CHAPTER 11

If-and-only-if

In classical logic, definitions are expressed by means of "if-and-only-if" (abbreviated "iff"). For example:

G* Grandparent(x,y) <-> ∃z [Parent(x,z) & Parent(z,y)]

Horn clause programs and databases, however, express only the "if-halves" of iff-definitions:

G Grandparent(x,y) <- Parent(x,z), Parent(z,y)

We have managed to avoid the full iff-form of definitions because the if-halves alone are adequate for deriving all positive instances of the relations. All variable-free assertions of the form

Grandparent(s,t) <-

which are implied by G* are already implied by G. It is not possible to compute more factorials with the iff-definition

F* Fact(x,y) <-> [x=0 & y=1] ∨
∃x'∃y'[x=x'+1 & Fact(x',y') & y=x*y']

than with the if-half alone:

F1 Fact(x,y) <- x=0, y=1
F2 Fact(x,y) <- x=x'+1, Fact(x',y'), y=x*y'

However, as we shall see in the next section, the full iff-form of definitions is needed for proving properties of programs. It is also needed in databases for answering queries involving universal quantifiers and negation.

In the informal use of natural language, the if-form of definitions is often employed even when the iff-definition is intended. This gives rise to the problem of distinguishing when the missing only-if half of the definition is intended and when it is not.

We shall argue that the problem is complicated by the fact that the only-if halves of definitions are ambiguous.

A only if B

can be interpreted in the object language

B <- A

or in the meta-language

"A <- B" expresses the only condition under which A holds.

Consequently, proofs which need to appeal to the only-if halves can be carried out either in the object language or in the meta-language. Despite this difference, however, the structure of the proofs is remarkably similar in both cases.

The need for the only-if halves of definitions

The only-if halves of definitions are needed for proving program properties and for verifying database integrity constraints. Consider, for example, the Horn clause program Fl-2 for computing factorials. It is a property of the program that

The only factorial of 0 is 1,
i.e. y=1 <- Fact(0,y).

To prove the property, however, requires the only-if half of the definition of factorial as well as the property of equality that

<- 0 = u+1.

The only-if halves of definitions are also needed for answering queries in logic databases. Consider, for example, the iff-definitions of the Teaches and Professor relations:

T* Teaches(x,y) <-> [x=A & y=104] V
[x=A & y=301] V
[x=B & y=221] V
[x=C & y=105] V
[x=C & y=201] V

P* Professor(x) <-> x=A V x=B

Given, in addition, the clauses

Isa(104,programming) <-
Isa(221,programming) <- ,

the query

Do all professors teach programming?

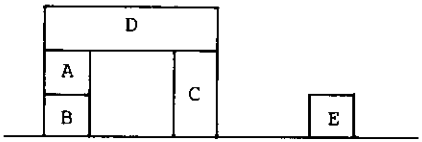
$\forall x \exists y$[Professor(x) -> Teaches(x,y) & Isa(y,programming)]?

can be answered positively. To answer the query, however, requires the only-if half of the definition of the Professor relation. The object language and meta-language proofs of the query are presented and compared later in the chapter.

Terms versus relations as data structures

The relationship between iff-definitions and their if-halves bears upon the relationship between the use of terms and the use of relations as data structures in logic programs. The use of terms in Horn clause programs gives some of the power of the use of relations defined by means of iff.

Consider, for example, the data depicted in the following scene:



Restricted to the use of Horn clauses, the On and Clear relations have to be defined independently:

```
On(A,B) <-              Clear(D) <-
On(D,A) <-              Clear(E) <-
On(D,C) <-
```

The connection between the two relations can be expressed only by means of an integrity constraint.

```
<- On(x,y), Clear(y)
```

By using iff-definitions, however, the Clear relation can be defined in terms of the On relation.

```
Clear(y) <->  ¬∃x On(x,y)
On(x,y)  <->  [x=A & y=B] ∨
              [x=D & y=A] ∨
              [x=D & y=C] ∨
```

Notice, however, that in this formulation and the next everything is clear except A, B and C. The Clear relation can be restricted, if necessary, by adding an extra condition to the definition

```
Clear(y) <-> Block(y) & ¬∃x On(x,y)
```

and appropriately defining the new predicate Block.

