CHAPTER 3

Top-down and Bottom-up Horn Clause Proof Procedures

Introduction

The parsing problem - to show that a string of words forms a sentence according to given rules of grammar - can be represented in logic as a problem of demonstrating the inconsistency of a set of Horn clauses.

Different parsing procedures for determining that a string is a sentence correspond to different proof procedures for demonstrating inconsistency. Top-down parsing procedures correspond to goal-directed proof procedures which work backwards from the conclusion by using implications to reduce problems to subproblems. The aim is to reduce the original problem to a set of subproblems each of which has been solved. Bottom-up parsing procedures correspond to proof procedures which work forward from the initial set of assumptions, by using implications to derive conclusions from assumptions. The aim is to derive assertions which directly solve each of the initially given problems.

Top-down and bottom-up proof procedures apply to the solution of any problem. Top-down inference is the analysis of goals into subgoals; bottom-up inference is the synthesis of new information from old. In this chapter we define top-down and bottom-up inference for Horn clauses only. Later we shall extend their definition to non-Horn clauses and investigate systems which combine both directions of inference.

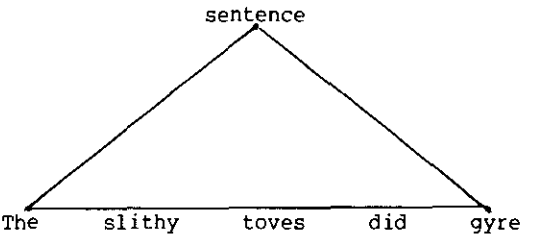
The parsing problem

The following description of the parsing problem is based on Foster's description [Foster 1970] of a formulation by Amarel.

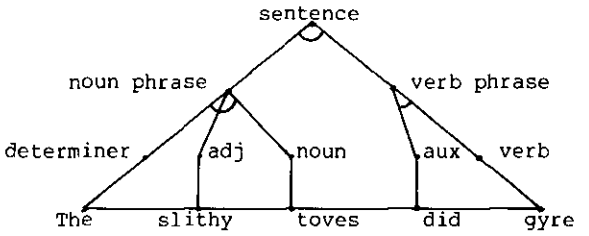
Given a grammar and a string of words such as

"The slithy toves did gyre"

the problem is to demonstrate that the string is a sentence. This can be done by filling in the triangle



with a parse tree:

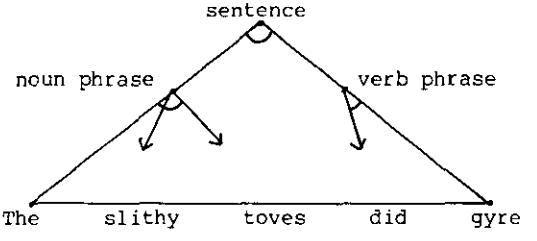


The parse tree is constructed in accordance with a grammar. In this example, the following rules of grammar have been used.

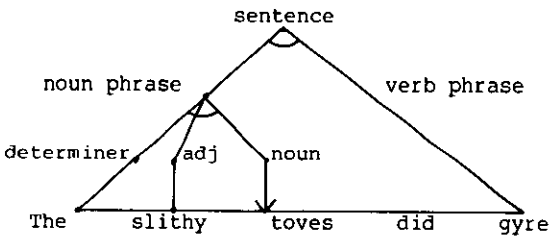
(1) A noun phrase followed by a verb phrase is a sentence.
(2) A determiner followed by an adjective followed by a noun is a noun phrase.
(3) An auxiliary followed by a verb is a verb phrase.
(4) "The" is a determiner.
(5) "slithy" is an adjective.
(6) "toves" is a noun.
(7) "did" is an auxiliary.
(8) "gyre" is a verb.

Different ways of filling in the triangle determine different parsing procedures. Top-down procedures are determined by filling in the triangle from the top downwards. Bottom-up procedures are obtained by filling in the triangle from the bottom upwards.

