

Normal programs

In which we discover that viewing logic programs as (a subset of) FOL is too weak.
In searching a better semantics, we finally arrive at inductive definition logic

1. Summary about definite programs

- Declarative semantics of logic:
logical consequence:
 $P \models F$ iff F true in all models of P

with P, F clauses: iff true in all H-models

with P definite program, F definite goal:
iff true in minimal H-model.
- procedural semantics of logic programs:
SLD-derivation and -refutation
symbol \vdash e.g $P \cup \{\leftarrow \mathcal{A}\} \vdash \square$
success set: refutable ground atoms

- fixpoint semantics least fixpoint:

$$T_P \uparrow \omega$$

- $\models \equiv \vdash$

$$\text{minimal H-model} = \text{success set} = T_P \uparrow \omega$$

- Soundness of SLD

if $\leftarrow A_1, \dots, A_n$ refuted with cas σ

then $P \models (A_1, \dots, A_n)\sigma$

- Completeness of SLD

if $P \models (A_1, \dots, A_n)\theta$ then

$\leftarrow A_1, \dots, A_n$ has a refutation with cas σ

such that $(A_1, \dots, A_n)\sigma \geq (A_1, \dots, A_n)\theta$

NOT $\sigma \geq \theta$

- Independance of computation rule

2. Negative information from definite programs

2.1. Motivation

Example

$male(john) \leftarrow$
 $female(mary) \leftarrow$

prove $\neg female(john)$ or $\leftarrow female(john)$?

add negation to theory:

$male(john) \leftarrow$
 $female(mary) \leftarrow$
 $female(john) \leftarrow$

NO refutation possible

Indeed $female(john)$ is true in B_P , the largest model

needs bigger theory (which is not definite) including:

$\leftarrow female(john)$
 $\leftarrow male(mary)$

$female(john) \leftarrow, \leftarrow female(john) \vdash \square$

Can be encoded as definite program

```
male(john) ←  
female(mary) ←  
not_female(john) ←  
not_male(mary) ←  
← not_female(john)
```

BUT

inconvenient

explosion of size of

data base/knowledge base/program

PROLOG programmers use “trick” using “cut”

if $p(\dots)$ finitely fails

then $not(p(\dots))$ holds

If one cannot prove $p(\dots)$ then $not(p(\dots))$ is

“proved” (assumed)

$not(x) : -x, !, fail.$

$not(x).$

$\dots, not(p(\dots)), \dots$

Why and When does it work?

Negation As Finite Failure (NAF rule)

2.2. Non-monotonic reasoning

Let P be consistent (everything follows from inconsistent theory)

Monotonic reasoning:

if $P \vdash \varphi$ then $P \cup P' \vdash \varphi$

SLD is monotonic: a refutation remains valid after extending a program

Sound reasoning: if $P \vdash \varphi$ then $P \models \varphi$

Weakly sound: if $P \vdash \varphi$ then $P \cup \varphi$ is consistent (has a model)

i.e. if φ derived, then it is possible

Definite program: always consistent (B_P is a model)

Lemma:

There is no weakly sound method \rightsquigarrow which allows to derive negative information from definite programs and which is monotonic

Proof

Suppose otherwise:

$P \rightsquigarrow \leftarrow A (= \neg A)$

$P \cup \{A \leftarrow\} \rightsquigarrow \leftarrow A$ (monotonicity)

$P \cup \{A \leftarrow\} \cup \{\leftarrow A\}$ is consistent (weakly sound)
contradiction

NAF is non-monotonic

$p \leftarrow q$

$p \leftarrow q$

$q \leftarrow$

$\leftarrow p$ finitely fails

$\leftarrow p$ succeeds

$\leftarrow \text{not}(p)$ succeeds

$\leftarrow \text{not}(p)$ fails

2.3. Closed World Assumption (Reiter 78)

$$\frac{\text{not}(P \vdash A)}{\neg A}$$

derive $\neg A$

when A is not provable in first order logic

because P is definite and SLD complete

when A is not provable by SLD.

$CWA(P) =$

$\{A : A \text{ is ground, no refutation of } \leftarrow A\}$

So $CWA(P) = \{p | p \in B_P \setminus M_P\}$

complement of success set in Herbrand base

$P \cup \{\neg A \mid A \in CWA(P)\}$ is always consistent
least Herbrand model of P is a model so CWA is weakly
sound

$CWA(P)$ is not always recursively enumerable
as one cannot always enumerate all non
theorems (FOL is semi-decidable)

2.4. Negation as finite failure and completion semantics (Clark78)

CWA: not what PROLOG/SLD does
infinite SLD-trees do not fail

NAF: a more restricted form of:
not able to proof

finite failure set (FFS): all ground atoms A
such that there exists a finitely failed SLD-
tree for $\leftarrow A$

$$\frac{A \in FFS(P)}{\neg A}$$

$FFS(P) \subseteq CWA(P)$ so $P \cup \{\neg A \mid A \in FFS(P)\}$
is always consistent hence NAF is weakly sound

there *exists* a finitely failed tree: computable?

fair SLD-derivation

Consider the atoms

Q

$Q\theta_i$

$Q\theta_i\theta_{i+1}$

\vdots

in successive goals as the *same* atom.

A SLD-derivation is fair if

it is finite

or if every atom which is introduced in a goal is eventually selected in one of the following goals.

example unfair derivation

$$p(0) \leftarrow$$

$$p(s(x)) \leftarrow p(x)$$

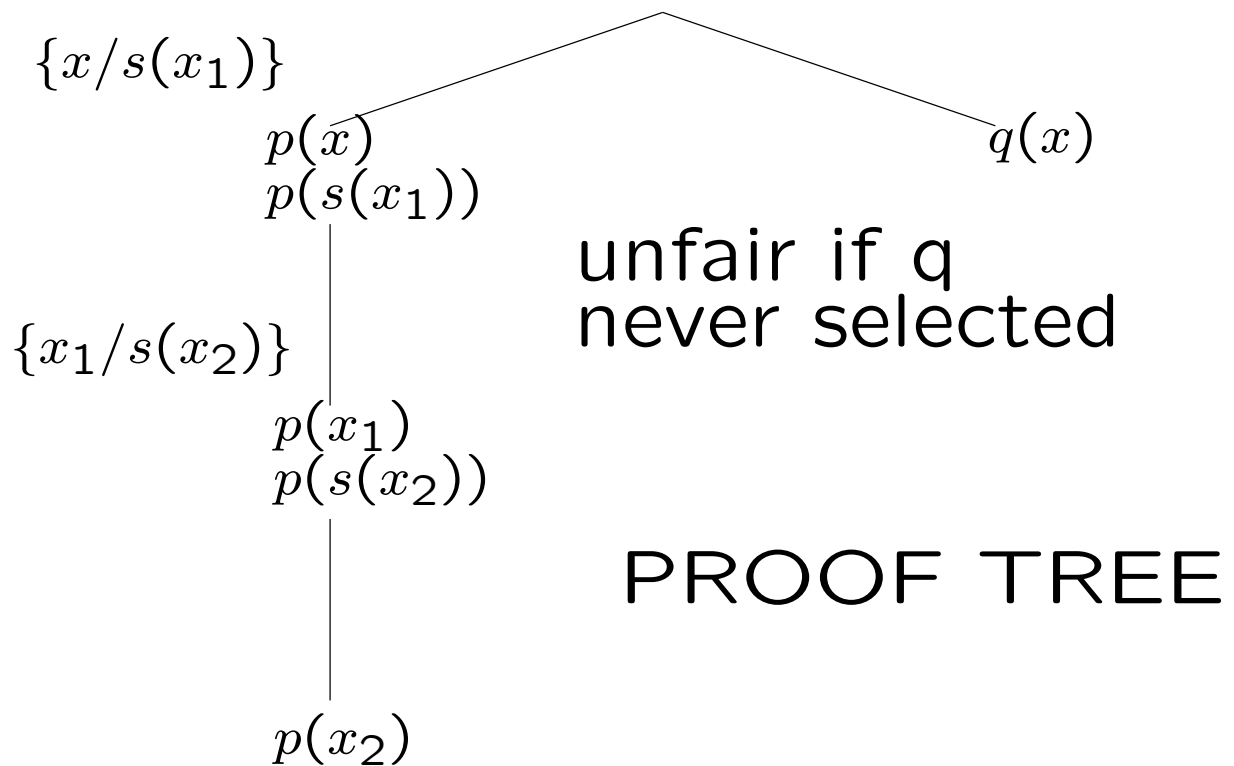
$$\leftarrow p(x), q(x)$$

$$\leftarrow p(x_1), q(s(x_1))$$

$$\leftarrow p(x_2), q(s(s(x_1)))$$

$$\vdots$$

If $q/1$ never selected, then *unfair* derivation



fair SLD-tree: every branch is a *fair* SLD-derivation

Theorem: equivalent:

- a. A is in finite failure set of P
- b. $A \notin T_P \downarrow \omega$
- c. every fair SLD-tree with $\leftarrow A$ as root is finitely failed

Proof sketch:

c. implies a.