Iff-definitions cannot usually be expressed by means of Horn clauses. However, some of the power of iff-definitions can be captured with Horn clauses by using terms instead of relations as data structures. If the data concerning the position of objects in the scene is collected in a single term, then the Clear relation can be defined in terms of the data about the scene. Here "On" is a predicate symbol, but "on" is a function symbol.

```
Scene(on(A,B).on(D,A).on(D,C).nil) <-
On(x,y) <- Scene(z), Member(on(x,y),z,T)
Clear(y)<- Scene(z), Member(on(x,y),z,F)
```

```
Member(x, x.y, T) <-
Member(x, nil, F) <-
Member(x, u.v, w) <- Diff(x,u), Member(x,v,w)
```

The term representation of the data is significantly less natural than the relational representation. However, both the iff-definition and its simulation by means of terms have several advantages over the simple, Horn clause if-half of the definition. Many properties of the scene, such as the number of objects it contains, can be determined from both the iff-definition and the term representation but cannot be determined from the simple if-half of the definition. Moreover, any change in the position of objects (either by altering the iff-definition of the On relation or by altering the assertion which describes the scene) automatically implies the appropriate modification of the Clear relation. However, if the two relations are defined independently, then alteration of the scene is more difficult. Both the On and Clear relations have to be changed explicitly and the new relationship between them needs to be checked against the integrity constraint.

The unstated only-if-assumption

The statement of only the if-halves of definitions is common in natural language, even when the full iff-definition is intended. Even logicians, who normally insist on the explicit statement of all assumptions, tolerate the unstated only-if assumption in the case of recursive definitions. It is common for a logician to state only the if-half of the definition of natural number, for example:

N1 0 is a natural number.
N2 If x is a natural number then x+1 is a natural number.

even when he intends the only-if half

N3 The only natural numbers are
those defined by statements N1-2.

as well.

Natural language, however, carries the unstated only-if assumption to the extreme. The classical fallacy of logic is probably an example of this. Suppose, for instance,

M1 Mortal(x) <- Human(x).

If we now assert

M2 Mortal(Bob) <-

then we may be tempted to conclude

M3 Human(Bob) <- .

But M3, although it may well be true, is not a logical consequence of the explicitly stated assumptions M1-2. The fallacy would disappear, however, if we could appeal to unstated assumptions - if we could assume, in

particular, that the full iff-definition

M* Mortal(x) <-> Human(x)

was intended when only the if-half was stated explicitly.

Comparing the two examples, the if-half of the definition of number and the incomplete characterisation M1 of mortality, we are faced with the dilemma of distinguishing when the unstated only-if assumption is justified and when it is not. The same dilemma arises in the field of databases where the problem is to decide whether the definition of the data has already been closed or whether it is still open. The problem has been investigated by Reiter [1978] who calls the assumption that the database contains all the information there is to know the closed world assumption and the assumption that it may not, the open world assumption. Our proposal is to identify the closed world assumption with the assumption that the missing only-if halves of definitions are intended and to identify the open world assumption with the assumption that they are not.

The problem of distinguishing between intended and unintended assumptions disappears, of course, if all intentions are made explicit. Explicit statement of intentions, moreover, makes it easy to mix closed and open world assumptions in the same database, applying different assumptions to different relations or even to different instances of the same relation. We might decide, for example, to close the instances of the Teaches relation which describe the courses taught by Bob, but to leave open the ones taught by John.

