

Are not reintroduced by the following steps as C_{m+i} are variants with new variables and as θ_{m+i} are relevant and idempotent.

Consider partial SLD-derivation:

$$N_0 = \leftarrow A_1, \dots, A_k$$

:

$$N_i = \leftarrow B_1, \dots, B_m$$

the *resultant* at level i is

$$(A_1 \wedge \dots \wedge A_k)\theta_0 \dots \theta_{i-1} \leftarrow B_1 \wedge \dots \wedge B_m$$

With $k = 1$:

$$\text{the clause } A_1\theta_0 \dots \theta_{i-1} \leftarrow B_1, \dots, B_m$$

Variant Lemma

Consider partial SLD-derivation $N_0 \dots N_i$

Consider a second derivation with N'_0 a variant of N_0 , with at each step:

- it selects the atom at the same position as the first (same computation rule)
- C'_i is a variant of C_i

Then N'_i is a variant of N_i , moreover the resultants at level i are also variants.

Names of variables do not matter

Corollary:

Consider SLD-derivation $N_0 \dots \square$ with cas (comp. answ. subst.) θ

Consider second derivation starting also from N_0 , which selects atoms at the same position but uses *other* variants of the clauses C_i , and which has cas σ

Then $N_0\theta$ and $N_0\sigma$ are variants i.e. $N_0\theta$ is more general than $N_0\sigma$ and $N_0\sigma$ is more general than $N_0\theta$

However θ is NOT more general than σ e.g. $p(f(y)). \leftarrow p(x).$

Both $\sigma = \{x/f(y)\}$ and $\theta = \{x/f(z)\}$ are cas. But $\theta \geq_{Var(N_0)} \sigma$ and $\sigma \geq_{Var(N_0)} \theta$.

SLD-tree via R

Each branch is a SLD-derivation

Each node: a child for each clause with head unifying with selected atom

Shape and size determined by R

$$1. p(x, z) \leftarrow a(x, y), p(y, z) \quad \leftarrow \underline{p(x, c)}$$

$$2. p(x, x)$$

$$3. a(b, c)$$

$$\{z_1/c, x_1/x\}$$

$$\leftarrow \underline{a(x, y_1)}, p(y_1, c) \quad \{x/c, x_1/c\}$$

$$\{x/b, y_1/c\}$$

□ refutation

$$\leftarrow \underline{p(c, c)}$$

$$\{x_2/c, z_2/c\}$$

$$\{x_2/c\}$$

$$\leftarrow \underline{a(c, y_2)}, p(y_2, c)$$

□ refutation

failure

$$\leftarrow \underline{p(x, c)}$$

$$\{z_1/c, x_1/x\}$$

$$\{x/c, x_1/c\}$$

$$\leftarrow a(x, y_1), \underline{p(y_1, c)}$$

□ refutation

$$\{x_2/y_1, z_2/c\}$$

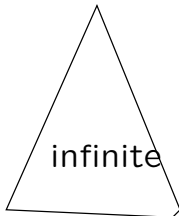
$$\{y_1/c, x_2/c\}$$

$$\leftarrow a(x, y_1), a(y_1, y_2), \underline{p(y_2, c)}$$

$$\leftarrow \underline{a(x, c)}$$

$$\{y_2/c, x_3/c\}$$

$$\{x/b, x_3/c\}$$



$$\leftarrow a(x, y_1), \underline{a(y_1, y_2)}$$

$$\{y_1/b, y_2/c\}$$

□ refutation

$$\leftarrow \underline{a(x, b)}$$

failure

An exercise about proof trees

1. Consider the program:

$p(X,Y) \leftarrow q(X,Y), r(Y).$

$q(X,h(Y)) \leftarrow q(X,Y).$

$r(g(X)) \leftarrow.$

$r(a) \leftarrow.$

and the query $\leftarrow p(X,Y).$

Choose a selection rule and draw the SLD-tree. What can you conclude about the query.