If every *fair* SLD-tree $\leftarrow A$ finitely fails, then there definitely exists a finitely failing tree, so by definition, A is in $FFS(P)$ (the finite failure set)

Other steps: Correspondence between

$A \in T_P \downarrow k$ and derivation of $\leftarrow A$ where after finite number of steps every remaining atom succeeds.

a. $(A \in FFS(P))$ implies b. $(A \notin T_P \downarrow \omega)$

we prove the stronger claim:

A has finitely failed tree of depth k implies
 $A \notin T_P \downarrow k$

for $k=1$: A is a failing literal, hence even with whole Herbrand base, no rule can derive A
i.e. $A \notin T_P \downarrow 1$

Assume it holds for $k - 1$

- After at most k steps, every derivation ends with selection of a failing literal
- For any ground instance of any derivation: it starts with a ground clause $A \leftarrow B_1, \dots, B_k$, hence $\leftarrow B_1, \dots, B_k$ fails after at most $k - 1$ steps.
- That derivation can be splitted in (partial) derivations for each atom
 $\exists i$: the derivation of atom B_i selects a failing literal after at most $k - 1$ steps i.e. $B_i \notin T_P \downarrow k - 1$ (induction hypothesis)
- Hence that derivation cannot show that $A \in T_P \downarrow k$, neither can any other ground instance of any other derivation.

b. $(A \notin T_P \downarrow \omega)$ implies c. (every fair tree fails finitely)

So $\exists k : A \notin T_P \downarrow k$

- If there is a fair non-failing derivation, then it can be instantiated to a fair non-failing ground derivation. So sufficient to show there is no fair non-failing ground derivation.
- $k = 1$: $A \notin T_P \downarrow 1$ hence $\leftarrow A$ fails in first step.
- Induction step: $A \notin T_P \downarrow k$
- For every ground clause $A \leftarrow B_1, \dots, B_k$ there exists a B_i such that $B_i \notin T_P \downarrow k - 1$
- Hence (induction hypothesis) every fair tree for $\leftarrow B_i$ fails finitely.
- Hence fair derivation of $\leftarrow B_1, \dots, B_k$ fails finitely as selection of atoms leading to failure in derivation of B_i cannot be delayed for ever.
- Hence also every fair derivation of $\leftarrow A$ fails finitely

a. $\rightarrow b. \rightarrow c. \rightarrow a.$ so all are equivalent

Exercises

1. Consider the program P :

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

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

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

$r(a) \leftarrow.$

Draw an infinite and a finite SLD-tree for the query $\leftarrow p(a)$. Are they fair?

Is $\neg p(a)$ provable by CWA? By NAF?

2. Consider the following program P :

$\text{parent}(a,b) \leftarrow.$

$\text{parent}(b,c) \leftarrow.$

$\text{anc}(X,Y) \leftarrow \text{parent}(X,Y).$

$\text{anc}(X,Y) \leftarrow \text{anc}(X,Z), \text{anc}(Z,Y).$

a. Construct for the query $\leftarrow \text{anc}(a,c)$: (i) a finite SLD-refutation, (ii) an unfair infinite SLD-derivation, (iii) a fair infinite SLD-derivation.

Is $\neg \text{anc}(a,c)$ provable by CWA? By NAF?

b. Construct for the query $\leftarrow \text{anc}(c,a)$: (i) a failed SLD-derivation, (ii) a fair SLD-tree, (iii) an unfair SLD-tree.

Is $\neg \text{anc}(c,a)$ provable by CWA? By NAF?

That was the procedural side, there is also a logical side

Completion semantics

A solution within framework of FOL

What you write (a definite program, implications)

is not what you mean (a more complex formula in FOL)

small example:

$$\begin{aligned} male(john) &\leftarrow \\ male(peter) &\leftarrow \end{aligned}$$

is abbreviation for

$$\forall x(male(x) \leftrightarrow x = john \vee x = peter)$$

In this formula, = stands for *identity* (see below)

construction of completion

1. Remove terms from heads and make conjunction explicit:

replace $p(t_1, \dots, t_n) \leftarrow B_1, \dots, B_n$ by

$p(x_1, \dots, x_n) \leftarrow (x_1 = t_1) \wedge \dots \wedge (x_n = t_n) \wedge B_1 \wedge \dots \wedge B_n$ with x_1, \dots, x_n new variables

2. Introduce existential quantifiers:

replace $p(x_1, \dots, x_n) \leftarrow F$ with y_1, \dots, y_m the variables not occurring in the head (i.e. the original clause variables) by

$p(x_1, \dots, x_n) \leftarrow \exists y_1 \dots \exists y_m F$

3. Group similar formulas:

replace

$$\begin{aligned} p(x_1, \dots, x_n) &\leftarrow F_1 \\ &\vdots \\ p(x_1, \dots, x_n) &\leftarrow F_k \end{aligned}$$

by

$$p(x_1, \dots, x_n) \leftarrow F_1 \vee \dots \vee F_k$$

4. Handle undefined predicates:

for each q/n used in a rhs but not in a head
add:

$$q(x_1, \dots, x_n) \leftarrow false$$

is valid formula under every interpretation

5. Introduce universal quantifiers:

replace $p(x_1, \dots, x_n) \leftarrow F$ by

$$\forall x_1 \dots \forall x_n (p(x_1, \dots, x_n) \leftarrow F)$$

so far we have not changed the meaning, the
next step does

the form after step 5 is abbreviated as $IF(P)$

6. Introduce the intended meaning:

replace $\forall x_1 \dots \forall x_n (p(x_1, \dots, x_n) \leftarrow F)$ by

$$\forall x_1 \dots \forall x_n (p(x_1, \dots, x_n) \leftrightarrow F)$$

This form is abbreviated as $IFF(P)$

ONLY-IF(P) is used as abbreviation of

$$\forall x_1 \dots \forall x_n (p(x_1, \dots, x_n) \rightarrow F)$$

example:

$male(john) \leftarrow$
 $male(peter) \leftarrow$
 $female(mary) \leftarrow$

completion:

$\forall x \, male(x) \leftrightarrow x = john \vee x = peter$
 $\forall x \, female(x) \leftrightarrow x = mary$

$\neg female(john)$ is true in every model of $IFF(P)$
provided that

$john = john, peter = peter, mary = mary$
are interpreted as *true* and

$john = mary, mary = john, peter = mary,$
 $mary = peter, peter = john, john = peter$
are interpreted as *false*

alternatively, one can transform $IFF(P)$ in clausal form and make a resolution proof

in general: Skolem functors introduced: $\forall x \dots \exists y \dots y \dots$ occurrences of y need be replaced by $f(x)$ with f a new function symbol

1. $male(x) \leftarrow x = john$
2. $male(x) \leftarrow x = peter$
2. $x = john, x = peter \leftarrow male(x)$
3. $female(x) \leftarrow x = mary$
4. $x = mary \leftarrow female(x)$
5. $john = john \leftarrow$
6. $peter = peter \leftarrow$
7. $mary = mary \leftarrow$
8. $\leftarrow john = mary$
9. $\leftarrow mary = john$
10. $\leftarrow peter = mary$
11. $\leftarrow mary = peter$
12. $\leftarrow peter = john$
13. $\leftarrow john = peter$

negated query

14. $female(john) \leftarrow$

resolution proof

15. $john = mary \leftarrow (14 + 4)$
16. $\square (15 + 8)$

Free (Clark) equality axioms

1. $\forall (x = x)$ reflexive
2. $\forall (x = y) \rightarrow (y = x)$ symmetry
3. $\forall (x = y) \wedge (y = z) \rightarrow (x = z)$ transitivity
4. for each f/n : equals can be replaced by equals:

$$\forall (x_1 = y_1) \wedge \dots \wedge (x_n = y_n) \rightarrow$$

$$f(x_1, \dots, x_n) = f(y_1, \dots, y_n)$$
5. for each p/n : equals can be replaced by equals:

$$\forall (x_1 = y_1) \wedge \dots \wedge (x_n = y_n) \rightarrow$$

$$p(x_1, \dots, x_n) \leftrightarrow p(y_1, \dots, y_n)$$
6. for each functor: corresponding arguments of equal terms are equal:

$$\forall f(x_1, \dots, x_n) = f(y_1, \dots, y_n) \rightarrow$$

$$(x_1 = y_1) \wedge \dots \wedge (x_n = y_n)$$
7. for each pair of distinct functors:

$$\forall f(x_1, \dots, x_n) = g(y_1, \dots, y_m) \rightarrow \text{false}$$
8. $\forall x = t \rightarrow \text{false}$ if t is not x and $x \in \text{Var}(t)$
 - 1,2 and 3: $=$ is equivalence relation
 - 2 and 3 can be derived from 5 and 1:
 $x = y \wedge x = x \rightarrow = (x, x) \leftrightarrow = (y, x)$ using 1: $x = y \rightarrow = (y, x)$
 $y = x \wedge y = z \rightarrow = (y, y) \leftrightarrow = (x, z)$
 - 1,(2,3),4,5: an equality theory
 - 1,(2,3),4,5,6,7,8: free/Clark's equality theory
 - Lloyd: 1,4,5,6,7,8; Apt 6,7,8 + " $=$ " as identity
i.e. $I(=) = \{\langle x, x \rangle\}$.
 - domain of pre-interpretation isomorphic with Herbrand Universe.
 - is the theory under which Martelli Montanari unification algorithm "solves" equations, is also the theory used by CLP(Herbrand domain)

Exercises

3. Choose a language and construct an interpretation such that axioms 1,2 and 3 are true but neither 4 nor 5 are true.
4. Let $D = \{1,2\}$, Choose a language and construct an interpretation I over domain D such that axioms 6,7,and 8 evaluate to true but not all of 1,2,3,4,5 evaluate to true.

the *completion* of P

$$COMP(P) = IFF(P) + FEQ$$

$COMP(P)$: in general not a clausal form

e.g. $p(x) \leftarrow q(x, y)$

$Comp(P)$ is $\forall x [p(x) \leftrightarrow \exists y q(x, y)]$ or

$\forall x \forall y (p(x) \leftarrow q(x, y)) \wedge \forall x \exists y (q(x, y) \leftarrow p(x))$

So not sufficient to consider Herbrand interpretations only!