T1 Teaches(Bob,x) <-> x=304 V x=323 V x=1.4
T2 Teaches(John,212) <-
T3 Teaches(John,1.13) <-

It is curious that natural language should be so careless about specifying whether or not only-if assumptions are intended. This may be a consequence, in part, of the awkwardness of the iff syntax. In order to close the definition of the courses taught by John, after adding the assertion

T4 Teaches(John,103) <-

for example, it is necessary either to replace T2-4 by

T* Teaches(John,x) <-> x=212 V x=1.13 V x=103

or to add to T2-4 the explicit only-if half of the definition

T5 Teaches(John,x) -> x=212 V x=1.13 V x=103.

A more convenient syntax might be one which leaves T2-4 alone and states that

T5* all instances of Teaches(John,x) are defined by T2-4.

Ambiguity of only-if

Our discussion of the relationship between iff-definitions and their if-halves has been simplified by ignoring the ambiguity of the expression

A only if B.

In some cases we have interpreted it as a statement in the object language

B <- A.

In other cases we have interpreted it in the meta-language

"A <- B" expresses the only condition under which A holds.

The only-if half of the definition of natural number, which was previously expressed in the meta-language, can also be expressed in the object language.

$$Numb(x) \rightarrow x = 0 \vee \exists x'[x = x'+1 \ \& \ Numb(x')]$$

Whether the expression "A only if B" is interpreted in the object language or the meta-language, it has similar properties. For example, in both cases the conclusion

B <-

is a consequence of the assumptions

A only if B
A <- .

If "only-if" is interpreted in the object language, the conclusion follows by one step of bottom-up reasoning. If it is interpreted in the meta-language, it follows by reasoning about proofs:

If the only way of proving A is by proving B, and
A <- can be proved,
then B <- can be proved as well.

This example illustrates a general phenomenon: The two interpretations of "only-if" justify similar conclusions in different, but structurally similar, ways.

Object language and meta-language solutions

The problem of showing that all professors teach programming

Q $\quad \forall x \exists y[Professor(x) \rightarrow Teaches(x,y) \ \& \ Isa(y,programming)]$

can be solved whether the only-if half of the definition P* of the Professor relation is expressed in the object language or in the meta-language.

Suppose the only-if half of P* is expressed as a non-Horn clause

x = A, x = B <- Professor(x)

in the object language. The query itself reduces to two clauses:

Q1 Professor(☺) <-
Q2 <- Teaches(☺,y), Isa(y,programming)

Bottom-up reasoning from the assertion Q1 derives the non-Horn clause

☺ = A, ☺ = B <- .

The two goals in Q2 can now be solved by case analysis. In the case ☺ = A, the first goal in Q2 is solved by

Teaches(x,y) <- x = A, y = 104
x = x <-

and the second goal by

Isa(104,programming) <- .

In the second case ☺ = B, the first goal is solved by

Teaches(x,y) <- x = B, y = 221
x = x <-

and the second goal by

Isa(221,programming) <- .

Suppose, on the other hand, that the only-if half of P* is expressed in the meta-language:

P1 Professor(x) <- x = A
P2 Professor(x) <- x = B
P3 P1 and P2 express the only conditions under which an individual is a member of the Professor relation.

To solve the problem, the query Q needs to be expressed in the meta-language as well.

Show that for every x which solves the goal
Q1* <- Professor(x)
there is a y which solves the goals
Q2* <- Teaches(x,y), Isa(y,programming).

Top-down reasoning from the goal Q1* derives only two solutions

x = A and x = B.

In the case x = A, the two goals in Q2* are solved by y = 104 using the clauses

```
Teaches(x,y) <- x = A, y = 104
x = x <-
Isa(104,programming) <- .
```

In the case x = B, they are solved by y = 221 using the clauses

```
Teaches(x,y) <- x = B, y = 221
x = x <-
Isa(221,programming) <- .
```

Notice that the object language and meta-language proofs have similar structure. In the meta-language proof, however, equality relates variables to the terms to which they are bound in the components of matching substitutions. In the object language proof, equality relates different names for the same individual. Thus the equality symbol used for expressing the only-if halves of definitions satisfies the axioms E1-3 of Chapter 2, page 43. In the general case, these axioms are extremely redundant. In this case, however, they are not even necessary.

