Chapter 3 Non-procedural Semantics

When we write down the clauses of a logic program we usually have in mind some intended interpretation. This is an interpretation of its constants, functors and predicates as the names of particular objects or individuals, particular functions, and particular relations. Relative to this interpretation, we make sure that each clause is a true statement, that is, we make sure that the intended interpretation is a model of the program. With this same interpretation in mind, we then compose a goal clause so that its conjunction of atoms denotes the relation an instance of which we want to compute.

Each successful evaluation of the goal clause will return an answer substitution

$$\{x1/t1,..,xk/tk\}$$

for all the variables of the goal clause. Relative to our intended interpretation of the program, the tuple of binding terms

$$\langle t1,..,tk\rangle$$

will denote a tuple of individuals, or, if some of the terms contain variables, a set of tuples of individuals. No matter what intended interpretation we had in mind, we would like to be sure that the tuples of individuals named by the answer substitution belong to the relation named by the goal clause.

We can use this naming requirement to give a non-procedural semantics for answer substitutions. We shall say that an answer substitution is correct if it names a set of instances of the relation named by the goal clause for every model of the program.

This is the model theory semantics of answer substitutions that we develop more fully in this chapter. Following van Emdem and Kowalski[1976], we show how it can be recast in the lattice theory framework of the Scott fixpoint semantics. Finally, we investigate its relationship with the procedural semantics of Chapter 1.

3.1 Interpretations and models

In the syntax of logic progams, the logical symbols and the implicitly quantified variables have a fixed meaning. The constants, functors and predicates are 'free' symbols to be used by the programmer as the names of any individuals, functions and relations he has in mind. An interpretation is the mapping from these free symbols to their

denotations.

DEFINITION
An **interpretation** comprises:
(1) A non-empty set D called the domain of the interpretation.
(2) For each constant the assignment of some individual in D.
(3) For each n-ary functor the assignment of some total function from D^n to D.
(4) For each n-ary predicate the assignment of an n-ary relation over D,
if n>0, this is some subset of D^n,
if n=0, it is a 0-ary relation which is truth value, true or false.

There is no requirement that different names be assigned different denotations.

Given an interpretation, each program clause, each goal clause, and each substitution also has a denotation as follows:

Denotation of a program clause

A clause

R(t1,..,tn)<-A1&..&Am

denotes one of the truth values, i.e. it is either true or false. It is true if and only if for every assignment of individuals of D to the variables of the clause, when the antecedent A1&..&Am is true, so is the consequent R(t1,..,tn).

A conjuction A1&..&Am is true if and only if each atom of the conjunction is true.

An atom R(t1,..,tn) is true if and only if
(i) for n=0, the predicate R has the value true,
(ii) for n>0, the tuple of individuals denoted by <t1,..,tn> is in the extension given to R.

The individual denoted by a term which is a variable is the individual assigned to the variable, that denoted by a term which is a constant is the individual assigned to the constant, and that denoted by a term f(t1,..,tk) is the value of the function assigned to f for the tuple of arguments denoted by <t1,..,tk>.

Denotation of a goal clause

The denotation of a goal clause

<-B1&..&Bn

is the denotation of its conjunction of atoms. This is a k-ary relation, where k is the number of variables in the clause.

Let x1,..,xk be these variables in lexicographical order. The relation denoted by the clause is:

(i) for k>0, the set of tuples

{<e1,..,ek> : for the assignment xi=ei, 1≤i≤k, B1&..&Bn is true}

(ii) for k=0, the truth value denoted by the conjunction of atoms B1&..&Bn

Denotation of a substitution

Let <t1,..,tk> be the tuple of binding terms of a substitution ordered by the lexicographical ordering of the variables x1,..,xk for which they are the bindings. The substitution denotes the k-ary relation

{<e1,..,ek> : <t1,..,tk> denotes <e1,..,ek> for some assignment to its variables}.

DEFINITIONS
(1) A **model** of a program P is an interpretation for which each clause is true.
(2) An answer substitution s for a goal clause C and program P is **correct** if for every interpretation that is a model of the program the relation denoted by s is included in the relation denoted by C. For the special case of a goal clause without variables, the answer **true** is **correct** if C is true for every model of P.
(3) Two answer substitutions (for the same variables) are **equivalent** if they denote the same relation in all interpretations.

The definition of a correct answer substitution could be reformulated as:

(A) for every model of P, Cs is true for every assignment to its variables.

or, as:

(B) the universal closure of Cs is a **logical consequence** of P.

The reformulation as (A) relies on the fact that s denotes a sub-relation of C if and only if Cs is true for every assignment to its variables. The re-formulation as (B) appeals to the standard model theory definition of logical consequence. The universal closure of Cs is (∀y1,..,yn)Cs, where y1,..,yn are all the variables of Cs. We shall generally use formulation (B).

The definition of equivalence of answer substitutions is the standard model theory definition of logical equivalence.