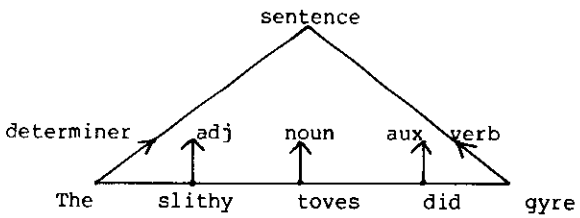
A top-down procedure might generate all branches in parallel:



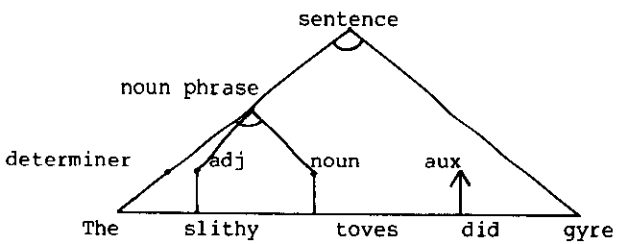
or it might generate one branch at a time, say from left to right.



Similarly, a bottom-up procedure might work on all words in the input string in parallel:



or it might work on one word at a time.

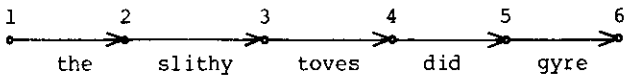


The triangle can be filled in from right to left, bi-directionally top-down and bottom-up, and even from the middle out. Every systematic method of filling in the triangle determines a parsing procedure. At this point, it is important to distinguish mainly between the top-down and bottom-up procedures.

A predicate logic representation of the parsing problem

There are many ways to represent the parsing problem in logic. The one we describe here has the property that different parsing procedures correspond to different proof procedures for the same representation.

We regard the initial string of words as a graph. A node of the graph occurs between adjacent words of the initial string and also at the beginning and end of the string. We regard words in the string as labels on the arcs connecting adjacent nodes:

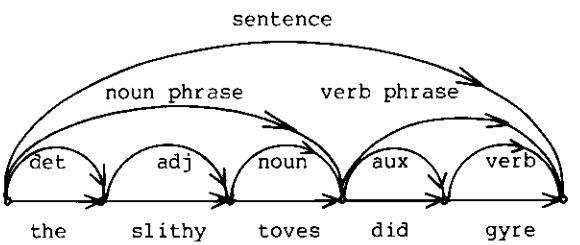


The nodes are arbitrarily named 1-6. No ordering is implied by the numbers used to name the nodes.

The rules of grammar can be regarded as statements concerning labelled graphs:

> If there is a path from node x to y labelled "the" then the path from x to y is also labelled "determiner", i.e.
>
> $$Det(x,y) \leftarrow the(x,y).$$
>
> If there is a path from x to u labelled "determiner" and a path from u to v labelled "adjective" and a path from v to y labelled "noun" then there is a path from x to y labelled "noun phrase", i.e.
>
> $$Np(x,y) \leftarrow Det(x,u), Adj(u,v), Noun(v,y).$$

A parse of the initial string of words can be regarded as a graph which is labelled according to rules of grammar and has a path, from the beginning of the string to the end, labelled "sentence":



The initial graph is represented by a set of assertions:

```
Parse 1          the(1,2)     <-
Parse 2          slithy(2,3)  <-
Parse 3          toves(3,4)   <-
Parse 4          did(4,5)     <-
Parse 5          gyre(5,6)    <-
```

The rules of grammar are represented by clauses containing variables:

```
Parse 6          Sent(x,y) <- Np(x,z), Vp(z,y)
Parse 7          Np(x,y)   <- Det(x,u), Adj(u,v), Noun(v,y)
Parse 8          Vp(x,y)   <- Aux(x,z), Verb(z,y)
Parse 9          Det(x,y)  <- the(x,y)
Parse 10         Adj(x,y)  <- slithy(x,y)
Parse 11         Noun(x,y) <- toves(x,y)
Parse 12         Aux(x,y)  <- did(x,y)
Parse 13         Verb(x,y) <- gyre(x,y)
```

These are the only rules of grammar needed to parse the original string of words. In a more realistic formulation of the problem, we have to consider the use of other rules of grammar as well. For example:

```
Parse 14         Np(x,y)   <- Det(x,z), Noun(z,y)
Parse 15         Np(x,y)   <- Noun(x,y)
Parse 16         Vp(x,y)   <- Verb(x,y)
Parse 17         Det(x,y)  <- a(x,y)
Parse 18         Adj(x,y)  <- brillig(x,y)
Parse 19         Noun(x,y) <- wabe(x,y)
Parse 20         Verb(x,y) <- gimble(x,y)
```