Object language and meta-language interpretations of negation

The only-if halves of definitions are necessary to show that a negative condition

```
<- not~P
```

holds. Depending on the interpretation of "only-if", the proof can be carried out either at the object level or at the meta-level. Clark [1978] has shown that for every meta-language proof of not~P obtained by a Horn clause theorem-prover augmented with negation proved by failure, there exists a structurally similar object language proof of not~P.

Consider the problem of showing that D is clear

```
<- Clear(D)
```

given the if-halves of the definitions of the On and Clear relations:

```
On1    On(A,B)  <-
On2    On(D,A)  <-
On3    On(D,C)  <-
On4    Clear(y) <- not¬∃x On(x,y)
```

In addition, the only-if half of the definition of the On relation is necessary for a solution. However, the if-half of the definition of the Clear relation is sufficient.

Suppose first that the only-if half of the definition is expressed in the object language:

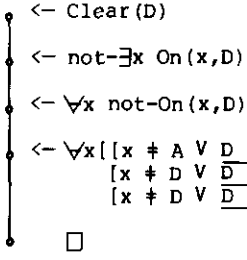
```
On5    On(x,y) -> [x = A & y = B] V
                  [x = D & y = A] V
                  [x = D & y = C] V
```

The sentence is more natural in the standard form than in the clausal

form. It is also more natural to carry out the proof using standard form. Moreover, the standard form proof is structurally similar to the meta-language proof, whereas the clausal form proof is not. It will be useful to reexpress the only-if half of the definition in the equivalent form

$$\begin{aligned} \text{not-On}(x,y) \leftarrow\ & [x \neq A \vee y \neq B] \ \&\\ & [x \neq D \vee y \neq A] \ \&\\ & [x \neq D \vee y \neq C] \end{aligned}$$

where $s \neq t$ is just an abbreviation for $\neg[s = t]$.



The last step of the proof verifies the three conditions by using the "negative assertions":

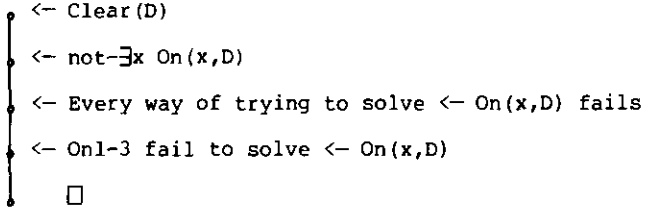
On6 $D \neq B \leftarrow$
On7 $D \neq A \leftarrow$
On8 $D \neq C \leftarrow$

The clausal form, resolution proof is left to exercise (2).

Suppose now that the only-if half of the definition is expressed in the meta-language:

> Clauses On1-3 express the only conditions
> under which the On relation holds.

The meta-level proof shows that every way of trying to solve the goal ←On(x,D) fails. The structure of the proof, however, is similar to that of the object level argument.



The last step of the proof shows that On1-3 fail to match On(x,D),

because D is different from (does not match) A, B and C. The object-level proof, however, needs to reason about equality explicitly. Clark [1978] shows that in general explicit axioms of equality are necessary at the object level in order to simulate failure of the matching algorithm at the meta-level.

Horn clauses augmented with negation interpreted as failure

The meta-language interpretation of "only-if" entails the interpretation of negation as failure:

not-P holds
if the if-halves of definitions fail to establish P.

The language of Horn clauses augmented with negation as failure provides a powerful extension of the language of Horn clauses alone. It is easy to implement, efficient to use and has much of the expressive power of the full standard form of logic. It is an important feature of all PROLOG implementations that either they provide the negation operator explicitly or else they provide means for defining it.

The expressive power of Horn clauses with negation is illustrated by the definition of subset

$$x \subseteq y \leftarrow \forall z\ [z \in x \rightarrow z \in y],$$

which can be reexpressed

$$x \subseteq y \leftarrow \text{not-}\exists z\ [z \in x,\ \text{not-}[z \in y]]$$

x is a subset of y if no z in x fails to belong to y.