Example models

We shall describe three different interpretations, each of which is a

model of the program

```
append(Nil,x,x)<-
append(u.x,y,u.z)<-append(x,y,z).
```

For each model we shall give the denotation of the goal clause

```
<-append(x,y,A.B.Nil),
```

and the denotation of each of the answer substitutions

```
s1={x/Nil,y/A.B.Nil}
s2={x/A.Nil,y/P.Nil}
s3={x/A.B.Nil,y/nil}.
```

These are all the correct answers for the goal clause and program.

Interpretation 1

Domain is the set of natural numbers.

Among the assignments to the constants,functors and predicates:

"Nil","A","P" are 1,2,3 respectively.

"." is the multiplication function.

"append" is the relation {<m,n,p> : mxn=p}.

For this interpretation the assertion of the "append" program is the statement that

for all numbers n, 1xn=n,

and the implication is the statement that

for all m,n,p,i, if mxn=p then (ixm)xn=ixp.

In other words, the assertion is the true statement that 1 is a multiplicative identity, and the implication is the true statement that multiplication is associative. The interpretation is a model of the program.

The goal clause denotes the relation

{<m,n> : mxn=2x3x1} = {<1,6>,<2,3>,<3,2>,<6,1>}.

s1 denotes the singleton set of pairs {<1,6>}.

s2 denotes the singleton set of pairs {<2,3>}.

s3 denotes the singleton set of pairs {<6,1>}.

As required each substitution denotes a sub-relation of the goal clause relation.

Interpretation 2

Domain is a set of objects D that includes some base set of objects B, and is closed under an object constructor function. The constructor, cons, takes a pair of objects o1,o2 into an object o3 which is different from o1 and o2 and unique to each pair of arguments.

One of the base objects is the empty list []. All the other base objects are called atoms. A special subset of the constructed objects is the set of finite length lists {[o1,..,ok] : k>0}. The list [o1,..,ok] is the value of cons(o1,[o2,..,ok]).

Among the assignments to the constants,functors and predicates:

"Nil" is the empty list [].

"A" and "P" and all the other constants are each assigned a unique atom. Since each constant denotes a different atom, we can identify the atom with the constant.

"." is the constructor function cons.

"append" is the relation

$$\{\langle[],o,o\rangle : o \text{ in } D\}$$
$$\text{union } \{\langle[o1,..,oj],[oj+1,..,ok],[o1,....,ok]\rangle : o1,..,ok \text{ in } D\}$$
$$\text{union } \{\langle[o1,..,ok],o,cons(o1,cons(o2,..,cons(ok,o)\rangle : o,o1..,ok \text{ in } D\}$$

This is the intended intepretation for the program that we referred to informally in Chapter 1. We leave the reader to check that it is model of the program.

The relation denoted by the goal clause comprises the three pairs of lists <[],[A,B]>, <[A],[B]>, <[A,B],[]>. These are the denotations of the binding terms of s1,s2 and s3 respectively. For this model, the three answer substitutions exactly cover the goal clause relation.

Interpretation 3

Domain is the set of strings that are the terms of our logic program syntax.

Each constant c is assigned the term c.

Each n-ary functor f is assigned the function that takes the tuple of terms <t1,..,tn> into the term f(t1,..,tn). Thus, "." is the function that takes "A" and "Nil" into ".(A,Nil)". (Remember that A.Nil is just syntactic sugar for ".(A,Nil)").

"append" is the relation

{<Nil,t,t> : t a term}
union
{<.(t1,.(t2,--,.(tk,Nil)--),t,.(t1.(t2.--.(tk,t)--)> :
t,t1,..,tk terms}

We leave the reader to check that this interpretation is a model. The goal clause denotes the relation

{<Nil,.(A,.(B,Nil))> , <.(A,Nil),.(B,Nil)>, <.(A,.(B,Nil)),Nil>}.

Each pair in the relation is denoted by the pair of binding terms of exactly one of the answer substitutions.

Covering the goal clause relation

For the last two interpretations the three answer substitutions s1,s2,s3 have together covered the corresponding goal clause relation. To cover the goal clause relation for the first interpretation we should need to add the substitution

s4={x/B.Nil,y/A.Nil},

which denotes the extra tuple <3,2>. But the binding terms of this substitution do not denote an instance of the goal clause relation for the other two models of the program. A set of correct answer substitutions does not necessarily cover the goal clause relation for each model. The extent of the cover is constrained by the fact that each substitution must denote a relation that falls inside the goal clause relation for every other model.

3.2 Herbrand interpretations

The last interpretation given above is an example of a special class of interpretations called Herbrand interpretations.

DEFINITION
A **Herbrand interpretation** is an interpretation with domain the set of terms which gives the constants and functors their free interpretation. That is,
(i) each constant c is assigned itself,
(ii) each k-ary functor f is assigned the function that takes any k-tuple of terms <t1,..,tk> into the term f(t1,..,tk).