7. Semantics of SLD

7.1 Soundness of SLD-resolution

Lemma Soundness of resolution step

Given:

goal: $\leftarrow \mathcal{A} = \leftarrow A_1, \dots, A_n$

clause $C = A \leftarrow B_1, \dots, B_k$

$$\theta = mgu(A, A_i)$$

resolvent $\leftarrow \mathcal{B} =$

$\leftarrow A_1\theta, \dots, A_{i-1}\theta, B_1\theta, \dots, B_k\theta, A_{i+1}\theta, \dots, A_n\theta$

then:

$C \models \mathcal{B} \rightarrow \mathcal{A}\theta$

$A \leftarrow B_1, \dots, B_k \models$

$A_1\theta, \dots, A_{i-1}\theta, B_1\theta, \dots, B_k\theta, A_{i+1}\theta, \dots, A_n\theta$

$\rightarrow A_1\theta, \dots, A_i\theta, \dots, A_n\theta$

Proof

To show: $I(\mathcal{B} \rightarrow \mathcal{A}\theta)$ is *true* for every interpretation I for which $I(C)$ is *true*

Two cases:

- $I(\mathcal{B})$ is *false* then $I(\mathcal{B} \rightarrow \mathcal{A}\theta)$ is *true*
- $I(\mathcal{B})$ is *true* i.e. $I(B_1\theta, \dots, B_k\theta)$ is *true* and $I(A_1\theta, \dots, A_{i-1}\theta, A_{i+1}\theta, \dots, A_n\theta)$ is *true*

I is model of C i.e. $I(\mathcal{A}\theta)$ is *true* (because body is *true*), but $\mathcal{A}\theta = A_i\theta$, so $I(A_i\theta)$ is *true*

Thus $I(A_1\theta, \dots, A_{i-1}\theta, A_i\theta, A_{i+1}\theta, \dots, A_n\theta) = I(\mathcal{A}\theta)$ is *true*

Thus $I(\mathcal{B} \rightarrow \mathcal{A}\theta)$ is *true*

Theorem: Soundness of SLD resolution

Given:

Program P , goal $\leftarrow \mathcal{A} = \leftarrow A_1, \dots, A_n$

There exists a SLD-refutation of $P \cup \{\leftarrow \mathcal{A}\}$
with substitutions $\theta_0, \dots, \theta_{m-1}$ ($\theta_0 \dots \theta_{m-1}$ is
cas)

then:

$P \models (A_1 \wedge \dots \wedge A_n)\theta_0 \dots \theta_{m-1}$

Proof:

refutation: $\leftarrow \mathcal{A}_0 = \leftarrow \mathcal{A}$,

$\leftarrow \mathcal{A}_1, \dots, \leftarrow \mathcal{A}_m = \square = \leftarrow true$

apply lemma: $P \models \mathcal{A}_1 \rightarrow \mathcal{A}_0\theta_0$

$P \models \mathcal{A}_2 \rightarrow \mathcal{A}_1\theta_1$

modus ponens: $P \models \mathcal{A}_2 \rightarrow \mathcal{A}_0\theta_0\theta_1$

Repeated lemma application gives:

$P \models \mathcal{A}_m \rightarrow \mathcal{A}_0\theta_0\theta_1 \dots \theta_{m-1}$ or

$P \models true \rightarrow \mathcal{A}_0\theta_0\theta_1 \dots \theta_{m-1}$ or

$P \models \mathcal{A}_0\theta_0\theta_1 \dots \theta_{m-1}$ or

$P \models (A_1 \wedge \dots \wedge A_n)\theta_0\theta_1 \dots \theta_{m-1}$

Corollary:

If there exists a SLD-refutation of $P \cup \{\leftarrow \mathcal{A}\}$
then $P \cup \{\leftarrow \mathcal{A}\}$ is inconsistent.

Proof:

according to Theorem:

$$P \models \mathcal{A}\theta_0\theta_1 \dots \theta_{m-1}$$

$$\text{Thus } P \models \exists x_1, \dots, x_k \mathcal{A}$$

Thus $P \cup \{\neg \exists x_1, \dots, x_k \mathcal{A}\}$ is inconsistent

Thus $P \cup \{\forall x_1, \dots, x_k \neg \mathcal{A}\}$ is inconsistent

Thus $P \cup \{\forall x_1, \dots, x_k (\text{false} \leftarrow \mathcal{A})\}$ is inconsis-
tent

Thus $P \cup \{\forall x_1, \dots, x_k (\leftarrow \mathcal{A})\}$ is inconsistent

Thus $P \cup \{\leftarrow \mathcal{A}\}$ is inconsistent

what about *completeness*? If $P \models \mathcal{A}\theta$, does there exist a refutation? But first:

7.2 Herbrand Models

The *Herbrand Universe* U_L of a language L (with at least one constant) is the set of all ground terms

e.g. constants a, b functor $f/1$

$$U = \{a, b, f(a), f(b), f(f(a)), \dots\}$$

Herbrand Interpretation

An interpretation with a fixed pre-interpretation:

Domain: Herbrand Universe U_L

preinterpretation constants: $J(c) = c$

preinterpretation function: $J(f(t_1, \dots, t_n)) = f(J(t_1), \dots, J(t_n))$

Variable assignment: mapping from variables to the (ground) terms of U_L

Term assignment: maps term to ground term of U_L

Hence isomorphism between terms and objects in the world

Interpretation of predicates: $I(p/n)$ is a set of ground tuples in U_L^n

$p(t_1, \dots, t_n)$ is true under a variable assignment V if $\langle V(t_1), \dots, V(t_n) \rangle$ belongs to $I(p/n)$

Thus $I(p/n)$ is a set of ground atoms.