The explicit existential quantifier $\exists z$ can be eliminated and the negation sign can be moved in front of atomic formulae if an auxiliary predicate Nosub(x,y), which holds when x is not a subset of y, is employed. The definition of subset becomes

$$x \subseteq y \leftarrow \text{not-Nosub}(x,y)$$
$$\text{Nosub}(x,y) \leftarrow z \in x,\ \text{not-}[z \in y].$$

x is a subset of y if it cannot be shown
that it is not a subset of y.
x is not a subset of y if there is a z in x
which fails to belong to y.

A similar transformation can be applied to the definition of Clear block:

Clear(y) <- not-Covered(y)
Covered(y) <- On(x,y)

Clark's analysis of negation interpreted as failure assumes that negations are so transformed that they stand only in front of atomic formulae.

Clark has shown that Horn clauses with negation interpreted as failure do not have the full power of negation in the standard form of logic. The simplest example of this is the sentence

```
P <- not-P
```

which implies

```
P <-
```

in the standard form of logic, since

```
P <- not-P     is equivalent to
P, P <-        is equivalent to
P <- .
```

But the attempt to solve

```
<- P      given      P <- not-P
```

does not succeed because it goes into a loop when negation is interpreted as failure.

A more complicated infinite loop arises during the attempt to solve the goal

```
<- A      using      1)      A <- P(x)
                     2)      A <- not-P(x)
                     3)      P(x) <- P(f(x))
```

with negation interpreted as failure. Both procedures (1) and (2) introduce the procedure call

```
<- P(x)
```

which neither succeeds nor fails in finite time. But in the standard form of logic, A <- is a resolvent of (1) and (2).

These examples suggest that the deductive power of negation as failure can be increased by adding loop detection to the resources of the Horn clause problem-solver. Because of the undecidability of logic [Church 1936] however, no problem-solver can recognise all situations in which a goal is unsolvable. There is no best theorem-prover and no limit to the extent to which a problem-solver can improve its ability to detect loops and to establish negation by failure.

The recognition of failure by detecting loops in the meta-language is equivalent to using proof by induction in the object language. by adding proof by induction to the resources of the problem-solver. Proof by induction is needed, moreover, in many cases when the only-if halves of definitions are used to prove program properties.

Proof of program properties

Consider the Horn clause if-half of the definition of the Append-relation

A1 Append(nil,x,x) <-
A2 Append(x.y, z, x.y') <- Append(y,z,y').

It has the property that

Append(x,nil,x) holds for all lists x.

Proof of the property requires induction on the structure of lists. We shall present both the object level and meta-level proofs. Both proofs have similar structure. But the meta-level proof, because it is informal, is easier to present first.

Suppose that A is any list. We need to show that

A3 Append(A,nil,A) <-

can be proved using (A1) and (A2). The proof is by induction on the structure of A. If A is nil, then there is a one-step proof of (A3) using (A1) alone. If A is B.A', then by the induction hypothesis there is some n-step proof of

Append(A',nil,A') <- .

By adding an extra step to the proof, using (A2), we obtain an n+1 step proof of

Append(x.A', nil, x.A') <-

for any x and therefore a proof of (A3) in particular.

For the object level proof, it is necessary to express an induction schema for lists in the object language.

A4 F(x) <- List(x) & F(nil) & $\forall y \forall z$ [F(z) -> F(y.z)]

where F(x) is any formula containing free occurrences of only the variable x, and F(t), for any term t, is obtained by replacing all free occurrences of x in F by t. The object level proof can be carried out in clausal form; but a non-clausal proof is more natural. We negate the theorem to be proved and reason backward from the goal:

A5 List(A) <-
<- Append(A,nil,A)

By A4, letting F(x) be Append(x,nil,x):

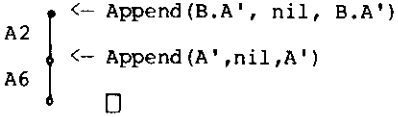
<- List(A), Append(nil,nil,nil),
$\forall y \forall z$[Append(z,nil,z) -> Append(y.z, nil, y.z)]

By A5 and A1:

<- $\forall y \forall z$[Append(z,nil,z) -> Append(y.z, nil, y.z)]

This reduces to an assertion and a subgoal:

A6 Append(A',nil,A') <-



The method of proving properties of logic programs by means of induction axioms expressed in the object language has been developed by Clark and Tarnlund [1977].