Herbrand interpretations differ only with respect to the extensions assigned to the predicates. Moreover, the extension assigned to a predicate R can be specified by the set of atoms

{R(t1,..,tn) : <t1,...,tn> is in the relation assigned to R}.

We can equate the set of all Herbrand interpretations with a set of all sets of atoms.

Herbrand representatives

The set of Herbrand interpretations is of special importance because they are the only interpretations that we need consider when determining the correctness of an answer substitution. In this matter, they represent the set of all interpretations. First, we define a mapping H which takes an arbitrary interpretation into its Herbrand representative.

DEFINITION
The function **H** from interpretations to Herbrand interpretations is such that

H(I) = {R(t1,..,tn) : the universal closure of R(t1,..,tn) is true for interpretation I}.

Theorem 3.1

If I is a model of some program P, then H(I) is a model of P.

Proof

Suppose that I is a model of P, but H(I) is not. We shall derive a contradiction.

If H(I) is not a model of P then there is some clause

R<-A1&..&Am

in P which is false for H(I). That is, for some assignment x1=t1,..,xk=tk of terms to the variables of the clause, the conjunction A1&..&Am is true but R is false.

The assignment of terms to the variables of the clause is a substitution s. R will be false for the assignment only if Rs is not in the atom set of H(I), and the antecedent conjunction will be true for the assignment only if each of the atoms of [A1&..&Am]s is in the atom set.

By definition of H(I), if Rs is not in the atom set then there is some assignment y1=e1,..,yi=ei of elements from the domain of I to the variables of Rs such that Rs is false for this assignment. If we extend this to an assignment y1=e1,..,yn=en to all the variables in [R<-A1&..Am]s, we have an assignment for which Rs is still false, but for which each of [A1]s,..,[Am]s is true. Each [Ai]s is true for this assignment because it is in the atom set of H(I). It must, therefore, be true for every assignment to its variables.

Now,

R{x1/t1,..,xk/tk} is false and [A1&..&Am]{x1/t1,..,xk/tk} is true

for the assigment yi=ei, $1 \leq i \leq n$, only if,

R is false and A1&..&Am is true

for the assignment xi=denotation of ti, $1 \leq i \leq k$.

In which case, the clause B<-A1&..&Am is false for interpretation I. This contradicts the assumption that I is a model of the program.

Theorem 3.2

The universal closure of Cs is true for an interpretation I if and only if it is true for interpretation H(I).

Proof

<= Assume that, for H(I), Cs is true for every assignment of terms to its variables.

Suppose y1,..,yn are the variables of Cs. Since Cs is true for every assignment of terms to these variables, it must be true for the assignment y1=y1,..,yn=yn. In other words, Cs must be true of the interpretation H(I). This means that each of the atoms of Cs is in the atom set of H(I). By definition of H, this is the case only if the universal closure of Cs is true for interpretation I.

=> We leave the proof of this implication for the interested reader. It is just as straightforward.

Equivalent interpretations

The function H induces an equivalence relation on interpretations, two interpretations being in the same equivalence class if they map into the same Herbrand interpretation. Since H maps a Herbrand interpretation into itself, each equivalence class contains exactly one Herbrand interpretation which we can take as the representing element of the class.

The above theorem tells us, that with respect to the correctness requirement for answer substitutions, the interpretations of each equivalence class are equivalent.

This, together with the fact that H maps models into models, gives us the following theorem.

Theorem 3.3

An answer substitution s is correct if and only if the universal closure of Cs is true for every Herbrand model of the program.

Proof

=> If s is correct, the universal closure of Cs is true for all models, hence for all Herbrand models.

<= Let M be any model. Theorem 3.1 tells us that its Herbrand representative H(M) is also a model. By assumption, the universal closure of Cs is true for model H(M). Hence, by Theorem 3.2 it is also true for the arbitrary model M.

3.3 Fixpoint semantics

Following van Emdem and Kowalski[1976] we can recast the model theory semantics in the Scott lattice theory framework [Scott 1970]. To do this, we must interpret each logic program P as an equation

$$x = P(x),$$

where **P** is a continuous function over some complete lattice of the possible denotations of P. The least fixpoint of **P**, i.e. the least solution of the above equation, is taken as the denotation of P. We give a brief summary of the key concepts of the fixpoint approach.

DEFINITIONS
(1) A **complete lattice** is a set S over which there is a reflexive, antisymmetric, transitive order relation, $\leq$. For each subset X of S there is a least upper bound, lub(X), in S.
(2) A function **P** over a complete lattice S is **continuous** if for every directed subset X of S

$$P(\text{lub}(X)) = \text{lub}\{P(x) : x \text{ in } X\}$$

(3) A **directed** set is a set which contains an upper bound for each of its finite subsets.

The fact that a complete lattice contains a lub for every subset implies that there is a top element, T, and a bottom element, $\perp$. It also implies that every subset X has a greatest lower bound, glb(X), in S.