Herbrand Base B_L is the set of all ground atoms in the language L .

So, a Herbrand interpretation is identified by a subset of the Herbrand base.

Herbrand model: a Herbrand interpretation which is a model.

e.g. $p(x) \leftarrow p(a) \leftarrow$

$B_L = \{p(a)\}$, \emptyset is a Herbrand interpretation which is not a model, $\{p(a)\}$ is one which is a model (and there are no others)

$I \models F$ iff I is a model of F for all variable assignments

$I \models F$ iff I is a model of $F\theta$ for all ground substitutions θ

Lemma

Given S is a set of universal formulas (thus of the form $\forall xF$ with F quantifier free; includes clauses)
If S has a model, it has a Herbrand model.

Proof idea:

- Given I construct I_H as follows:

$$I_H = \{p(t_1, \dots, t_n) \mid \text{ground}(p(t_1, \dots, t_n)) \wedge p_I(I(t_1), \dots, I(t_n))\}.$$

- For every formula F , one can show, by induction over the structure:

$$\text{If } I \models F \text{ then } I_H \models F$$

Exercise: Let $D = N$. $J(a) = 0$, $J(f(n)) = n + 1$ and
 $p_I = \{ \langle n \rangle \mid n \text{ is even} \}$. Let F be $p(X) \rightarrow p(f(f(X)))$.

Show that $I \models F$

Construct I_H

Show that $I_H \models F$

for proving validity/inconsistency of set of clauses (logic program + goal), it *suffices* to check *Herbrand interpretations*

Not for all formulas! e.g.

$p(a)$

$\exists x : \neg p(x)$

domain $D = \{1, 2\}$

$I(a) = 1, \quad I(p/1) = \{\langle 1 \rangle\}$

$I(p(a)) = true, \quad I(\exists x \neg p(x)) = true$ (for $x = 2$)

Thus I is model.

Two Herbrand interpretations possible:

$I_H = \emptyset : I_H(p(a))$ is false

$I_H = \{p(a)\} : I_H(\exists x \neg p(x))$ is false (only 1 domain element)

There exists a consistency preserving transformation to clausal form:

$p(a)$

$\neg p(sk)$

with sk a skolem constant

Typically, a program has several Herbrand models

e.g. $p(a) \leftarrow q(b) \leftarrow$

$U_L = \{a, b\}, B_P = \{p(a), p(b), q(a), q(b)\}$

The models are: $\{p(a), q(b)\}, \{p(a), p(b), q(b)\},$
 $\{p(a), q(a), q(b)\}, \{p(a), p(b), q(a), q(b)\} = B_P$

B_P is model of every ground clause $A \leftarrow B_1, \dots, B_k$ so it is model of every program.

Model intersection property: Intersection of Herbrand models is Herbrand model

consider ground clause: $A \leftarrow B_1, \dots, B_k$ if $\{B_1, \dots, B_k\} \subseteq I_1 (\subseteq I_2)$

then $A \in I_1 (\in I_2)$. Thus if $\{B_1, \dots, B_k\} \subseteq I_1 \cap I_2$ then $A \in I_1 \cap I_2$