The monotonicity criticism of logical consequence

Logic has often been the subject of criticism. One of the most recent and influential of these criticisms is that formulated by Minsky [1975] concerning the monotonicity of logical consequence.

Consider again the blocks world example

On1 On(A,B) <-
On2 On(D,A) <-
On3 On(D,C) <-
Clear(y) <- not-∃x On(x,y)

supplemented by the unstated only-if half of the definition of the On relation. These assumptions imply the conclusion

Clear(D) <- .

The monotonicity of logical consequence entails that the same conclusion continues to hold no matter what new assumptions are added. In particular, if we add the new assumption

On4 On(E,D) <-

the previous conclusion that D is clear still holds, even though it is obviously inconsistent with the new information.

The critics argue that the monotonicity of logical consequence contradicts common sense. Given the new assumption On(E,D) <- common sense abandons the previous conclusion Clear(D) <- . Logic, because it requires that the conclusion continues to hold, is unacceptable as a model of human reasoning.

The argument is mistaken, in our opinion, because it oversimplifies what is involved when a new assumption is added to a logic database. We shall argue in the last chapter that, when a database becomes inconsistent, consistency needs to be restored by rejecting or suitably modifying an assumption in the database. In this example, either we reject the new information or we reject or modify the only-if half of the definition of the On relation. It is probably most natural either to replace the original only-if assumption by the new assumption that only

On1-4 define the On relation or else to abandon the only-if assumption altogether. In either case the previous conclusion Clear(D) <- no longer holds in the new database.

Logic avoids the monotonicity criticism of logical consequence, if proper account is taken of only-if assumptions and a realistic view is taken of the way in which databases change in time.

Exercises

1) Use the only-if half of the definition of factorial together with the assumption

<- ∅ = u+1

to show that the only factorial of ∅ is 1.

2) Show that

not-Append(nil, a.nil, nil)

is a consequence of the iff-definition of the Append relation. Compare the object language and meta-language proofs and identify the axioms of equality needed for the object language proof.

3) Transform assumptions On4-8 into clausal form and use resolution to show that

Clear(D)

is a consequence.

4) Show by means of resolution and factoring that

Append(A,nil,A)

is implied by the iff definition of Append together with the appropriate induction and equality axioms expresed in clausal form.

5) Using negation as failure, reformulate the definions of arch and tower given in Chapter 4 so that the problem

<- Arch(w)

has only two solutions

w = a(t(B,A), D, C) and
w = a(C, D, t(B,A))

for the scene described by A4-12.

6) Given the Horn clauses

Append(nil,x,x) <-
Appned(x.y, z, x.u) <- Append(y,z,u)
Member(x, x.y) <-
Member(x, y.z) <- Member(x,z)

show by means of induction in the meta-language that

for all x, u, v and w,
if Append(u,v,w) and Member(x,w)
then Member(x,u) or Member(x,v).

7) a) Given the assumptions

N1 $x \subseteq y$ <- not-Nosub(x,y)
N2 Nosub(x,y) <- $z \in x$, not-[$z \in y$]

$a \in A$ <-
$a \in B$ <-
$b \in B$ <-

show that $A \subseteq B$

interpreting negation as failure.

b) Let membership in the sets A and B be expressed by means of Horn clauses. Discuss the circumstances under which it can be shown that

i) $A \subseteq B$

ii) $\emptyset \subseteq B$
where there is no clause expressing membership in $\emptyset$

iii) $A \subseteq U$
given $x \in U$ <-

iv) $A \subseteq A$.