Theorem (Apt and Van Emden JACM82)

(i) A Herbrand interpretation I_H is a model of $COMP(P)$ iff $T_P(I_H) = I_H$

remember that terms under Herbrand interpretations satisfy the free equality axioms

(ii) $COMP(P)$ has a Herbrand model

because T_P has fixpoints for definite programs

(iii) ground atom A : $COMP(P) \cup A$ has Herbrand model iff $A \in gfp(T_P)$

because $gfp(T_P)$, the largest fixpoint includes all others, thus it is the largest H-model of $COMP(P)$

Theorem: There exists a program P and a ground atom A such that $COMP(P) \cup \{A\}$ has a model but no Herbrand model

proof: consider

$$p(f(x)) \leftarrow p(x). \quad q(a) \leftarrow p(x).$$

$$gfp(T_P) = \emptyset$$

so $COMP(P) \cup A$ has no Herbrand model

A is false in the only H-model of the completion

There is a non-Herbrand model in which $q(a)$ is true:

Domain = $Z \cup N$ (integers + natural numbers)

$$J(a) = 0_N, J(f(n)) = n + 1, J(f(z)) = z + 1$$

$$I(p/1) = \{\langle z \rangle \mid z \in Z\}, I(q/1) = \{\langle 0_N \rangle\}, I(=):$$

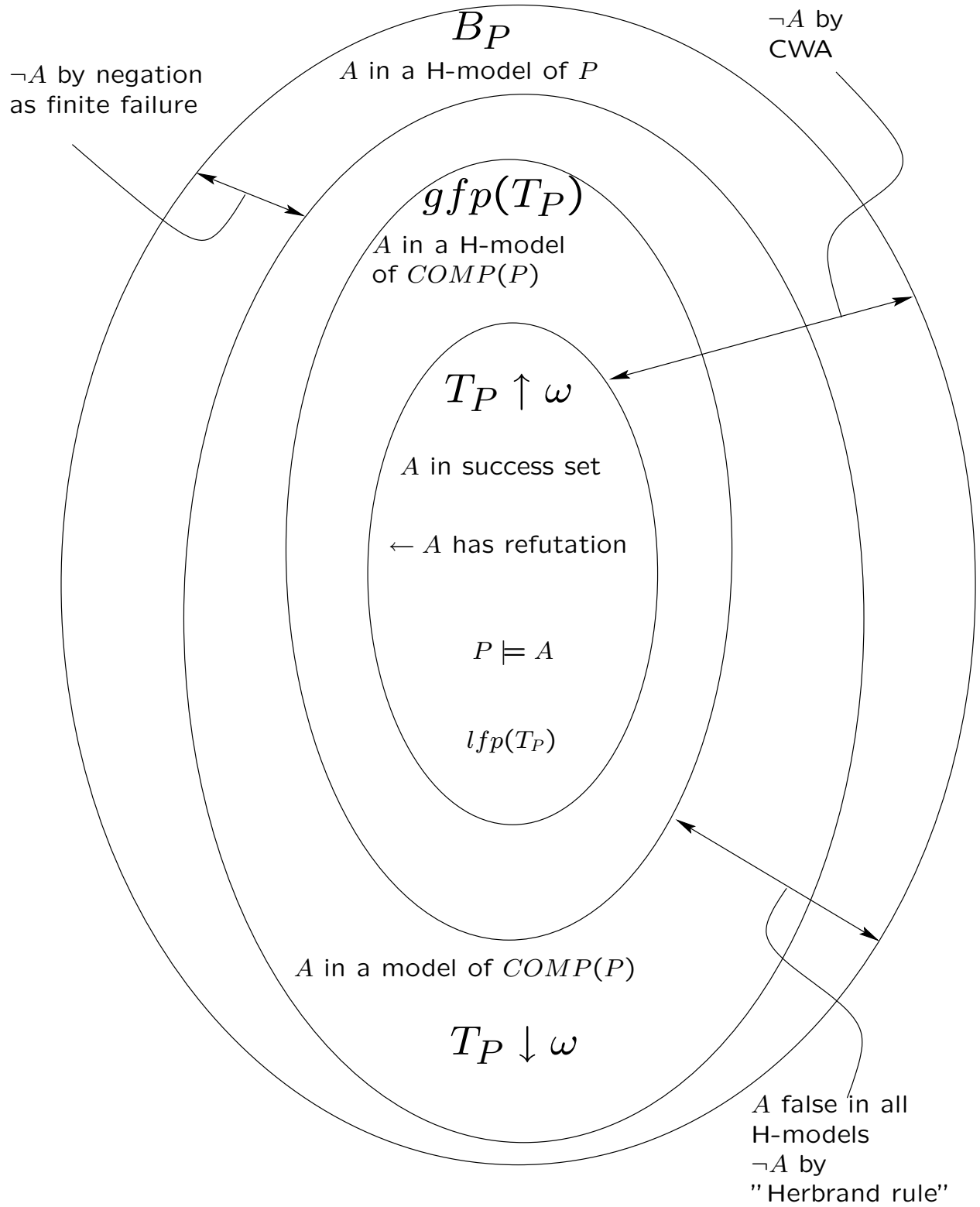
identity

Note: N is isomorphic with Herbrand universe

it is the “Herbrand” component of the interpretation

Exercises

5. Verify that the above interpretation is indeed a model of the completion of the above program.
6. For the same program: Show that with $D = \{0\}$, $J(a) = 0$, $J(f(0)) = 0$, $I(p/1) = I(q/1) = \{\langle 0 \rangle\}$, $I(=)$ the identity relation, I is not a model of the completion.
7. For the above program, is $\neg q(a)$ provable by the closed world assumption? by negation as finite failure? from the completion of the program?



Completion is the “logical” reconstruction of negation as finite failure

Finite Failure Theorem

P a program and A a ground atom

The following are equivalent

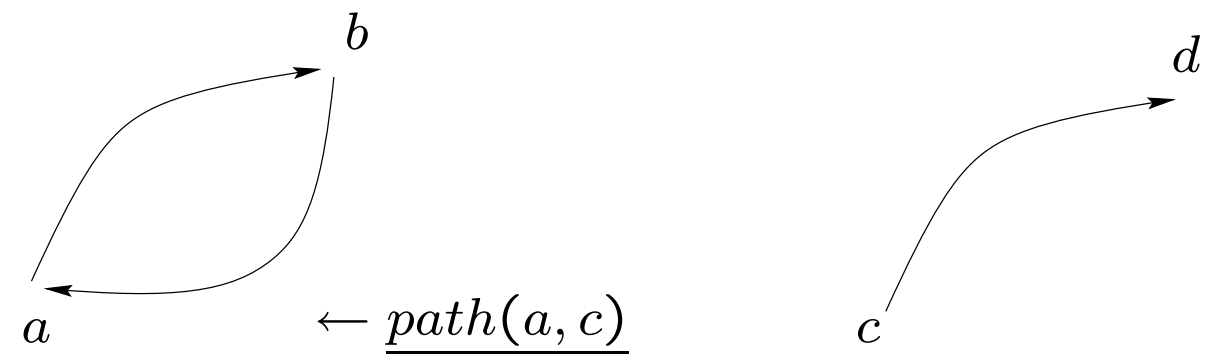
- A is in the finite failure set of P
- $A \notin T_P \downarrow \omega$
- Every fair SLD-tree for $\leftarrow A$ is finitely failed
- $COMP(P) \models \neg A$

For programs where $T_P \uparrow \omega = T_P \downarrow \omega$:

all notions of negation coincide (see figure)

$COMP(P)$ elegant reconstruction of the negative information derivable from definite programs via the SLD proof procedure

2.5 Is completion the intended meaning?



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

$\leftarrow \underline{edge(a, c)}$
failure

$\leftarrow \underline{edge(a, z), path(z, c)}$

$\leftarrow \underline{path(b, c)}$

$\leftarrow \underline{edge(b, c)}$
failure

$\leftarrow \underline{edge(b, z), path(z, c)}$

fair and infinite

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

$path(x, y) \leftarrow edge(x, y)$
 $path(x, y) \leftarrow edge(x, z), path(z, y)$
 $edge(a, b) \leftarrow$
 $edge(b, a) \leftarrow$
 $edge(c, d) \leftarrow$
 $\leftarrow path(a, c)$

$\neg path(a, c)$
cannot be proven
but is in
intended meaning
CWA(P) is intended

Another example

$nat(0) \leftarrow$
 $nat(s(x)) \leftarrow nat(x)$

$even(0) \leftarrow$
 $even(s(s(x))) \leftarrow even(x)$

$odd(s(0)) \leftarrow$
 $odd(s(s(x))) \leftarrow odd(x)$

completion

$\forall x : nat(x) \leftrightarrow x = 0 \vee \exists y : x = s(y) \wedge nat(y)$

$\forall x : even(x) \leftrightarrow x = 0 \vee \exists y : x = s(s(y)) \wedge even(y)$

$\forall x : odd(x) \leftrightarrow x = s(0) \vee \exists y : x = s(s(y)) \wedge odd(y)$

is $\forall x : nat(x) \rightarrow even(x) \vee odd(x)$ true in completion?

$\leftarrow nat(x), not(even(x)), not(odd(x))$ does not terminate

There is a model in which

$\forall x : nat(x) \rightarrow even(x) \vee odd(x)$ is *false*

domain: $N \cup Z$

$J(0) = 0_N, J(s(n)) = n + 1, J(s(z)) = z + 1$

$I(nat/1) = \{x \mid x \in N \cup Z\}$

$I(even/1) = \{0_N, 2_N, 4_N, \dots\}$

$I(odd/1) = \{1_N, 3_N, 5_N, \dots\}$

all elements of Z satisfy

$nat(x) \leftrightarrow x = 0 \vee \exists y : x = s(y) \wedge nat(y)$

$true \leftrightarrow false \vee true$

$true \leftrightarrow true$

$even(x) \leftrightarrow x = 0 \vee \exists y : x = s(s(y)) \wedge even(y)$

$false \leftrightarrow false \vee true \wedge false$

$false \leftrightarrow false \vee false$

$false \leftrightarrow false$

with $Z = \{a, b, c\}, I(0) = a, I(s(a)) = b, I(s(b)) = c, I(s(c)) = a$

also model of $IFF(P)$ but free equality axioms violated

2.6 INDUCTIVE DEFINITIONS

Def: (simple) definition

a description of a concept in terms of more primitive concepts

example:

x is grandparent of y iff

there exists a z such that

x is a parent of z and

z is a parent of y

$grandparent(x, y) \leftarrow parent(x, z), parent(z, y)$

IFF: this is the only way to obtain a grandparent

there are no other rules

completion captures the intended meaning of such simple definitions

Def: inductive definition

a description of a concept in terms of *itself*
and of more primitive concepts

examples:

x is a natural number *iff* $x = 0$ or if it is the
successor of a natural number

- 0 is a natural number
- if x is a natural number,
then $s(x)$ is a natural number
- nothing else is a natural number

$nat(0) \leftarrow$

$nat(s(x)) \leftarrow nat(x)$

- a variable is a term
- a constant is a term
- if f is a n -ary functor and t_1, \dots, t_n are
terms
 $f(t_1, \dots, t_n)$ is a term
- nothing else is a term

- an edge from x to y is a path from x to y
- an edge from x to z followed by a path from z to y is a path from x to y
- nothing else is a path

$path(x, y) \leftarrow edge(x, y)$

$path(x, y) \leftarrow edge(x, z), path(z, y)$

In all these cases, $COMP(P)$ is too weak to capture this intended meaning

natural numbers: because there are other objects

path: because there are several fixpoints

The intended meaning is the least fixpoint M_P which is captured by the non-monotonic closed world assumption

Is there a reconstruction in classical monotonic logic?