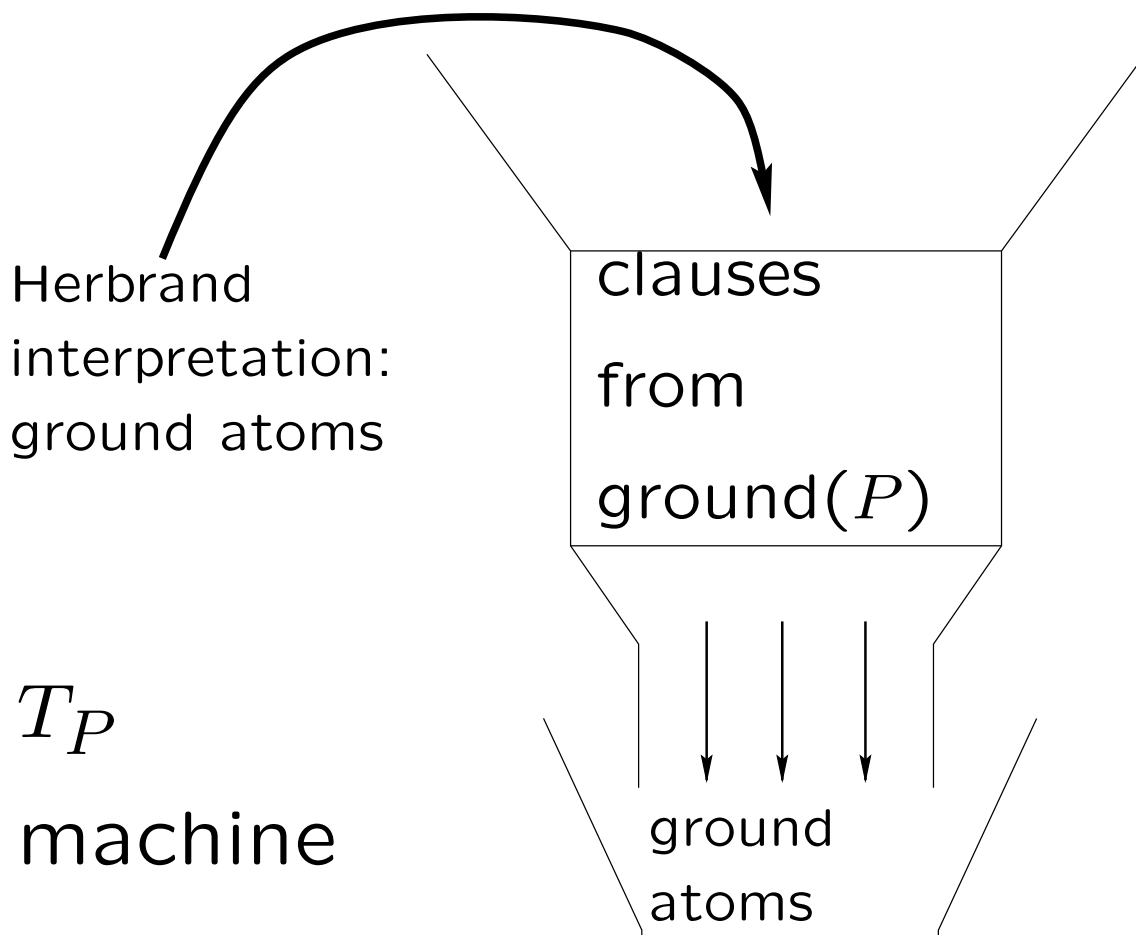
Thus there exists a least model, subset of all others

What is true in all models (is *valid*) = what is true in least model

7.3 Fixpoint semantics

how to construct least model?

Immediate consequence operator T_P



$A \in T_P(I)$ iff $A \leftarrow B_1, \dots, B_n$ is a ground instance of a clause in P and $\{B_1, \dots, B_n\} \subseteq I$

Example:

$$q(a) \leftarrow p(b) \leftarrow p(f(x)) \leftarrow p(x)$$

$$T_P(\emptyset) = \{q(a), p(b)\}$$

$$T_P(\{q(a)\}) = \{q(a), p(b)\}$$

$$T_P(\{p(a)\}) = \{q(a), p(b), p(f(a))\}$$

$$T_P(\{q(a), p(b)\}) = \{q(a), p(b), p(f(b))\}$$

$$T_P(\{q(a), p(b), p(f(b))\}) = \\ \{q(a), p(b), p(f(b)), p(f(f(b)))\}$$

$$T_P(B_P) = \{q(a), p(b), p(f(a)), p(f(b)), p(f(f(a))), \\ p(f(f(b))), \dots, p(f(f \dots (a) \dots)), p(f(f \dots (b) \dots))\}$$

More input: more output $I \subseteq J$ implies $T(I) \subseteq T(J)$:

T_P is monotonic

for infinite sequence $I_0 \subseteq I_1 \subseteq I_2 \subseteq I_3 \dots$:

$$T_P\left(\bigcup_{n=0}^{\infty} I_n\right) = \bigcup_{n=0}^{\infty} T_P(I_n) \text{ continuous}$$

Lemma I_H is model of P iff $T_P(I_H) \subseteq I_H$ (a *pre-fixpoint* of T_P)

Consider all ground clauses $A \leftarrow Body$

Assume I_H is model: $A \in T_P(I_H)$ if $Body \subseteq I_H$. Since I_H is a model, also $A \in I_H$ hence $T_P(I_H) \subseteq I_H$