That **P** is continuous implies that it is monotonic, i.e. that

$$x \leq y \text{ implies } P(x) \leq P(y).$$

Hence, by Tarski's fixpoint theorem for a monotonic function over a complete lattice [Tarski 1955], the least fixpoint of **P** exists, and is

(A) $\text{glb}\{x : P(x) \leq x\}$.

There is a second identification of the least fixpoint. By a generalisation of the Kleene recursion theorem[Kleene 1952], it is

(B) $\text{lub}\{P^i(\perp) : i \geq 0\}$.

For a proof, see [Stoy 1977 p.112]. It appeals to the continuity property of **P**.

Lattice of Herbrand interpretations

Given a logic program P we can use it to compute a set of answer substitutions for every goal clause of the form <-R(x1,..,xk), R a k-ary predicate. Each set of answer bindings defines a Herbrand extension for R, and by computing a Herbrand extension for each predicate we compute a Herbrand interpretation. So, as the set S of candidate denotations for the program we take the set of all Herbrand interpretations.

Under the partial order relation

$I \leq I'$ iff the atom set of I is included in that of I',

they are distributed over a complete lattice S. The least upper bound of any subset X of S is the atom set which is the union of the Herbrand interpretations in X. The greatest lower bound of X is the intersection of its Herbrand interpretations. The top element of the lattice is the Herbrand interpretation comprising the set of all atoms, the bottom element is the Herbrand interpretation with the empty set of atoms.

We re-interpret the program P as the equation

$$I = \mathbf{P}(I)$$

where **P** is the function

P:I -> {B' : B'<-A'1&..&A'm is an instance of a clause in P such that A'1,...,A'm are all in I}.

We shall call **P** the immediate consequence function for the program. If I defines the extension of each relation accessed by the procedure calls of a program clause, then **P**(I) is the interpretation that defines the extension of each relation that can be computed by the program. This accords with the conventions of the fixpoint approach, and is consistent with the re-interpretation of the program as the equation I=**P**(I).

Theorem 3.4

P is continuous.

Proof

Let X be a directed set of Herbrand interpretations. To prove that **P** is continuous we must show that an atom B' is in **P**(lub(X)) if and only if it is in lub{**P**(I):I in X}.

B' is in **P**(lub(X))

iff B'<-A'1&..&A'm is an instance of a clause in P such that A'1,..,A'm are in lub(X) (definition of **P**)

iff B'<-A'1&..&A'm is an instance of a clause in P such that A'1,..,A'm are in interpetations I1,..,Im of the set X (lub(X) is the union of the I in X)

iff B'<-A'1&..&A'm is an instance of a clause in P such that A'1,..,A'm are in some I in X (the if-half is trivial, for the only-if half I is the lub of I1,..,Im. Since X is directed, and I is the lub of a finite subset of X, I must be in X.)

iff B' is in **P**(I) for some I in X (definition of **P**)

iff B' is in lub{**P**(I): I in X} (definition of lub).

The lattice S contains every Herbrand interpretation. For the model theory semantics only those interpretations which are models are of interest. The following theorem characterises the Herbrand models in terms of the function **P**.

Theorem 3.5

A Herbrand interpretation I is a model of a program P iff $\mathbf{P}(I) \leq I$.

Proof

A Herbrand interpretation I is a model of P

iff for each clause B<-A1&..&Am in P, and for every substitution s for its variables, whenever [A1&..&Am]s is true Bs is true

iff for every substitution s, whenever each of [A1]s,...,[Am]s are in I, Bs is in I

iff I contains all the atoms in **P**(I)

iff $I \geq \mathbf{P}(I)$.

Denotation of a program

DEFINITION
The least fixpoint of **P** is the **fixpoint denotation** of the program P.

Theorem 3.6

The fixpoint denotation of a program P is the Herbrand model which is the intersection of all the Herbrand models of P.

Proof

The fixpoint denotation of the program is an interpretation I such that $I \geq \mathbf{P}(I)$. By Theorem 3.5, it is a model of the program.

To prove that it is the intersection of all the Herbrand models we use the Tarski identification of the least fixpoint. It is

glb({I:$I \geq \mathbf{P}(I)$}) = intersection of {I:I is a Herbrand model of P}.

Expressed more intuitively, the fixpoint denotation of the program is the Herbrand interpretation that assigns the least extension to each predicate compatible with the interpretation's being a model of the program.

Correct answers

The fixpoint denotation of a program P is a single Herbrand model. An answer substitution s must be deemed correct if the universal closure of Cs is true for this particular model. This gives us an alternative

definition of correctness.

DEFINITIONS
An answer substitution s is **fixpoint correct** for a goal clause C and program P if the universal closure of Cs is true of the fixpoint denotation of P.

The following theorem tells us that this is exactly the model theoretic characterisation of a correct answer recast in the fixed point framework.