Yes, as second order logic

Has been studied in Mathematics:

theory of definitions

Has been studied in Artificial Intelligence:

circumscription

Has been studied in Logic Programming:

inductive definitions

$$IND(P) = P \cup FEQ \cup DCL \cup MIM$$

- *FEQ*: Free Equality axioms as in *COMP(P)*
- *DCL*: Domain Closure axioms they define that the elements of the Herbrand Universe are the only objects
- *MIM*: second order axioms expressing that an interpretation is only a model if it is a Minimal Model of all the other axioms

$$IND(P) \models \neg A \text{ iff } \neg A \in CWA(P)$$

SLD is a sound (if $\text{cas } \theta$ then $A\theta$ is true)

if it finitely fails, then $\neg A$ is true)

but not a complete proof procedure (infinite branches)

Complete procedures exist for function free logic programs based on:

tabulation (OLDT proof procedure, e.g. XSB Prolog) Or bottom-up methods (implementing T_P like procedures)

anyway, with (efficient) depth first search, the search in the SLD-tree was already incomplete

so, complementary to the correct declarative semantics, programmer should “show” that query terminates

Exercises

8. Consider the following program P :

parent(a,b) \leftarrow .
parent(b,c) \leftarrow .
anc(X,Y) \leftarrow parent(X,Y).
anc(X,Y) \leftarrow anc(X,Z), anc(Z,Y).

a. (i) Give the completion $Comp(P)$ of P . (ii) Give 2 different Herbrand models of $Comp(P)$. (iii) Give a Herbrand interpretation which is a model of P but not of $Comp(P)$. (iv) Which of the following are true? $Comp(P) \models anc(a,c)$, $Comp(P) \models anc(c,a)$, $Comp(P) \models \neg anc(c,a)$.

b. Is $\neg anc(c,a)$ provable by (i) CWA/IND(P), (ii) NAF, (iii) Completion, (iv) Prolog?

9. Consider the following program P :

parent(a,b) \leftarrow .
parent(b,c) \leftarrow .
anc(X,Y) \leftarrow parent(X,Y).
anc(X,Y) \leftarrow anc(X,Z), parent(Z,Y).

a. Is $\neg anc(c,a)$ provable by (i) CWA/IND(P), (ii) NAF, (iii) Completion, (iv) Prolog?

b. Same questions for $P \cup \{parent(a,a)\}$.

10. Consider the following program P :

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

Is $\neg q(a)$ true according to (i) CWA/IND(P), (ii) NAF, (iii) Completion, (iv) Prolog?

11. Consider the following program P :

r(X) \leftarrow p(X), q(X).
p(a) \leftarrow .
p(X) \leftarrow p(f(X)).
q(b) \leftarrow .

Give the completion of P .

Which of $\neg r(a)$, $\neg r(b)$, and $\neg r(c)$ are provable by (i) CWA/IND(P), (ii) NAF, (iii) Completion, (iv) Prolog?

3. GENERAL/NORMAL PROGRAMS

3.1 Motivation

- what? negative literals in bodies and goals
- definite programs \equiv universal Turing machine

So, WHY?

$even(0) \leftarrow$
 $even(s(x)) \leftarrow not(even(x))$
one can as well write:
 $even(s(s(x))) \leftarrow even(x)$

$flies(x) \leftarrow bird(x), not(abnormal_flier(x))$
 $bird(x) \leftarrow parrot(x)$
 \vdots
 $bird(x) \leftarrow ostrich(x)$
 $abnormal_flier(x) \leftarrow ostrich(x)$
 $abnormal_flier(x) \leftarrow penguin(x)$

much more concise than

$flies(x) \leftarrow bird(x), normal_flier(x)$
 $normal_flier(x) \leftarrow parrot(x)$
 \vdots
 $normal_flier(x) \leftarrow owl(x)$

note: still negation in:

$normal_flier(x) \leftarrow x \neq ostrich, x \neq penguin$

core fragment of event calculus:

$$\begin{aligned} \text{holds}(Prop, Time) &\leftarrow \\ &\text{happens}(Event), \text{before}(Event, Time), \\ &\text{initiates}(Event, Prop) \\ &\text{not}(\text{clipped}(Event, Prop, Time)) \\ \text{clipped}(Event, Prop, Time) &\leftarrow \\ &\text{happens}(Clip), \text{terminates}(Clip, Prop), \\ &\text{between}(Clip, Event, Time) \end{aligned}$$

very cumbersome without negation:

assume at most one event at time n , assume effects of event at time n visible at time $n + 1$

$$\begin{aligned} \text{not_clipped}(E, P, T) &\leftarrow T \leq E \\ \text{not_clipped}(E, P, T) &\leftarrow E < T, E' = E + 1, \\ &\text{no_clipping_event}(E, P), \\ &\text{not_clipped}(E', P, T) \\ \text{no_clipping_event}(T, P) &\leftarrow \text{no_event}(T) \\ \text{no_clipping_event}(T, \text{ontable}(Block)) &\leftarrow \\ &\text{action}(T, \text{put}(B)) \\ &\vdots \\ \text{no_event}(4) \\ &\vdots \end{aligned}$$

For each type of action, one need to enumerate all properties it does not change

frame problem

In Prolog:

```
not(X) :- X, !, fail.  
not(X).
```

it works?:

```
parrot(kiki) ←  
ostrich(oliver) ←  
← flies(kiki)  
← bird(kiki), not(abnormal_flier(kiki))  
← parrot(kiki), not(abnormal_flier(kiki))  
← not(abnormal_flier(kiki))  
↪ ← abnormal_flier(kiki)  
    ← ostrich(kiki)  
    failure
```

□

```
← even(s(s(s(0))))  
← not(even(s(s(0))))  
↪ ← even(s(s(0)))  
    ← not(even(s(0)))  
    ↪ ← even(s(0))  
        ← not(even(0))  
        ↪ ← even(0)
```

□

failure

□

failure

it works ...

... sometimes

$p(a) \leftarrow q(b) \leftarrow$
find x such that $p(x)$ and not $q(x)$
 $\leftarrow p(x), \text{not}(q(x)) \leftarrow \text{not}(q(x)), p(x)$
 $\leftarrow \text{not}(q(a)) \quad \rightsquigarrow \leftarrow q(x)$
 $\rightsquigarrow q(a) \quad \square$ with $x = b$
failure failure
 \square

find even numbers
first branch second branch
 $\leftarrow \text{even}(x) \quad \leftarrow \text{even}(x)$
 \square with $x = 0 \quad \leftarrow \text{not}(\text{even}(x'))$
 $\rightsquigarrow \leftarrow \text{even}(x')$
 \square with $x' = 0$
failure

problem: subderivations $\leftarrow q(x)$ and $\leftarrow \text{even}(x')$ look for values which make them succeed

To make the query succeed, they should look for values which make $\leftarrow q(x)$ and $\leftarrow \text{even}(x')$ fail

need to construct terms which make unifications fail: "constructive negation"

more drastic solution: allow only selection of $\text{not}(p(x))$ when $p(x)$ is ground

execution can block: *floundering negation*

3.2 The SLDNF proof procedure

basic intuitions:

- $\text{not}(P)$ succeeds if P finitely fails
- $\text{not}(P)$ finitely fails if P succeeds
- $\text{not}(P)$ does not terminate if P does not
- top level query flounders if somewhere $\text{not}(x)$ is selected with x non ground

Lloyds' definition: concept of SLDNF-tree undefined when a subderivation does not terminate

Better definition by Apt and Doets JLP feb 94

SLDNF resolvent

two cases:

- $\leftarrow L_1, \dots, \underline{L_i}, \dots, L_n$
standardised apart clause: $H \leftarrow B_1, \dots, B_k$
mgu σ of *positive* literal L_i and head H
resolvent:
 $\leftarrow (L_1, \dots, B_1, \dots, B_k, \dots, L_n)\sigma$
- $\leftarrow L_1, \dots, \underline{L_i}, \dots, L_n$
 L_i is negative
the substitution is ϵ , the empty substitution
the resolvent:
 $\leftarrow L_1, \dots, L_{i-1}, L_{i+1}, \dots, L_n$

successful tree: has a leaf labeled \square

finitely failed tree: finite, all leaves are marked with *failure*

forest: a set of trees with one main tree and with some nodes having a *subsidiary* tree

nodes with negative literal selected will be given a subsidiary tree

partial SLDNF-tree for a program P and a query $\leftarrow \mathcal{A}$

- a forest with a main tree with $\leftarrow \mathcal{A}$ as single node and no subsidiary trees is a partial SLDNF-tree
- an extension of a partial SLDNF-tree is a partial SLDNF-tree

extension: obtained by applying the following operation on *all* unmarked leaves different from \square

avoids one works infinitely on one tree without ever considering the others in the forest

- if no literal selected: select one

- if positive literal selected:
 - if no resolvents, then mark leaf with *failure*
 - if resolvents, then add them as the children of the node

- if negative literal ($\text{not}(A)$) selected:
 - if A is non-ground then mark leaf with *floundering*
 - if A is ground then
 - if it has no subsidiary tree, create subsidiary tree with root $\leftarrow A$
 - if it has a successful subsidiary tree, mark leaf with *failure*
 - if it has a finitely failed subsidiary tree, add the resolvent as child

SLDNF-tree for query Q : limit of the sequence $\mathcal{F}_0, \mathcal{F}_1, \dots$

with \mathcal{F}_0 the initial partial SLDNF-tree

with \mathcal{F}_{i+1} the extension of \mathcal{F}_i

successful/failed SLDNF-tree: the main tree is successful/failed

(some subsidiary trees can be infinite)