To show that the string of words from 1 to 6 is a sentence we show that the denial of the goal

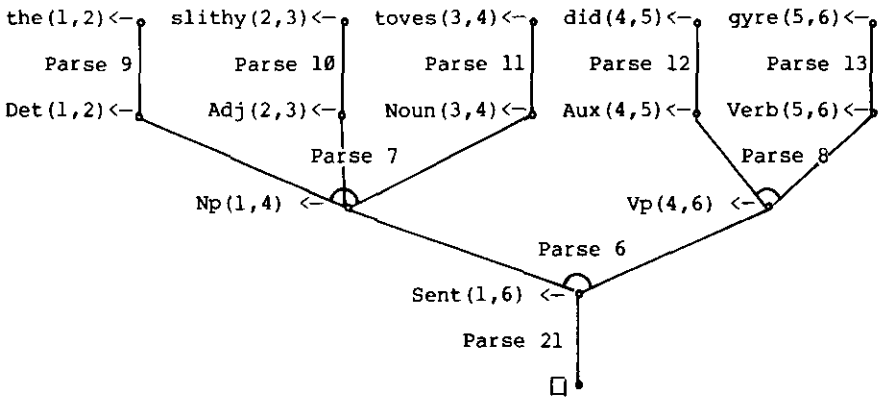
```
Parse 21         <- Sent(1,6)
```

is inconsistent with Parse 1-20.

Bottom-up inference

A bottom-up refutation begins with assertions in the input set of clauses. It uses implications to derive new assertions from old ones, and ends with the derivation of assertions which explictly contradict the denial of the goal.

A graphical representation of the bottom-up refutation of Parse 1-21 is shown below. It resembles the parse tree turned upside-down. Nodes are labelled by assertions. The implication used to derive a new assertion labels the bundle of arcs leading from the old assertions to the new one.



The assertion

Np(1,4) <-

for example, is obtained from the three assertions

Det(1,2) <-
Adj(2,3) <-
Noun(3,4) <-

by matching them with the three conditions of the clause

Np(x,y) <- Det(x,u), Adj(u,v), Noun(v,y).

Matching is accomplished by finding a most general substitution, in this case

{x=1, u=2, v=3, y=4},

which makes the assertions identical to the conditions.

In general, one step of bottom-up inference matches (in the most general possible manner) a number of assertions with the conditions of a clause and derives a new assertion. The new assertion consists of the conclusion of the clause instantiated by the matching substitution. If the clause is a denial (which has no conclusion) then the derived clause is the empty clause. A more precise definition is given at the end of the chapter.

Bottom-up inference is a generalisation of instantiation combined with the classical rule of modus ponens:

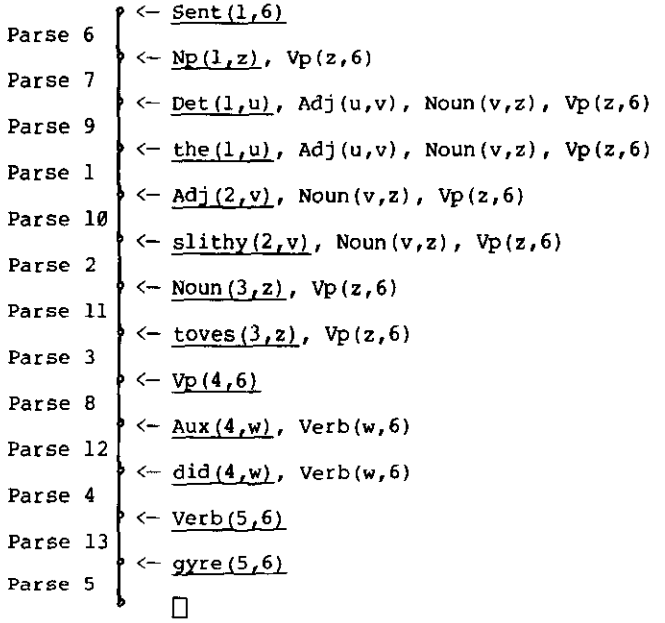
From A <- and B <- A derive B <- .