Theorem 3.7

An answer substitution s is fixpoint correct iff it is correct

Proof

s is fixpoint correct

iff the universal closure of Cs is true for the fixpoint denotation of P

iff it is true for the Herbrand model which is the intersection of all the Herbrand models of P (Theorem 3.6)

iff it is true for every Herbrand model of P

iff s is correct (Theorem 3.3).

Because of this equivalence of the model theoretic and fixpoint definition of correctness we can use either as our non-procedural semantics for answer substitutions. From now on we shall generally appeal to the fixpoint semantics. Note that we have yet to make use of the identification

$$\text{lub}\{\mathbf{P}^i(\underline{\perp}): i \geq 0\}$$

of the least fixpoint of **P**. We shall use this in proving the equivalence of the procedural and non-procedural semantics, the usual use for the Kleene identification of the least fixpoint.

3.4 Relation to the procedural semantics

The key intermediary in our proof of the equivalence of the procedural and non-procedural semantics is the concept of a complete proof tree as defined in Chapter 1. Remember that a complete proof tree is the object constructed by a successful evaluation.

First, we shall prove that a correct answer substitution is the answer substitution 'displayed' by the goal clause atoms of a complete proof tree. We shall prove that a substitution s is correct for C and P if and only if the atoms of Cs are the goal clauses atoms of a finite complete proof tree for C and P. From this it follows immediately that every computed answer is correct. However, it does not follow that every correct answer can be computed, for the finite proof tree that

displays s may not be a tree that can be constructed by some evaluation.

To prove that every correct answer answer can be computed, we show that the set of finite proof trees are the set of substitution instances of the R-constructed proof trees, R any computation rule. It follows from this that every correct answer is a substitution instance of an R-computable answer.

Theorem 3.8

An answer substitution s is correct if and only if the atoms of Cs are the goal clause atoms of a finite complete proof tree for C and P.

Proof

We reduce the proof of the theorem to a definition and three lemmas.

DEFINITION
An atom B' is an n-level consequence of a program P if B' is in $P^n(\perp)$ for some $n \geq 0$.

Remember that P is the immediate consequence function associated with the program, and $\perp$ is the empty set of atoms.

The 1-level consequences are all those atoms that are substitution instances of the assertions in P.

The n+1 level consequences are the n-level consequences together with the set of atoms

{B': B'<-A'1&..&A'm is an instance of a clause in P
and A'1,..,A'm are n-level consequences}.

The following lemma can be established by a simple induction on n.

Lemma 1

An atom is an n-level consequence iff all its substitution instances are n-level consequences.

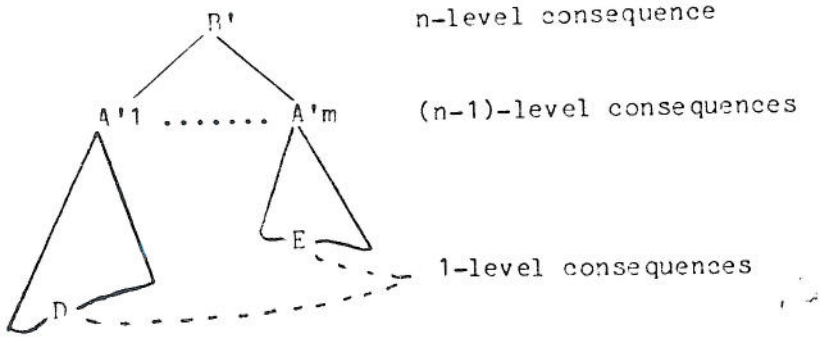
Proof tree display

We can display the derivation of an atom B' in $P^n(\perp)$ as a tree of height at most n. Such a tree is depicted in Fig. 3.1.

By making B' the single descendent of an unlabelled root node the tree becomes a complete proof tree for the goal clause <-B' and the program P. Since it is of finite height, by König's lemma it has a finite number of nodes. Hence:

Lemma 2

An atom B' is an n-level consequence of P iff there is a finite complete proof tree for <-B' and P.



Derivation of n-level consequence B'

Fig. 3.1

We know that a substitution s is correct iff the universal closure of Cs is true for the fixpoint denotation of P, iff every substitution instance of each of its atoms is in the atom set of the least fixpoint of **P**.

The following lemma tells us that each instance of an atom B is in the least fixpoint of P if and only if B is an n-level consequence for some n. With lemma 2 it implies the theorem. This is because a complete proof tree for Cs comprises separate complete proof trees for each of its atoms joined to a common root.

Lemma 3

Each instance of an atom B is in the least fixpoint of **P** iff it is an n-level consequence of P for some n.

Proof

An instance of B is in the least fixpoint of **P**

iff it is in $\text{lub}\{\mathbf{P}^n(\underline{\perp})\}$ (identity (P) for the least fixpoint)

iff it is in $\mathbf{P}^n(\underline{\perp})$ for some n (definition of lub)

iff B is an n-level consequence of P (lemma 1).