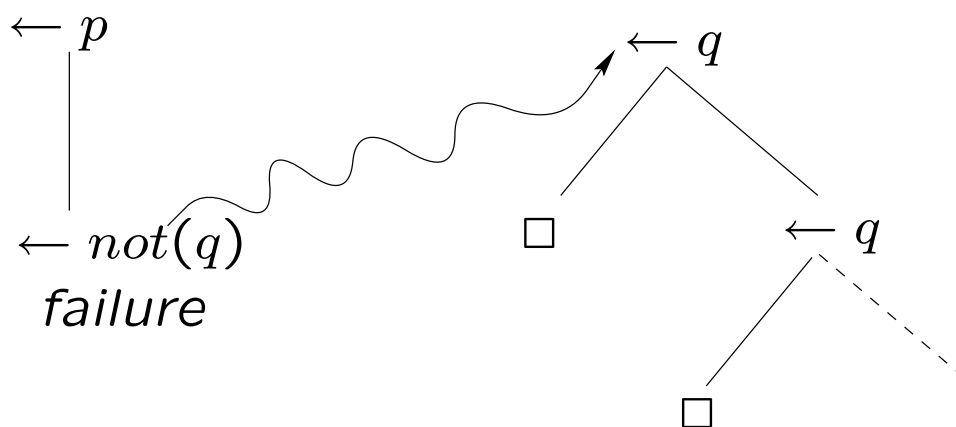
finite tree: no infinite paths

a failed (main tree) can have infinite subsidiary trees!

$$p \leftarrow \text{not}(q)$$

e.g. $q \leftarrow$

$$q \leftarrow q$$



SLDNF-derivation: branch of main tree + subsidiary trees

computed answer substitution of SLDNF-refutation:
composition of substitution on the branch of main tree restricted to variables of the query

ϵ was substitution when negative literal was selected

selection/computation rule: selection of literal is based on the partial SLDNF-tree

safe computation rule: never selects non-ground negative literal

blocked goal: all literals are nonground and negative

program P flounders for query $\leftarrow \mathcal{A}$ if the SLDNF-tree has a blocked goal

will a program flounder? undecidable

but restricted classes of non-floundering programs/queries exist:

in every goal: each variable occurs in a positive literal i.e.

- every variable of query occurs in a positive literal (*allowed query*)
- clauses defining preds. occurring positively in bodies: every clause variable has occurrence in a positive body literal (*allowed clause or range restricted clause*)
- clauses defining preds. occurring only negatively in bodies (calls are ground upon selection): every clause variable has occurrence in head or positive body literal (*admissible clause*)

strong condition, e.g. append is not allowed

for such programs: no floundering, all computed answers are ground

Exercises

1. Consider P :

$q(X) \leftarrow r(X)$, not $p(X)$.

$p(X) \leftarrow s(X)$.

$p(a)$.

$r(a)$.

$r(b)$.

$s(c)$.

Construct SLDNF-trees for the queries $\leftarrow q(a)$. and $\leftarrow q(X)$.

2. Add the clause

$p(b) \leftarrow p(b)$.

to the previous program. What changes to the SLDNF-tree of the query $\leftarrow q(X)$?

4. SEMANTICS of NORMAL PROGRAMS

4.1 Completion Semantics

General clauses have one positive literal in head, so they are true in B_P , the interpretation consisting of all ground atoms
i.e. B_P is a model, and no negative information is derivable

The construction of $COMP(P) = IFF(P) \cup FEQ$ is also applicable on general programs

Is $COMP(P)$ a good semantics?

Consider $p \leftarrow not(p)$

the completion: $p \leftrightarrow not(p)$

INCONSISTENT

also CWA: adding $\neg p$ because p cannot be proven results in inconsistent theory

Note: $T_P \uparrow 1 = \{p\}$, $T_P \uparrow 2 = \emptyset$, $T_P \uparrow 3 = \{p\}$, ...: no fixpoint, T_P is not monotonic

Fixpoints of T_P are models of $COMP(P)$, as before; however not a fixpoint for all programs

One approach defines classes of programs for which $COMP(P)$ is consistent

captures behaviour of SLDNF for programs with consistent completion

but does not cover all consistent programs
(floundering is undecidable)

but SLDNF draws also reasonable conclusions from programs with inconsistent completion

e.g. adding $p \leftarrow not(p)$ to a well behaving program not mentioning p preserves all previous SLDNF-derivations

One rather simple approach to capture behaviour of SLDNF is due to Melvin Fitting and uses a three valued logic

true: provable

false: failure

Undefined (or \perp): cannot decide nontermination

A three valued interpretation: $I = (I^+, I^-)$

I^+ : the true atoms

I^- : the false atoms

$B_P \setminus (I^+ \cup I^-)$: the undefined atoms

with respect to the “real” model:

I^+ (I^-): an underestimation of true (false) atoms

$B_P \setminus I^-$ ($B_P \setminus I^+$): an overestimation of true (false) atoms

$$I^+ \cap I^- = \emptyset$$

there is also a 4-valued logic with a fourth truth value \top , then atoms in $I^+ \cap I^-$ have value \top

I is total when $I^+ \cup I^- = B_p$

truth ordering $<_t$: $f <_t u <_t t$

truth of ground clause $A \leftarrow B_1, \dots, B_m, \neg C_1, \dots, \neg C_n$:

$I(\neg C) = \neg(I(C))$ where $\neg(u) = u$, $\neg(t) = f$, $\neg(f) = t$

$I(B_1, \dots, \neg C_n) = \min_t(I(B_1), \dots, I(\neg C_n))$

$I(A \leftarrow Body) = I(A) \geq_t I(Body)$ hence t or f !

$COMP(P)$ is consistent in this 3-valued logic

A T_P like operator: Φ_P (Fitting operator)

Consider all ground clauses for atom A :

$A \leftarrow Body_1 \quad \dots \quad A \leftarrow Body_m$

$\Phi_P(I)(A) = \max_t(I(Body_1), \dots, I(Body_m))$

true a proof iff exist a true body

false finite failure iff all bodies false

$\Phi_P(I^+, I^-) = (S^+, S^-)$ where

$H \in S^+$ iff $\Phi_P(I)(H) = true$

$H \in S^-$ iff $\Phi_P(I)(H) = false$

rules: $a \leftarrow \dots b_i \dots \neg c_j \dots$

$S^+ = \{H \mid \exists \mathcal{B} : H \leftarrow \mathcal{B} \in \text{ground}(P)$
and \mathcal{B} true in $(I^+, I^-)\}$

\mathcal{B} true in (I^+, I^-) : all b_i in I^+ ,
all c_j in I^-

$S^- = \{H \mid \forall \mathcal{B} : H \leftarrow \mathcal{B} \in \text{ground}(P)$
implies \mathcal{B} is false in $(I^+, I^-)\}$

\mathcal{B} false in (I^+, I^-) :

at least one b_i in I^- ,

or at least one c_j in I^+

Alternatively

$\Phi_P(I^+, I^-) = (S^+, S^-)$ where

$S^+ = A^-$: the result of applying the rules using the underestimation

$a \leftarrow b_i \dots \neg c_j$ derives $a \in A^-$ when all b_i are true according to underestimation I^+

and

when all $\neg c_j$ are true according to underestimation I^-

$S^-: B_P \setminus A^+$ with A^+ the result of applying the rules using the overestimation

$a \leftarrow b_i \dots \neg c_j$ derives $a \in A^+$ when all b_i are true according to overestimation $B_P \setminus I^-$

and

when all $\neg c_j$ are true according to overestimation $B_P \setminus I^+$

Properties of Φ_P

- If I consistent Body cannot be true AND false then $\Phi_P(I)$ is consistent
- knowledge order $<_k$:
 $u <_k t, u <_k f, f <_k \top, t <_k t\top$
 $(I_1^+, I_1^-) \leq_k (I_2^+, I_2^-)$ iff
 $I_1^+ \subseteq I_2^+$ and $I_1^- \subseteq I_2^-$
- $(I_2^+, I_2^-) \geq_k (I_1^+, I_1^-)$ implies
 $\Phi_P(I_2^+, I_2^-) \geq_k \Phi_P(I_1^+, I_1^-)$:
monotonicity and existence of fixpoints
- in least fixpoint: $I^+ \cap I^- = \emptyset$
- Φ_P is not continuous for all programs
 $S^- = B_P \setminus T_p(B_p \setminus I^-)$, $T_P \downarrow$ is not continuous
hence false atoms not recursively enumerable for all programs (no procedure is always “effective”)
- fixpoints of Φ_P are three valued models of $COMP(P)$, i.e. $\Phi(I) = I$ iff $I \models_3 COMP(P)$
- With $lfp(\Phi_P) = (I^+, I^-)$
if $A \in I^+$ then $COMP(P) \models_3 A$
if $A \in I^-$ then $COMP(P) \models_3 \neg A$

SOUNDNESS

if SLDNF computes answer θ for query $\leftarrow \mathcal{A}$
then $COMP(P) \models_3 \forall \mathcal{A} \theta$

if SLDNF finitely fails for query $\leftarrow \mathcal{A}$ then
 $COMP(P) \models_3 \forall \neg \mathcal{A} = \neg \exists \mathcal{A} = \forall \leftarrow \mathcal{A} = \leftarrow \mathcal{A}$

these soundness results hold also for two valued completion –
everything follows from inconsistent theory–

COMPLETENESS

No completeness for all programs

Let $\phi(P) \uparrow \omega$ be (I^+, I^-)

if $A\theta \in I^+$ then SLDNF finds proof for $\leftarrow A$
or flounders

if $\forall \theta : A\theta \in I^-$ then SLDNF finitely fails for
 $\leftarrow A$ or flounders

“empty box” —implementations do not perform a complete search— UNLESS absence of floundering and termination is proved for your program and query

some examples:

$$r \leftarrow p \quad r \leftarrow \text{not}(p) \quad p \leftarrow p$$

$$\text{lfp}(\Phi_P) = (\emptyset, \emptyset)$$

Completion: $r \leftrightarrow p \vee \neg p$, $p \leftrightarrow p$

$COMP(P) \models_2 r$, but neither r nor $\neg r$ is a logical consequence of $COMP(P)$ in three valued semantics

Queries $\leftarrow r$ and $\leftarrow \text{not}(r)$ do not terminate and do not flounder in agreement with $\text{lfp}(\Phi_P)$

$$p \leftarrow \text{not}(q(x)) \quad q(a) \leftarrow \quad r(b) \leftarrow$$

$$\text{lfp}(\Phi_P) = (\{p, q(a), r(b)\}, \{q(b), r(a)\})$$

$COMP(P) \models_3 p$, $COMP(P) \models_3 \text{false} \leftarrow \text{not}(p)$

ideally $\leftarrow p$ should succeed, it flounders

ideally $\leftarrow \text{not}(p)$ should fail finitely, it flounders

Floundering: SLD is “trained” to find substitutions which make goal succeed, when it flounders it gives up because it is asked to construct a substitution which makes the goal fail

constructive negation does construct such substitutions

Exercises: The Fitting operator Φ_P

1. Consider P :

$q(X) \leftarrow r(X)$, not $p(X)$.