Assume $T_P(I_H) \subseteq I_H$: $Body \subseteq I_H$ implies $A \in T_P(I_H)$ and by assumption $A \in I_H$ hence I_H is model of clause

The set of all subsets of B_P is (as every powerset) a complete lattice every set of subsets has lub and glb under the partial order \subseteq

Least upperbound: $\text{lub}(I, J) = I \cup J$

greatest lower bound: $\text{glb}(I, J) = I \cap J$

So let have a look at properties of monotonic continuous operators over complete lattices

continuity revisited

upward continuous: consider $x_0 \leq x_1 \leq x_2 \dots$

$T(\text{lub}(x_0, x_1, \dots)) = \text{lub}(T(x_0), T(x_1), \dots)$

e.g. reals, a discontinuous function:

$T(x) = \text{if } x < 2 \text{ then } 10 \text{ else } 20$

$T(\text{lub}(1.9, 1.99, 1.999, \dots)) = T(2) = 20$

$\text{lub}(T(1.9), T(1.99), T(1.999), \dots) =$

$\text{lub}(10, 10, 10, \dots) = 10$

downward continuous: consider $x_0 \geq x_1 \geq x_2 \dots$

$T(\text{glb}(x_0, x_1, \dots)) = \text{glb}(T(x_0), T(x_1), \dots)$

e.g. reals, a discontinuous function:

$T(x) = \text{if } x > 2 \text{ then } 20 \text{ else } 10$

$T(\text{glb}(2.1, 2.01, 2.001, \dots)) = T(2) = 10$

$\text{glb}(T(2.1), T(2.01), T(2.001), \dots) =$

$\text{glb}(20, 20, 20, \dots) = 20$

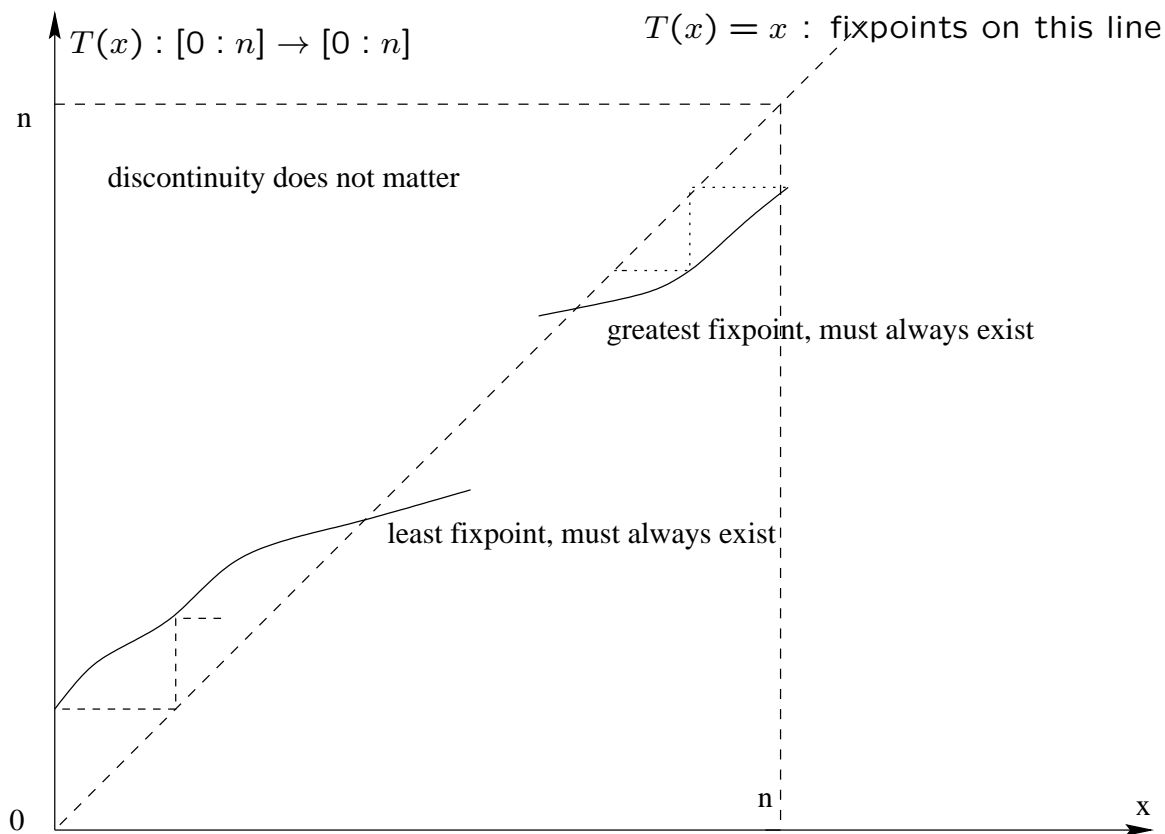
I fixpoint of T : $T(I) = I$

I pre-fixpoint of T : $T(I) \subseteq I$

I post-fixpoint of T : $T(I) \supseteq I$

for T_P : \emptyset is a post-fixpoint, B_P is pre-fixpoint

Fixpoint theorem from Knaster and Tarski:
A monotonic operator T over a complete lattice has a least fixpoint $lfp(T)$ which is also the least pre-fixpoint (least: smaller than all others) and a greatest fixpoint $gfp(T)$ which is also the greatest post-fixpoint (greatest: larger than all others)



Powers of monotonic operators

$$T \uparrow 0(I) = I$$

$$T \uparrow (n + 1)(I) = T(T \uparrow n(I))$$

$$T \uparrow \omega(I) = \bigcup_{n < \omega} T \uparrow n(I)$$

$$T \downarrow 0(I) = I$$

$$T \downarrow (n + 1)(I) = T(T \downarrow n(I))$$

$$T \downarrow \omega(I) = \bigcap_{n < \omega} T \downarrow n(I)$$

ω : first infinite ordinal: limit of $0, 1, 2, \dots$

similar definitions for other “infinite ordinals”

abbreviations $T \uparrow n(\emptyset) = T \uparrow n$ and $T \downarrow n(B_P) = T \downarrow n$

where \emptyset : the least element of the lattice, B_P the greatest element of the lattice

Lemma (Kleene) If T is upward continuous then $T \uparrow \omega$ is least fixpoint and least pre-fixpoint of T .

If T is downward continuous then $T \downarrow \omega$ is greatest fixpoint and greatest post-fixpoint.

T_P is not downward continuous e.g. try for :
 $p(f(x)) \leftarrow p(x) \quad q(a) \leftarrow p(x)$

all pre-fixpoints are models, there is a least pre-fixpoint (which is the lfp), so:

Characterisation Theorem (Van Emden-Kowalski JACM76)

A definite program P has a Herbrand model M_P which satisfies:

- M_P is the least Herbrand model of P
- M_P is the least pre-fixpoint of T_P
- M_P is the least fixpoint of T_P
- $M_P = T_P \uparrow \omega$

7.4 Completeness of SLD

Success set: The success set of P is the set of all ground atoms A such that $P \cup \{\leftarrow A\}$ has an SLD-refutation

Corollary: The success set is contained in M_P

SLD is sound so $P \models A$, so every model of P is model of A , so also M_P is, and $I_H(A) = A$ so $A \in M_P$

Herbrand models, T_P operator

For each of the following programs (in the language underlying the program):

- i. Give the Herbrand base B_P .
- ii. Give all Herbrand models.
- iii. Give the least Herbrand model ($T_P \uparrow \omega$).
- iv. Give $T_P \downarrow \omega$ (is it a fix-point?).
- v. Is T_P downward continuous?

1. $p(X) \leftarrow q(X).$
 $q(a) \leftarrow.$
 $r(X) \leftarrow s(X).$

2. $p(X) \leftarrow p(X).$
 $r(X) \leftarrow s(X).$
 $s(a) \leftarrow.$

3. $p(f(X)) \leftarrow p(X).$
 $q(a) \leftarrow p(X).$

4. $p(Y) \leftarrow p(X), q(X, Y).$
 $p(a) \leftarrow.$
 $q(a, b) \leftarrow.$
 $q(c, d) \leftarrow.$

Lifting Lemma:

If there exists a refutation for $P \cup \{\leftarrow A\theta\}$,
then also for $P \cup \{\leftarrow A\}$

intuition: if $A\theta$ unifies with a head, so does A and the resolvent
is more general

completeness results for SLD

Theorem: the success set of a program is
equal to M_P

sufficient to show that M_P is contained in
success set

intuition: T_P builds a derivation of ground atom $A\theta$
(with ground clauses and ground goals) bottom up.
This derivation can be reorganised to a refutation of
 $\leftarrow A\theta$ in $ground(P)$, and then lifted to a refutation of
 $\leftarrow A\theta$ in P

e.g. $A \leftarrow B_1, B_2$ $C_1 \leftarrow$
 $B_1 \leftarrow C_1$ $D_1 \leftarrow$
 $B_2 \leftarrow D_1, D_2$ $D_2 \leftarrow$

$$C_1\theta \in T_P \uparrow 1 \quad \leftarrow C_1\theta. \quad C_1\theta \leftarrow \in gr(P)$$