Theorem 3.8 has the following immediate corollary:

Corollary to Theorem 3.8

Every R-computable answer substitution is correct

Proof

A successful evaluation using computation rule R constructs a complete proof tree. The answer of the evaluation is the substitution that takes the goal clause C into the goal clause atoms of this tree. By the theorem, the substitution is correct.

In resolution terms, the above corollary tells us that LUSH resolution is sound.

We cannot prove exactly the converse of the above corollary despite the fact that the theorem tells us that every correct substitution is 'displayed' on some finite complete proof tree. This is because not every finite proof tree can be constructed. However, what is true, is that every finite complete proof tree is a substitution instance of a constructed tree.

Theorem 3.9

If T' is a finite complete proof tree then, for any computation rule R, there is an R-constructed proof tree T such that T'=Ts for some substitution s.

Proof

A finite complete proof tree T' for a goal clause <-B1&..&Bn is a tree as depicted in Fig. 3.2(b). The initial proof tree for this goal clause, Fig. 3.2(a), is a constructed tree. We can prove the theorem by showing that, using any computation rule R, the initial proof tree can be extended into a proof tree T that maps onto T' under some substitution s.

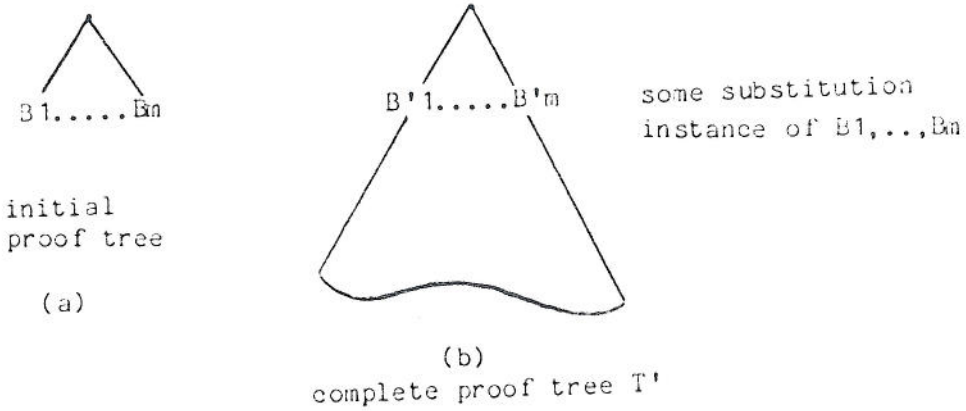
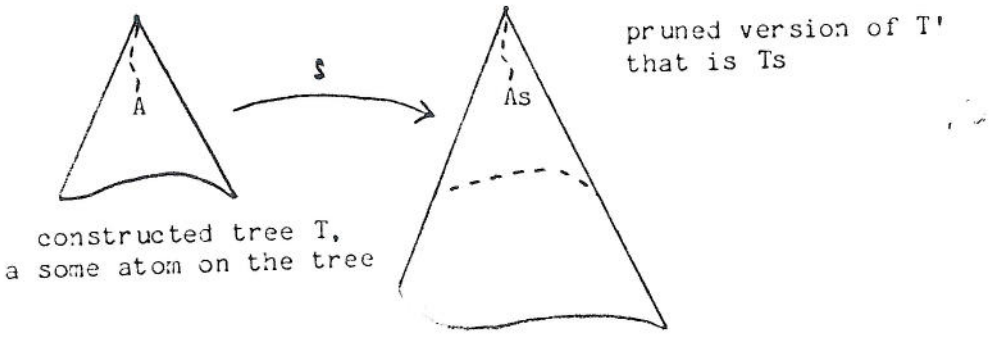


Fig. 3.2

The initial proof tree is just a special case of a constructed proof tree T that maps <u>into</u> T' as depicted in Fig. 3.3. That is, it is a



Mapping of T <u>into</u> T'

Fig. 3.3

constructed tree such that some substitution, s maps it into a pruned version of T'. Let us measure the difference between T and T' by:

number of extra nodes on T' + number of leaf nodes on T' . not marked as terminal on T

By an induction on the size of this difference, we prove that every constructed tree T that maps <u>into</u> T' can be extended into a complete proof tree that maps <u>onto</u> T'.

<u>Basis</u>

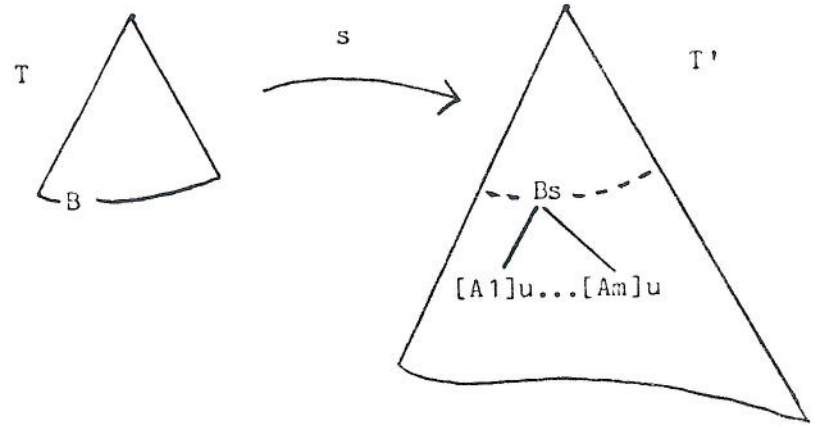
When the difference is zero T is already a constructed complete proof tree that maps <u>onto</u> T'.