Instantiation is restricted to the minimum needed to match assertions with conditions, so that modus ponens can be applied.

Top-down inference

A top-down refutation begins with a denial in the input set of clauses. It uses implications and assertions to derive new denials from old ones and ends with the derivation of the empty clause.

A graphical representation of a top-down refutation of Parse 1-21 is given below. Nodes are labelled by denials. An arc is labelled by the input clause which is used to derive the denial at the bottom of the arc. Selected atoms are underlined.



Beginning with the initial denial

```
<- Sent(1,6)
```

top-down inference matches the condition of the denial with the conclusion of the implication

```
Sent(x,y) <- Np(x,z), Vp(z,y)
```

deriving the new denial

```
<- Np(1,z), Vp(z,6)
```

which consists of the conditions of the input clause instantiated by the matching substitution

```
{x=1, y=6}.
```

The inference step formalises the reasoning that

> if there is no sentence from 1 to 6 then there is no z such that there is a noun phrase from 1 to z followed by a verb phrase from z to 6.

The same inference step can also be interpreted from a problem-solving point of view:

> The goal of showing that there is a sentence from 1 to 6 can be solved if a z can be found such that the subgoals of showing there is a noun phrase from 1 to z and a verb phrase from z to 6 can be solved.

In the problem-solving interpretation, the original goal is reduced to two new subgoals.

In general, top-down inference involves matching a selected condition of a denial with the conclusion of an implication and deriving a new denial by replacing the selected condition by the conditions of the implication and applying the matching substitution. If the implication is an assertion, which has no conditions, then the selected condition is simply deleted and the matching substitution is applied. If, in addition, the selected condition is the only condition in the denial then the derived clause is the empty clause. In the problem-solving interpretation, a denial is interpreted as a collection of goals. Top-down inference replaces a selected goal (in the context of a collection of goals) by a set of subgoals. A precise definition of top-down inference is given at the end of the chapter, while the problem-solving interpretation is investigated in the next chapter.

Top-down inference is a generalisation of instantiation combined with *modus tollens*:

> From not-A and A <- B derive not-B.

Instantiation is restricted to the minimum needed to apply the modus tollens rule.

Different top-down refutations are determined by selecting different atoms in denials for the application of top-down inference. For example, clause Parse 8 could be applied to the denial

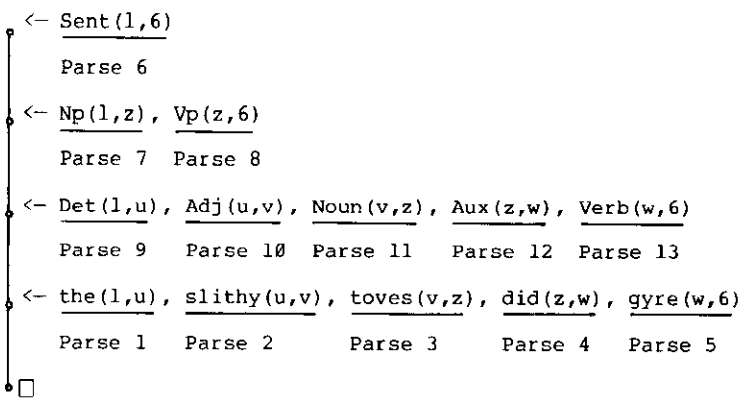
```
<- Np(1,z), Vp(z,6)
```

to derive the new denial

```
<- Np(1,z), Aux(z,u), Verb(u,6)
```

If there is a refutation for one selection of atoms then there is a refutation for any other selection.

It is also possible (as in bottom-up inference) to select all conditions in a denial simultaneously. The figure below illustrates such a top-down parallel refutation. Below each selected condition is the name of the clause used in the derivation of the next denial.



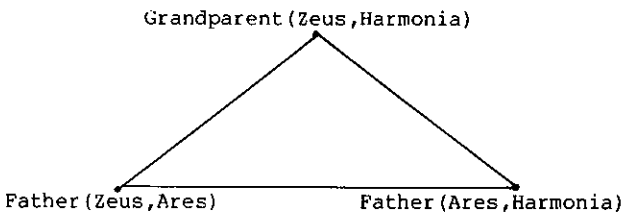
This formulation of the parsing problem was obtained by Alain Colmerauer with the author by expressing his Q-system [Colmerauer 1973] in logic. It is significant that, whereas the Q-system is a bottom-up parsing procedure, the Horn clause formulation is more abstract and can be used either top-down or bottom-up.