□

$$D_1\theta \in T_P \uparrow 1 \quad \leftarrow D_1\theta. \quad D_1\theta \leftarrow \in gr(P)$$

□

$$D_2\theta \in T_P \uparrow 1 \quad \leftarrow D_2\theta. \quad D_2\theta \leftarrow \in gr(P)$$

□

$$B_1\theta \in T_P \uparrow 2 \quad \leftarrow B_1\theta. \quad B_1\theta \leftarrow C_1\theta \in gr(P)$$

$$\leftarrow C_1\theta. \quad C_1\theta \leftarrow \in gr(P).$$

□.

$$B_2\theta \in T_P \uparrow 2 \quad \leftarrow B_2\theta. \quad B_2\theta \leftarrow D_1\theta, D_2\theta \in gr(P)$$

$$\leftarrow D_1\theta, D_2\theta. \quad D_1\theta \leftarrow \in gr(P)$$

$$\leftarrow D_2\theta. \quad D_2\theta \leftarrow \in gr(P)$$

□

$$A\theta \in T_P \uparrow 3 \quad \leftarrow A\theta. \quad A\theta \leftarrow B_1\theta, B_2\theta \in gr(P)$$

$$\leftarrow B_1\theta, B_2\theta. \quad B_1\theta \leftarrow C_1\theta \in gr(P)$$

$$\leftarrow C_1\theta, B_2\theta. \quad C_1\theta \leftarrow \in gr(P)$$

$$\leftarrow B_2\theta. \quad B_2\theta \leftarrow D_1\theta, D_2\theta \in gr(P)$$

$$\leftarrow D_1\theta, D_2\theta. \quad D_1\theta \leftarrow \in gr(P)$$

$$\leftarrow D_2\theta. \quad D_2\theta \leftarrow \in gr(P)$$

□

Theorem: If $P \cup \{\leftarrow \mathcal{A}\}$ is inconsistent then a SLD-refutation starting from $\leftarrow \mathcal{A}$ exists

Proof:

- Inconsistent: $false \leftarrow \mathcal{A}$ is false in the minimal model M_P
- There exists variable assignment, hence a ground θ such that $\leftarrow \mathcal{A}\theta$ is false in M_P
- Thus $\mathcal{A}\theta = (\dots, A_i, \dots)\theta$ true in M_P
- $A_i\theta \in M_P$ is ground, so a refutation of A_i exists
- plug together and lift: refutation for $\leftarrow \mathcal{A}$

Theorem: If $P \models \mathcal{A}\tau$ then there exists a SLD-refutation of $\leftarrow \mathcal{A}$ with computed answer σ such that $\mathcal{A}\sigma$ more general than $\mathcal{A}\tau$
 NOT: σ more general than τ (error in Lloyd)

e.g $p(f(x)) \leftarrow P \models p(f(a)) = p(y)\{y/f(a)\}$
 goal $\leftarrow p(y)$ gives cas $\{y/f(x)\}$

Intuition: SLD-refutation of $\leftarrow \mathcal{A}\tau$ exists (cas θ), can be lifted to SLD-refutation of $\leftarrow \mathcal{A}$, with cas σ such that $\mathcal{A}\sigma$ more general than $\mathcal{A}\tau\theta$

Theorem Independence of computation rule
If $P \cup \{\leftarrow \mathcal{A}\}$ is inconsistent, then in *every* SLD-tree for $P \cup \{\leftarrow \mathcal{A}\}$, there exists a refutation of $P \cup \{\leftarrow \mathcal{A}\}$

Selection of the literal in a resolution step does not matter.

BUT one selection can cause infinite branches, another can keep tree finite (and PROLOG can run away in infinite branch)

intuition: based on a *switching lemma*: if steps are reordered, (e.g $\leftarrow A, B$ first selection of A next of B or other way round) the resolvent obtained after the reordered steps is identical (up to renaming)

in other words, you can completely reshuffle a derivation (of course not selecting a literal before it exists)

Different branching: because selected literal is more or less general, more or less clause heads can unify

Soundness and Completeness of SLD

Given some program P . What can you say about

- i. the inconsistency;
 - ii. the existence of an SLD-refutation
- in each of the following cases:

- a. $P \cup \{\leftarrow p(X)\}$
- b. $P \cup \{\leftarrow p(f(X))\}$
- c. $P \cup \{\leftarrow p(f(a))\}$

when

1. $P \cup \{\leftarrow p(f(X))\}$ is inconsistent.
2. $P \cup \{\leftarrow p(f(X))\}$ has an SLD-refutation.
3. $P \models p(f(b))$.

a few words on

8 Constraint Logic Programming

Represent a goal N_i as :

$\leftarrow A_1, \dots, A_m; E_i$

E_i : set of equations, initially empty

Resolution step

- Select A_j

- Select a clause $A \leftarrow B_1, \dots, B_k$

where $pred(A) = pred(A_j)$

- If $E_{i+1} = E_i \cup \{A = A_j\}$ is solvable

then derive resolvent

$A_1, \dots, A_{j-1}, B_1, \dots, B_k, A_{j+1}, A_m; E_{i+1}$

With equation solving as unification over terms:
standard logic programming (with pre-interpretation
fixed to be the Herbrand interpretation!)

natural for “programming” not for “logic”

With some symbols predefined and fixed in-
terpretation:

Constraint Logic Programming

E.g terms “1+3” and “4” are different (many interpretations where they denote different objects)

LP uses a relation plus (with “infinite” set of tuples): $plus(1, 3, x)$ yields $x = 4$

CLP(Q): fixes the pre-interpretation of numbers and of + as in arithmetic of the rational numbers

powerful solvers:

In CLP $\leftarrow x + y = 5, x - y = 3$ will give solution without search.

Closest in LP:

$enumerate(0) \leftarrow$

$enumerate(1) \leftarrow$

⋮

$enumerate(100) \leftarrow$

$\leftarrow enumerate(x), plus(x, y, 5), plus(y, 3, x)$

Careful reordering, still large search space, need to limit range of x

or write symbolic solver which collect equations and reduces them is solving at meta level

Some examples:

$length([], 0) \leftarrow$
 $length([X | L], N) \leftarrow length(L, M),$
 $N > 0, N = M + 1$

possible queries:

$\leftarrow length([a, b, c], N)$
 $\leftarrow length(L, 5)$

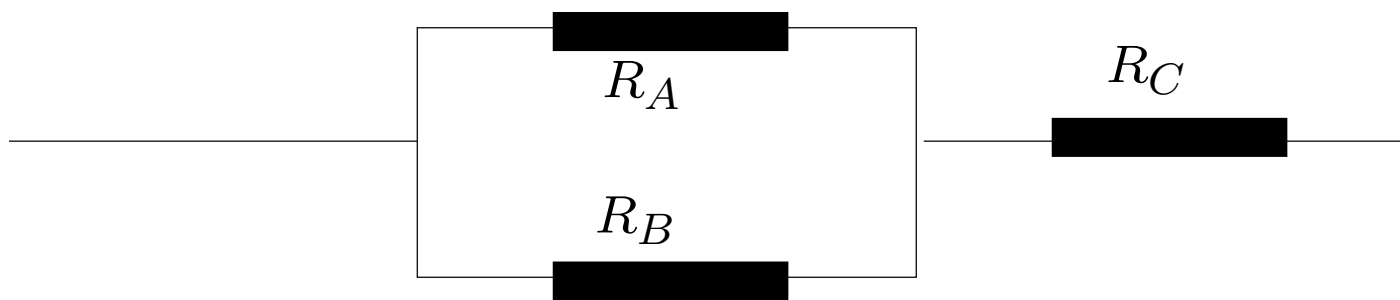
Prolog: has to write program in function of one query:

either ... $length(L, M), N$ is $M + 1$
or ... $N > 0, M$ is $N - 1, length(L, M)$

$loan([], Interest, 0) \leftarrow$
 $loan([Instalment | L], Interest, Capital) \leftarrow$
 $loan(L, Interest, C),$
 $C = Capital * (100 + Interest) / 100 - Instalment$

possible queries:

$\leftarrow loan([I, I, I], 10, 10000)$
 $\leftarrow loan([1000, 1500, 2000], 9, Capital)$
 $\leftarrow loan([I_1, I_2, I_2], 10, 10000), I_2 = 2I_1$
but typical solver not powerful enough for:
 $\leftarrow loan([1000, 1000, 1000], Interest, 5000)$



$scheme(R_A, R_B, R_C, R) \leftarrow$

$$R = R_{AB} + R_C, 1/R_{AB} = 1/R_A + 1/R_B)$$

possible queries:

$\leftarrow scheme(10, 5, 3, R)$

$\leftarrow scheme(10, R_B, 3, 8)$

$\leftarrow scheme(10, 5, R_C, 13)$