$p(X) \leftarrow s(X)$.

$p(b) \leftarrow p(b)$.

$p(a)$.

$r(a)$.

$r(b)$.

$s(c)$.

Compute the least fixpoint of the Fitting operator.

2. Consider P :

$\text{shaves}(b,X) \leftarrow \text{citizen}(X)$, not $\text{shaves}(X,X)$.

$\text{citizen}(a)$.

$\text{citizen}(b)$.

Compute the least fixpoint of the Fitting operator.

3. Consider P :

$r(a)$.

$r(b) \leftarrow \text{not } q$.

$q \leftarrow p(X)$.

$p(f(X)) \leftarrow p(X)$.

Compute the least fixpoint of the Fitting operator.

4.2 Intended model semantics

4.2.1 Stratified programs as inductive definitions

BUT while completion does attempt to capture behaviour of SLDNF, this often does not correspond to intended meaning as we said already for definite programs

For definite programs, the least H-model did correspond to intended meaning

Is there an *intended* model for general programs?

For the class of *stratified* programs, there is general agreement

In FOL:

$$p \leftarrow \neg q, r$$

$$q \leftarrow \neg p, r$$

$$p \vee q \leftarrow r$$

all have the same meaning

not when reading them as definitions

$person(a) \leftarrow$
 $person(b) \leftarrow$
 $male(a) \leftarrow$
as definition $male(b)$ is false
once $male/1$ defined it can be used
 $female(x) \leftarrow not(male(x)), person(x)$

as FOL, there are 2 models:

$\{person(a), person(b), male(a), male(b)\}$ and
 $\{person(a), person(b), male(a), female(b)\}$

as definitions, only ONE model:

$\{person(a), person(b), male(a), female(b)\}$

$person(a) \leftarrow$
 $person(b) \leftarrow$
 $female(b) \leftarrow$
as definition $female(a)$ is false
 $male(x) \leftarrow not(female(x)), person(x)$

same FOL formula as above

as FOL: 2 models, as definitions: ONE

once a concept is defined, its NEGATION can be used to define other concepts

programs in layers: STRATA i.e. *stratified* program

stratum 1:

$person(a) \leftarrow$

$person(b) \leftarrow$

$male(a) \leftarrow$

stratum 2:

$female(x) \leftarrow not(male(x)), person(x)$

some definitions:

Def. Dependency graph D_P

$(p, q)^+ \in D_P$ if

a clause $p(\dots) \leftarrow \dots, q(\dots), \dots \in P$

$(p, q)^- \in D_P$ if

a clause $p(\dots) \leftarrow \dots, not(q(\dots)), \dots \in P$

$(p, q)^+$: positive arc $(p, q)^-$: negative arc

Def. program P is stratified if D_P has no cycles containing a negative arc (no recursion through negation)

$even(s(s(x))) \leftarrow even(x)$

$D_P = \{(even/1, even/1)^+\}$

cycle but no negative arc: stratified

$even(s(x)) \leftarrow not(even(x))$

$D_P = \{(even/1, even/1)^-\}$

cycle with negative arc: not stratified

$even(s(x)) \leftarrow odd(x)$

$odd(s(x)) \leftarrow even(x)$

$D_P = \{(even/1, odd/1)^+, (odd/1, even/1)^+\}$

cycle but no negative arc: stratified

$person(a) \leftarrow \dots$

$male(a) \leftarrow \dots$

$female(x) \leftarrow not(male(x)), person(x)$

$D_P = \{(female/1, male/1)^-, (female/1, person/1)^+\}$

no cycles: stratified

a set of strata P_1, \dots, P_n is a *stratification* of P if

- P is the union of P_1, \dots, P_n
- the strata are disjoint
- if a relation occurs positively in P_i , its definition is contained in $P_1 \cup \dots \cup P_i$ i.e. in same or lower stratum
- if a relation occurs negatively in P_i , its definition is contained in $P_1 \cup \dots \cup P_{i-1}$ i.e. in lower stratum

undefined predicates are in P_1 and P_1 can be empty!

$p \leftarrow not(q)$

stratified: $P_1 = \emptyset, P_2 = \{p \leftarrow not(q)\}$

as definitions, q is false, p is true, as FOL, two models

Lemma: A program is stratified iff it admits a stratification

Proof:

- a stratification of $P : D_P$ can have no cycles through negation
- stratified: use D_P to construct strata: select a minimal group of predicates which depend on predicates already placed in a stratum or (positively) on themselves) (always exist) put the group in the lowest stratum satisfying the conditions

Constructing the *intended* model (the explicit enumeration of what is defined) is by using *cumulative powers*

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

$$T_P \uparrow (n + 1) (I) = T_P(T_P \uparrow n(I)) \cup T_P \uparrow n (I)$$

keep what you already got

$$T_P \uparrow \omega (I) = \bigcup_{n=0}^{\infty} T_P \uparrow n (I)$$

Let P_1, \dots, P_n be a stratification of P

Define:

$$M_1 = T_{P_1} \uparrow \omega (\emptyset) (= T_{P_1} \uparrow \omega)$$

$$M_2 = T_{P_2} \uparrow \omega (M_1)$$

⋮

$$M_n = T_{P_n} \uparrow \omega (M_{n-1})$$

The *perfect model* M_P is M_n

- M_P is a minimal model
- M_P is *smaller* than any other minimal model

smaller: let (I_1, \dots, I_n) and (J_1, \dots, J_n) be models with I_k, J_k the atoms true in stratum k

$(I_1, \dots, I_n) < (J_1, \dots, J_n)$ if $I_1 \subset J_1$ or $I_1 = J_1 \wedge I_2 \subset J_2$ or ...

- M_P is the model of $IND(P)$ ($P \cup FEQ \cup DCL \cup MIM$)
- M_P does not depend on stratification
- M_P is a H-model of P and of $COMP(P)$
 $COMP(P)$ is consistent

Exercises Stratification, Perfect Models

1. Consider the program P :

$p(X) \leftarrow r(X), \text{ not } q(X).$

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

$q(X) \leftarrow s(X), q(X).$

$r(a).$

$s(b).$

(a) Give the dependency graph. (b) Give –if possible– 2 different stratifications. (c) Compute M_P the perfect model. (d) Is M_P a fix-point of T_P ? (e) Is M_P equal to $T_P \uparrow \omega$?

2. Same questions for :

$p(X) \leftarrow \text{not } q(X).$

$p(X) \leftarrow p(X).$

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

$q(X) \leftarrow s(X), r(X).$

$r(a).$

$s(b).$

Concept of stratification and of inductive definitions can be stretched further

$$\begin{aligned} \text{even}(0) &\leftarrow \\ \text{even}(s(x)) &\leftarrow \text{not}(\text{even}(x)) \end{aligned}$$

is a meaningful program, a sensible definition ($\text{even}(x) \leftarrow \text{not}(\text{even}(s(x)))$) is not)

the infinite ground program:

$$\begin{aligned} \text{even}(0) &\leftarrow \\ \text{even}(s(0)) &\leftarrow \text{not}(\text{even}(0)) \\ \text{even}(s(s(0))) &\leftarrow \text{not}(\text{even}(s(0))) \\ &\vdots \end{aligned}$$

is isomorphic to the infinite propositional program:

$$\begin{aligned} \text{even_0} &\leftarrow \\ \text{even_s_0} &\leftarrow \text{not}(\text{even_0}) \\ \text{even_s_s_0} &\leftarrow \text{not}(\text{even_s_0}) \\ &\vdots \end{aligned}$$

which is stratified

a stratification of B_P (all ground atoms)

local stratification

... and further

$even(x) \leftarrow zero(x)$

$even(x) \leftarrow pred(x, y), not(even(y))$

$zero(0) \leftarrow$

$pred(s(x), x) \leftarrow$

$ground(P)$ is not locally stratified e.g.