<u>Induction step</u>

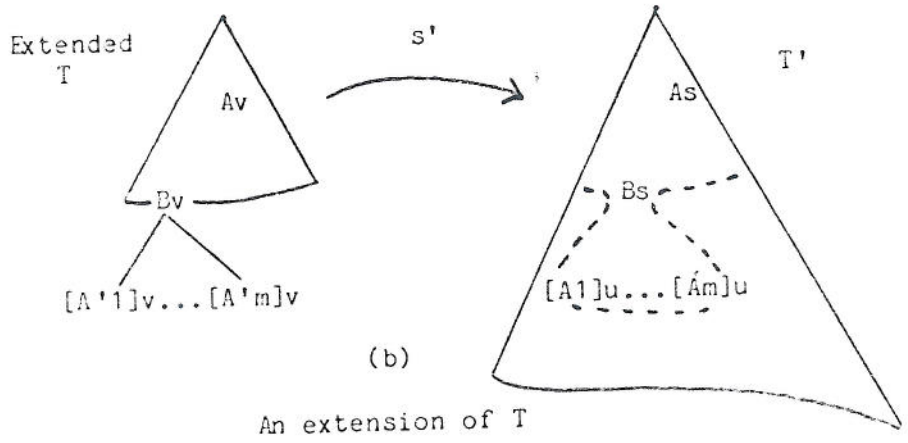
Let the difference between T and T' be K+1. Assume that the extension can be achieved for all T for which the difference between T and T' is K or less.

In extending the tree T the computation rule R will select an leaf atom B of T not marked as terminal. We know that Ts is a pruned version of T'. As depicted in Fig. 3.4(a), there must be an atom Bs on T' that corresponds to the selected atom B.

Since T' is a complete proof tree it must be the case that Bs is the



(a)



(b)

An extension of T

Fig. 3.4

consequent atom of a substitution instance

$$[B'\leftarrow A1\&..\&Am]u,\ m\geq 0$$

of some clause in the program P. The atoms [A1]u,..,[Am]u will be the immediate descendants of Bs (which is identical to B'u) on T'. Let us call

$$B'\leftarrow A1\&..\&Am$$

the <u>validating clause</u> for Bs. A variant

$$B''\leftarrow A'1\&..\&A'm$$

of this clause can be used to extend the partially constructed tree T.

Since Bs=B'u, B and the variant B" of B' will be unifiable. Suppose that {x1/y1,..,xk/yk} is the change of variable substitution that takes B' into B". Since B and B" have no variables in common, s union {y1/[x1]u,..,yk/[xk]u} will be a set of bindings for distinct variables.

Applied to B, it will give us Bs. Applied to B", it will give us B'u. Hence, this substitution is a unifier of B and B".

The unification of B and B" will return a most general unifier v. T will then be extended by adding A'1,..,A'm as immediate descendants of B, or, if m=0, by marking B as terminal. The substitution v is then applied to every node on the extended tree (see Fig. 3.4(b)).

The difference between the extended T and T' will be k or less. To apply the induction hypothesis, we must prove that the extended T also maps into T' under some substitution s'.

We know that s union {y1/[x1]u,..,yk/[xk]u} unifies B and B". Since v is a most general unifier of B and B", there is a substitution s' such that

v*s'=s union {y1/[x1]u,..,yk/[xk]u}

s' is the substitution that will take the extended tree into T'.

If s' is applied to each of the new nodes, which are labelled by the antecedent atoms of

[B'<-A1&..Am]{x1/y1,..,x1/y1}*v,

it will give us the antecedent atoms of

[B'<-A1&..&Am]{x1/y1,..,xk/yk}*{y1/[x1]u,..,yk/[xk]u}.

These are the atoms [A1]u,..,[Am]u on T'. Applied to Bv, and any other atom label Av of one of the old nodes, it will produce the atoms Bs and As respectively. Hence, s' maps every atom of the new tree into the corresponding atom on T' as required.

Corollary to Theorem 3.9

For each correct answer substitution s' for a goal clause C there is an R-computable answer substitution s such that Cs'=Cs*s" for some substitution s".