Although the example uses only context-free rules of grammar, it is easy to extend the representation to express context-sensitive grammars and arbitrary rewriting systems [Chomsky 1957].

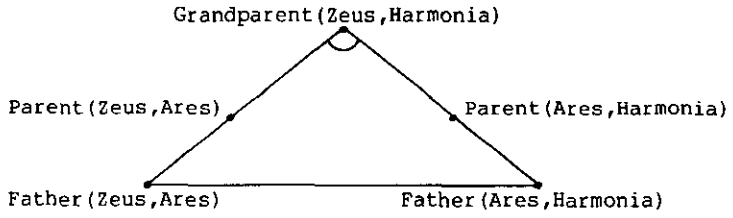
The family relationships example

The concepts of top-down and bottom-up inference apply to any set of Horn clauses. The clauses which define family relationships, F1-19 of Chapter 1, provide another example.

Given clauses F1-19, the problem of showing that Zeus is a grandparent of Harmonia can be represented as the problem of filling in the triangle



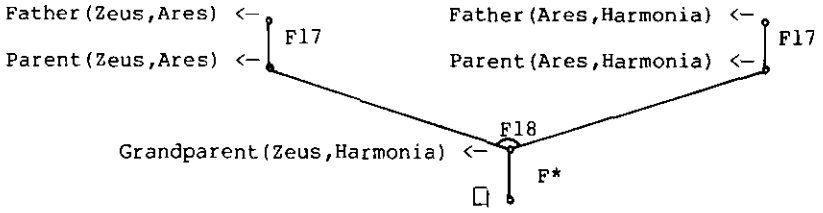
with a derivation tree:



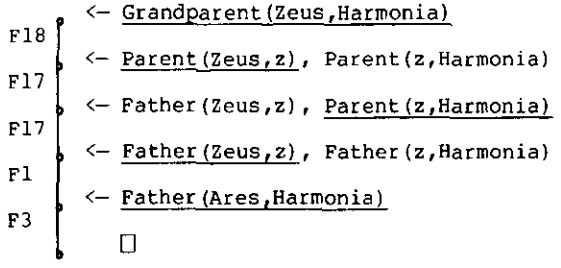
In the clausal form of logic, the problem is to show that the denial

F* <- Grandparent(Zeus,Harmonia)

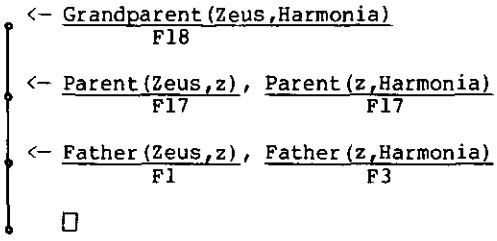
is inconsistent with the clauses F1-19. The figures below illustrate bottom-up, top-down, and parallel top-down refutations.



A bottom-up refutation of F* and F1-19



A top-down refutation of F* and F1-19

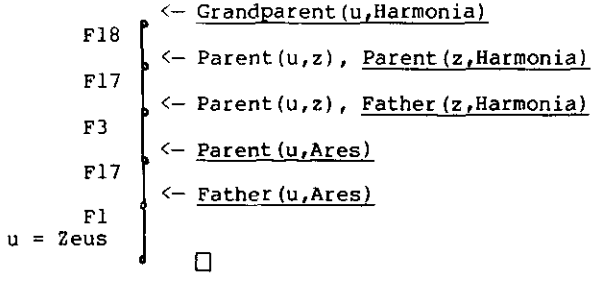


A parallel top-down refutation of F* and F1-19.

Because the operation of matching atomic formulae is so general, top-down and bottom-up inference can be used not only to show that Zeus is a grandparent of Harmonia but also to find a grandparent of Harmonia or to find a grandchild of Zeus. This is illustrated in the top-down refutation which shows the inconsistency of F1-19 with F**.