$even(s(0)) \leftarrow pred(s(0), s(0)), not(even(s(0)))$

However it is a sensible definition (because all instances of $pred(x, x)$ are false in the model)

removal of all rules from $ground(P)$ which have a false condition wrt $zero/1$ and $pred/2$ (which have correct definitions) results in a locally stratified program, hence $even/1$ is correctly defined

$p \leftarrow not(p)$ is not correct as a definition, it is a bug. One cannot define a concept in terms of its own negation

$shaves(b, X) \leftarrow citizen(X), not shaves(X, X).$

$citizen(a).$

$citizen(b).$

bug for $shaves(b, b)$

4.2.2 Well-founded programs as inductive definitions

Two alternative ways for computing the well-founded model (using $ground(P)$)

A. As fixpoint of Positive Induction Operator

Represent:

facts as $p \leftarrow true$

atoms without definition as $p \leftarrow false$

Candidate proof tree \mathcal{T} of atom p

- p is the root of \mathcal{T}
- Each non leaf is an atom q ; its descendants are the literals in the body of some rule $q \leftarrow Body$
- Each leaf is either $true$, or $false$ or a negative literal
- Tree is finite

$$\mathcal{PI}(I) = S$$

with $I = (I^+, I^-)$, $S = (S^+, S^-)$ partial interpretations

- $p \in S^+$ iff p has candidate proof tree with ALL leaves *true* in I
- $p \in S^-$ iff ALL candidate proof trees of p have a leaf that is *false* in I

monotone in knowledge order \leq_k

more *true/false* atoms, more decisions

hence least fixpoint exist:

WELL-FOUNDED MODEL

one can discard candidate trees with repetition of same atom on a branch

$$r \leftarrow p$$

$$r \leftarrow \text{not } p$$

$$p \leftarrow p$$

$$\mathcal{PI}(\emptyset, \emptyset) = (\emptyset, \{p\})$$

$$\mathcal{PI}(\emptyset, \{p\}) = (\{r\}, \{p\}) = \text{fixpoint}$$

B Gelfond-Lifschitz operator \mathcal{GL}

$$\mathcal{GL}(I) = S$$

with $I = (I^+, I^-)$, $S = (S^+, S^-)$ partial interpretations

- S^+ : atoms that are definitely true
 - remove clauses with literal *not a* if $a \in B_P \setminus I^-$, i.e. it is possible that *not a false* is
 - remove negative literals from remaining clauses: $a \in I^-$ hence *not a* is definitely true
 - S^+ is the least Herbrand model of the remaining program
- S^- : atoms that are definitely false
 - remove clauses with literal *not a* if $a \in I^+$, i.e. *not a* is definitely *false*
 - remove negative literals from remaining clauses: $a \in B_P \setminus I^+$ i.e. it is possible that *not a true* is
 - S^- is the complement of the least Herbrand model of the remaining program

monotonic in knowledge order \leq_k :

- I^- increases: less clauses removed in computation of S^+ : larger lfp, increase of S^+
- I^+ increases: more clauses removed in computation of S^- : smaller lfp, increase of complement S^-

$$r \leftarrow p \quad r \leftarrow \text{not } p \quad p \leftarrow p$$

$\mathcal{GL}(\emptyset, \emptyset)$

S^+	S^-
$r \leftarrow p$	$r \leftarrow p$
	$r \leftarrow$
$p \leftarrow p$	$p \leftarrow p$
lfp = \emptyset	lfp = $\{r\}$, complement = $\{p\}$

$\mathcal{GL}(\emptyset, \{p\})$

$S^+ :$	$S^- :$
$r \leftarrow p$	$r \leftarrow p$
$r \leftarrow$	$r \leftarrow$
$p \leftarrow p$	$p \leftarrow p$
lfp = $\{r\}$	lfp = $\{r\}$, complement = $\{p\}$

$(\{r\}, \{p\})$ is fixpoint

Well-founded semantics corresponds best to *intuitions/common sense* of programmer

Theory of inductive definitions gives another foundation not dependent on common sense

Not always computable

SLDNF: sound, not complete

Better procedures: + tabulation (XSB-Prolog)

SLG-resolution

Without functors (DATALOG): polynomial in size of program

Correct/Total definition: well-founded model is two valued, no atoms undefined

Bug: undefined atoms (their definition depends on their own negation)

Exercises Well-founded semantics

1. Consider P :

$q(X) \leftarrow r(X)$, not $p(X)$.

$p(X) \leftarrow s(X)$.

$p(b) \leftarrow p(b)$.

$p(a)$.

$r(a)$.

$r(b)$.

$s(c)$.

Compute the well-founded model

2. Consider P :

$\text{shaves}(b,X) \leftarrow \text{citizen}(X)$, not $\text{shaves}(X,X)$.

$\text{citizen}(a)$.

$\text{citizen}(b)$.

Compute the well-founded model

5 INCOMPLETE INFORMATION

5.1 Inductive definitions extended with open predicates

Inductive definitions (and to some extent completion) do not leave room for incomplete information

one approach: disjunctive programs

e.g. $mother(x) \vee father(x) \leftarrow parent(x)$

another approach: default logic where “not” is understood as “don’t know”

e.g. $flies(x) \leftarrow bird(x), not(abnormal_flier(x))$

our approach: distinguish between

program part –inductive definitions– perhaps using some concepts which are left undefined

the *open* predicates

and knowledge part partial knowledge in FOL about the undefined predicates

ID-LOGIC

but first the Prolog/CLP approach

```
queens(Q,N) :- generate(Q,N,N),
               safe(Q),
               instantiate(Q).
generate([],0,_).
generate([X|T],M,N) :- M > 0,
                       X in 1..N,
                       M1 is M - 1,
                       generate(T,M1,N).

safe([]).
safe([X|T]) :- noAttack(X,1,T),
               safe(T).
noAttack(_,_,[]).
noAttack(X,N,[Y|Z]) :- X \= Y,
                       Y \= X + N,
                       X \= Y + N,
                       S is N + 1,
                       noAttack(X,S,Z).

instantiate([]).
instantiate([X|T]) :- enum(X),
                     instantiate(T).
```

Missing information encoded as list of queens
Constraint propagation directs the search

In ID-Logic

```
open position/2.
```

```
% DEFINITIONS
```

```
size(8).
```

```
row(R) :- size(N), R in 1..N.
```

```
column(C) :- size(N), C in 1..N.
```

```
row_has_queen(R) :- position(R,C).
```

```
% CONSTRAINTS
```

```
% the arguments of position/2 have the
```

```
%             correct types
```

```
row(R) <- position(R,C).
```

```
column(C) <- position(R,C).
```

```
% at least one queen on each row
```

```
row_has_queen(R) <- row(R).
```

```
% no more than one queen on each row
```

```
C1=C2 <- position(R,C1), position(R,C2).
```

```
% the configuration is illegal if
```

```
% a queen attacks a queen on a higher row
```

```
false <- position(R1,C1), position(R2,C2),
```

```
    R1<R2,
```

```
    (C1=C2 ; abs(R2-R1)=abs(C2-C1)).
```

- Semantics:

Augmented with a definition (a model, a set of facts) of the open predicates, the definition part becomes a standard logic program

Its meaning is given by the well-founded model (two valued if bug-free)

The definition of the open predicates is a *solution* if the constraints (first order logic expressions) evaluate to true in the well-founded model

- Procedural:

find a model for the open predicates

e.g. (for size(4))

position(1,2), position(2,4), position(3,1),
position(4,3)

How to compute

```
open broken_spokes/0, leaky_valve/0,  
    punctured_tube/0.
```

```
wobbly_wheel :- flat_tyre.  
wobbly_wheel :- broken_spokes.  
flat_tyre    :- punctured_tube.  
flat_tyre    :- leaky_valve.
```

How to solve query e.g. \leftarrow *wobbly_wheel*

Guess a program for the open predicates and
try a refutation
(it suffices to guess facts
is called abduction)

e.g. guess: *punctured_tube* \leftarrow

refutation:

```
 $\leftarrow$  wobbly_wheel  
 $\leftarrow$  flat_tyre  
 $\leftarrow$  punctured_tube  
 $\square$ 
```

other good guesses are *broken_spokes* \leftarrow and *leaky_valve* \leftarrow
also any combination of these three (are not minimal guesses)

Better approach: make the guesses while proving:

:

\leftarrow *punctured_tube*

this is an open predicate, and is unsolvable with guesses so far:

extend guesses with *punctured_tube* \leftarrow

with definite propositional programs: no problem with collecting guesses while proving: the SLD-tree constructed so far remains as it was

with definite predicate programs: what remains but perhaps one has to add a branch here and there

with general programs: can have drastic effect: what failed finitely may suddenly succeed!

SLDNFA: SLDNF + Abduction

With partial knowledge:

e.g. go and check: no broken spokes

one can add the “partial” knowledge as a FOL expression

$false \leftarrow broken_spokes$

also the query is a constraint:

$wobbly_wheel \leftarrow true$

How to use that during the derivation?

“trick”: write constraints in the form:

$false \leftarrow body$ i.e.

$false \leftarrow broken_spokes$

it is inconsistent to make a guess implying $broken_spokes \leftarrow$

$false \leftarrow not(wobbly_wheel)$

it is inconsistent to have no proof for $\leftarrow wobbly_wheel$

start proof with query:

$\leftarrow not(false)$

one should not find a proof for $\leftarrow false$

now we HAVE negation!

false ← *not(wobbly_wheel)*
false ← *broken_spokes*
wobbly_wheel ← *broken_spokes*
wobbly_wheel ← *flat_tyre*
flat_tyre ← *punctured_tube*
flat_tyre ← *leaky_valve*
 we look for a refutation of:

← *not(false)*
 ~→ ← *false* should fail!
 ← *not(wobbly_wheel)* ← *broken_spokes*
 ~→ ← *wobbly_wheel* fails with current guesses
 should succeed! don't guess *broken_spokes* ←
 ← *broken_spokes*
 not allowed to guess *broken_spokes* ←
 ← *wobbly_wheel*
 ← *flat_tyre*
 ← *punctured_tube*
 guess *punctured_tube* ←
 □
 fails
 all branches fail
 □
punctured_tube ← was "abduced"

another solution is by guessing/abducing;
leaky_valve ←

The predicate case: open $q/2$

program:

$p(a, b) \leftarrow$

$p(a, x) \leftarrow q(x, y)$

we want a proof for $\leftarrow p(a, a)$

$false \leftarrow not(p(a, a))$

query:

$\leftarrow not(false)$

$\rightsquigarrow \leftarrow false$ should fail!