Proof

Since s' is correct, Theorem 3.8 tells us that the atoms of Cs' are the goal clause atoms of some complete proof tree T'. By the above theorem, we know that there is a constructable proof tree T that maps onto T'. The goal clause atoms of T will be the atoms of Cs, where s is the answer computed by the successful evaluation that constructs T. The substitution s" that maps T onto T' will map Cs onto Cs'. Hence, Cs'=Cs*s" as required.

In resolution terms, the above corollary tells us that LUSH resolution is complete. It is a slight generalisation of the normal completeness result which only guarantees the existence of the computable substitution s when the substitution s' has binding terms without variables. Completeness for LUSH resolution was first proved by Hill[1974]. The above proof is somewhat different from the one he gave.

3.5 Independence of the computation rule

Each answer substitution denotes a relation over the Herbrand universe. This is the set of tuples of terms that can be obtained by instantiating the tuple of binding terms of the substitution. Let us call the union of the Herbrand relations denoted by the set of R-computable answers for a goal clauses C, the R-computable extension of C. The following theorem tells us that this is always the extension given to C by the Herbrand interpretation that is the fixpoint denotation of P.

Theorem 3.10

For any computation rule R, the R-computable extension of a goal clause C is the relation assigned to C by the fixpoint denotation of the program.

Proof

We show that each extension includes the other.

The relation assigned to C by the fixpoint denotation of the program is

$$\{\langle t1,..,tk\rangle: \text{each atom of } C\{x1/t1,..,xk/tk\} \text{ is in the fixpoint interpretation}\}$$

$$= \{\langle t1,..,tk\rangle: \{x1/t1,..,xk/tk\} \text{ is a correct answer}\}.$$

That is, a tuple $\langle t1,..,tk\rangle$ of binding terms is in this relation only if it is the tuple of binding terms of a correct answer. By the completeness corollary of Theorem 3.9, $\langle t1,..,tk\rangle$ is in the Herbrand extension of an R-computable answer. It follows that the relation assigned to C by the fixpoint interpretation is included in the Herbrand extension of the set of R-computable answers.

By the soundness corollary of Theorem 3.8, each R-computable answer is correct. By the fixpoint definition of correctness, the Herbrand relation denoted by the answer is included in the extension of the relation assigned to C by the fixpoint denotation of the program. So, the Herbrand extension of the set of R-computable answers is included in the extension of this goal clause relation.

Corollary

The computed extension of a goal clause is independent of the computation rule.

The above corollary is our first result concerning the independence of the computation rule. There is a stronger result. We can prove that the different sets of computable answers not only denote the same Herbrand relation, but that they contain essentially the same substitutions.

Let us look again at the proof of Theorem 3.9. In the induction step, where we extended the proof tree T at the selected atom B, we used a variant of the validating clause of the corresponding node on T'. The particular atom that appeared on this node was not relevant. We can replace T' by any other tree T" that has the same validating clause associated with each node. The tree constructed by the inductive proof would be unchanged.

DEFINITION
The proof skeleton **T** of a proof tree T is a structurally identical tree in which each node is labelled by the validating program clause of the corresponding node on T.

We leave the reader to check that in the construction of T implicit in the induction of Theorem 3.9:

(1) The sequence of extension steps that will construct T from the initial proof tree is uniquely determined by the computation rule and the proof skeleton **T'** of T'.

(2) T' could be replaced by any other proof tree with the same skeleton.

(3) The proof skeleton of T is the proof skeleton of T'.

A proof skeleton is what is constructed by the stack implementation described in Chapter 1 if we discard the binding environments. We can summarise the above properties in the following theorem.

Theorem 3.11

For a given computation rule R and proof skeleton **T** there is exactly one successful evaluation for the goal clause C. This constructs a most general proof tree T with proof skeleton **T**. That is, T maps onto any other proof tree with skeleton **T**.

Corollary

An answer substitution s is R-computable only if an equivalent substitution is R'-computable, where R' is any other computation rule.

Proof

The computation of s using rule R will construct a complete proof tree T with the atoms of Cs as the goal clause atoms. Since there is only one successful evaluation corresponding to the rule R and the proof skeleton **T** of T, T must be a most general proof tree for skeleton **T**.

For proof skeleton **T** and rule R' there is also exactly one successful evaluation of C. This will compute an answer s' and in so doing will construct a proof tree T', with the atoms of Cs' as its goal clause atoms. This is also a most general proof tree for skeleton **T**.

Since T and T' are each most general proof trees for skeleton **T**, there is a substitution that maps the atoms of Cs into the atoms of Cs', and vice versa. Cs and Cs' must therefore be variants. If s and s' are the

substitutions

$$s=\{x1/t1,..,xk/tk\},\ s'=\{x1/t'1,..,xk/t'k\},$$

then, since x1,..,xk are all the variables of C, the term tuples <t1,..,tk>, <t'1,..,t'k> must also be variants. They therefore denote the same relation for every interpretation.

The above corollary is the justification for our claim that the family of algorithms implicit in a given logic program are equivalent. It tells us that each algorithm must compute the same set of answers (modulo equivalence of substitutions).