$\leftarrow not(p(a, a))$

$\rightsquigarrow \leftarrow p(a, a)$ should succeed

$\leftarrow q(a, y)$

minimal guess: $q(a, sk1) \leftarrow$

□

failure

□ with $q(a, sk1) \leftarrow$ abduced

$sk1$: so called skolem constant, can be equal to existing constant (a) or be a new constant)

it can unify or fail to unify with other term as needed by proof

with integrity constraints: open $q/2$

program:

$p(a, b) \leftarrow$

$p(a, x) \leftarrow q(x, y)$

integrity constraint:

$false \leftarrow q(a, b)$

we want a proof for $\leftarrow p(a, a)$

$false \leftarrow not(p(a, a))$

query:

$\leftarrow not(false)$

$\rightsquigarrow \leftarrow false$ should fail!

first branch:

$\leftarrow q(a, b)$

fails but do not abduce $q(a, b) \leftarrow!$

second branch

$\leftarrow not(p(a, a))$

$\rightsquigarrow \leftarrow p(a, a)$ should succeed

$\leftarrow q(a, y)$

minimal guess: $q(a, sk1) \leftarrow$

but with constraint $sk1 \neq b$

□

failure

□ with $q(a, sk1) \leftarrow$ abduced and $sk1 \neq b$

i.e. $q(a, sk1) \leftarrow not(sk1 = b)$

SLDNFA Differences with SLDNF:

while searching for a *refutation*:

- A_i selected, is call to open predicate p/n :
backtrackpoint with *choices*:
 - already abduced facts about p/n (cannot invalidate anything)
 - if all these branches fail, then a new fact can be abduced (skolems)
(if needed add constraints to preserve existing finite failures; undo on backtracking beyond this point)
- L_i selected, is call to defined p/n :
(pos or neg literal) as SLDNF
- $not(A_i)$ selected, call to open pred. p/n :
start a subsidiary derivation: search for finite failure of $\leftarrow A_i$
- \square obtained: report success to “caller”
- finite failure: report failure to “caller”

while searching for *finite failure*:

- A_i selected, call to open predicate p/n :
each of resolvents (using abduced facts)
should fail finitely
- constrain skolems if that helps to force failure
- L_i selected, (defined literal): as SLDNF
- $\text{not}(A_i)$ selected, call to open predicate:
choice point
 - start a subsidiary derivation: search for refutation of $\leftarrow A_i$
 - if that fails, try to establish finite failure of rest of goal
- \square report: failure to caller
- finite failure: report success to caller

```
open: cat/1, parrot/1, ostrich/1

animal(x) :- cat(x).
animal(x) :- bird(x).
bird(x) :- parrot(x).
bird(x) :- ostrich(x).
fly(x) :- bird(x), not(abn_flier(x)).
abn_flier(x) :- ostrich(x).

% integrity constraints
% the goal <- fly(tweety) becomes:
false <- not(fly(tweety)) (1)
% other constraints
false <- not(animal(tweety)) (2)
false <- fly(felix) (3)
false <- not(animal(felix)) (4)
false <- cat(x), parrot(x) (5)
false <- cat(x), ostrich(x) (6)
false <- parrot(x), ostrich(x) (7)
```

$\leftarrow not(false)$
 $\rightsquigarrow \leftarrow false$ should fail!
 branch 1
 $\leftarrow not(fly(tweety))$
 $\rightsquigarrow \leftarrow fly(tweety)$ should succeed
 $\leftarrow bird(tweety), not(abn_flier(tweety))$
 branch 1.1
 $\leftarrow parrot(tweety), not(abn_flier(tweety))$
 abduce $parrot(tweety) \leftarrow not(abn_flier(tweety))$
 $\rightsquigarrow \leftarrow abn_flier(tweety)$ should fail
 $\leftarrow ostrich(tweety)$
 failure, do not abduce $ostrich(tweety) \leftarrow$
 \square
 branch 1 fails
 branch 2
 $\leftarrow not(animal(tweety))$
 $\rightsquigarrow \leftarrow animal(tweety)$ should succeed
 branch 2.1
 $\leftarrow cat(tweety)$
 abduce $cat(tweety)$
 will lead to \square but branch 5 will not fail
 branch 2.2
 $\leftarrow bird(tweety)$
 $\leftarrow parrot(tweety)$
 \square using abduced fact
 branch 2 fails
 branch 3
 $\leftarrow fly(felix)$
 $\leftarrow bird(felix)$
 branch 3.1
 $\leftarrow parrot(felix)$
 branch 3.1 fails
 branch 3.2
 $\leftarrow ostrich(felix)$
 branch 3.2 fails
 branch 3 fails
 and so on

Nixon's diamond

open: $abn_quaker/1, abn_repub/1$

$dove(x) \leftarrow quaker(x), not(abn_quaker(x))$

$hawk(x) \leftarrow republ(x), not(abn_repub(x))$

$quaker(nixon) \leftarrow$

$repub(nixon) \leftarrow$

integrity constraint

$false(x) \leftarrow hawk(x), dove(x)$

$\leftarrow not(false)$

$\rightsquigarrow \leftarrow false$ should fail!

$\leftarrow hawk(x), dove(x)$

$\leftarrow republ(x), not(abn_repub(x)), dove(x)$

$\leftarrow not(abn_repub(nixon)), dove(nixon)$

$\rightsquigarrow \leftarrow abn_repub(nixon)$ should succeed

abduce $abn_repub(nixon) \leftarrow$

□

failure

□

or

:

$\leftarrow not(abn_repub(nixon)), dove(nixon)$

$not(abn_repub(nixon))$ succeeds with
current state of affairs

$\leftarrow dove(nixon)$

$\leftarrow quaker(nixon), not(abn_quaker(nixon))$

$\leftarrow not(abn_quaker(nixon))$

$\rightsquigarrow \leftarrow abn_quaker(nixon)$ should succeed

abduce $abn_quaker(nixon) \leftarrow$

□

failure

□

Exercises

1. Consider the following program P :

```
open predicates broken/1, melted_fuse/1, general_power_failure/0.
%definitions
faulty(X) :- lamp(X), broken(X).
faulty(X) :- lamp(X), no_current(X).
no_current(X) :- fuse(X,Y), melted_fuse(Y).
no_current(X) :- general_power_failure.
lamp(a).
lamp(b).
%constraints
false <- faulty(b).
faulty(a) <- true.
```

Using SLDNFA, construct an abductive solution.

5.2 Stable semantics - Answer Set Programming

Consider an interpretation S of program P (a set of true atoms) (based on $\text{ground}(P)$)

- Remove clauses with literal $\text{not } a$ if $a \in S$, i.e. $\text{not } a$ is *false* according to S .
- Remove negative literals from remaining clauses: $a \notin S$ hence $\text{not } a$ is true according to S .

If S is the least Herbrand model of the remaining definite program (the *reduct*), then S is a *stable set*

Intuition: possible set of beliefs that a rational agent hold

Indeed: the reduct is the result of “believing” S and its least Herbrand model gives all consequences from this belief. If they coincide: a rational belief

A stable set is a minimal Herbrand model

Proof: Let S be a stable set

- It is a model of P :
 - consider $h \leftarrow \dots, q_i, \dots, \text{not } p_j, \dots$
 - If $\exists j : p_j \in S$ then body of clause is false, hence clause is true
 - Else, $\text{not } p_j$ true for all j and clause is true iff $h \leftarrow \dots, q_i, \dots$ is true. This is true as it is part of the reduct and S is a model of the reduct.
 - Hence S is a model of all clauses thus of P
- It is a minimal model of P :
 - Assume $S_1 \subset S$ a model of P .
 - S_1 is a model of the reduct because:
 - S_1 is model of $h \leftarrow \dots, q_i, \dots, \text{not } p_j, \dots$
 - $h \leftarrow \dots, q_i, \dots$ is part of the reduct iff for all $j : p_j \notin S$
 - but then $(S_1 \subset S)$ also $\text{not } p_j$ is true according to S_1 hence S_1 is a model of $h \leftarrow \dots, q_i, \dots$
 - hence of all clauses in the reduct.
 - but S is the least model of the reduct: contradiction

If well-founded model is total, then that model is the unique stable set (the stable model)

i.e. for correct definitions, the stable set and the well-founded model coincide

(compare with Gelfond-Lifschitz operator on $(I, B_P \setminus I)$)

what to do with incomplete knowledge?

```
p :- not neg_p.  
neg_p :- not p.
```

p and neg_p are undefined according to well-founded semantics.

$\{p\}$ and $\{neg_p\}$ are stable sets: *answer sets*: possible models for the undefined predicates

Use bug as feature

add constraints $\leftarrow l_1 \dots l_n$ to eliminate undesired models

```

% DEFINITIONS
p(1). p(2). p(3). p(4).
row(R) <- p(R).
col(C) <- p(C).
row_has_queen(R) <- posit(R,C).

% the ‘‘open’’ predicate posit/2
posit(R,C) <- not neg_posit(R,C), row(R), col(C).
neg_posit(R,C) <- not posit(R,C), row(R), col(C).

% CONSTRAINTS
% at least one queen on each row
<- row(R), not row_has_queen(R).
% no more than one queen on each row
<- posit(R,C1), posit(R,C2), not C1=C2.
% a queen does not attack a queen on a higher row
<- posit(R1,C1), posit(R2,C2), R1<R2, C1=C2.
<- posit(R1,C1), posit(R2,C2), R1<R2,
    abs(R2-R1)=abs(C2-C1).

```

Hamiltonian cycle

```
% DEFINITIONS
```

```
node(1). node(2). node(3).
```

```
edge(1,2). edge(2,3). edge(2,1). edge(3,1).
```

```
reachable(u) <- in(1,u).
```

```
reachable(u) <- reachable(v), in(v,u).
```

```
% open predicates in/2 and out/2 (neg_in)
```

```
% ‘‘in’’ edges form the Hamiltonian cycle
```

```
in(u,v) <- not out(u,v), edge(u,v).
```

```
out(u,v) <- not in(u,v), edge(u,v).
```

```
% CONSTRAINTS
```

```
% no two incoming edges
```

```
<- in(u,w), in(v,w), not u=v.
```

```
% no two outgoing edges
```

```
<- in(u,v), in(u,w), not v=w.
```

```
% all nodes reachable
```

```
<- node(u), not reachable(u).
```

ID-Logic and Answer Set programming:

active areas of research to develop systems supporting this declarative programming style

Exercises

1. $\{posit(1, 2), posit(2, 4), posit(3, 1), posit(4, 3)\}$ is a solution for the 4-queens problem. For the program on p. 91, construct a stable set that contains this solution (and verify that it indeed is a stable set).
2. $\{in(1, 2), in(2, 3), in(3, 1)\}$ is a Hamiltonian cycle for the graph of the program on p. 92. Construct a stable set that contains this solution (verify that it indeed is a stable set).