F** <- Grandparent(u,Harmonia)

The grandparent of Harmonia whose existence contradicts F** can be determined by analysing the matching substitutions used in the refutation. The last step of the refutation matches the variable u from the initial denial with the constant symbol "Zeus", determining that u = Zeus is a grandparent of Harmonia.



Notice that the first step of the refutation matches the condition

Grandparent(u,Harmonia)

with the conclusion

Grandparent(x,y).

Top-down inference uses a most general substitution which makes the two atoms identical, in this case

{x = u, y = Harmonia}.

Any less general substitutions, such as

{x = Ares, u = Ares, y = Harmonia}
or {x = Zeus, u = Zeus, y = Harmonia}

which also makes the two atoms identical, need not be considered.

Given any two atoms, all (most general) matching substitutions differ only in the names they give to variables and are otherwise equivalent. Consequently, it is necessary to use only one of them in any inference step. The matching substitution

{u = x, y = Harmonia}

for example, is equivalent to the one used in the first step of the refutation above. It gives rise to the equivalent denial

<- Parent(x,z), Parent(z,Harmonia)

which is a variant of the other.

The possibility of restricting instantiation to the generation of most general matching substitutions was observed by Prawitz [1960] and elaborated by Robinson [1965a] who incorporated it into the resolution rule (Chapter 8), which generalises the top-down and bottom-up inference rules investigated in this chapter. Unification algorithms for matching atomic formulae have been the subject of much investigation [Robinson 1971], [Paterson and Wegman 1976], [Martelli and Montanari 1977].

Inference rules and search strategies

Inference rules are the building blocks of proof procedures. A proof procedure is a systematic method for showing that a set of assumptions imply a conclusion. Proof procedures for the clausal form of logic are refutation procedures, which show that assumptions imply a conclusion by demonstrating that the assumptions are inconsistent with the denial of the conclusion.

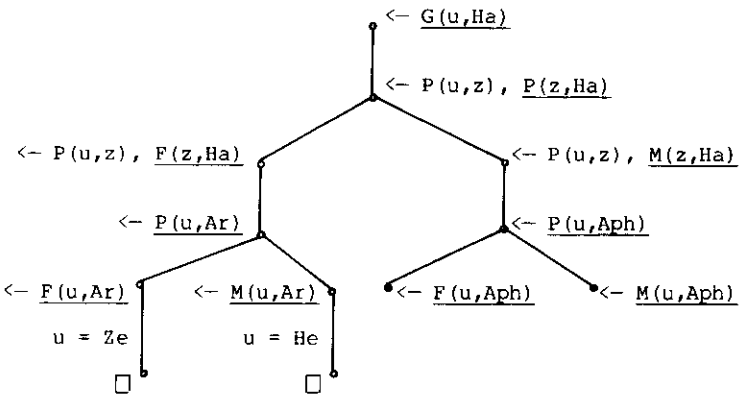
Inference rules specify the form of the individual steps which make up a proof. All possible ways of applying the inference rules, both to an initially given set of clauses and to the clauses derived from them, determine the search space for the set of clauses. Specifying a systematic search strategy for investigating clauses in the search space determines a proof procedure.

Top-down inference determines search spaces which have the form of a tree. Individual nodes of the search space are labelled by denials which contain a selected condition. For each input clause whose conclusion matches the selected condition there is an arc, labelled by the input clause, which leads to the denial obtained by applying top-down inference. A refutation is a path in the search space leading from the initial denial to the empty clause □.

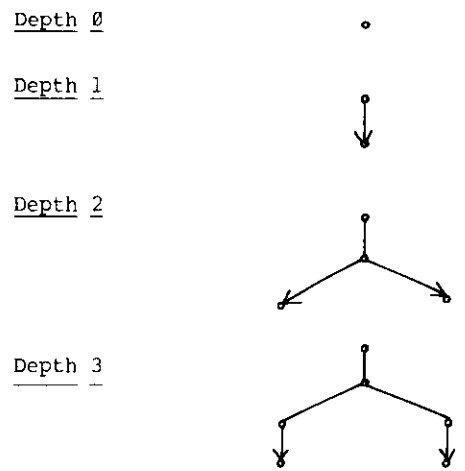
A top-down search space for the problem of finding a grandparent of Harmonia is illustrated in the figure below. To save space, abbreviations such as

Ha for Harmonia
He for Hera
P for Parent etc.

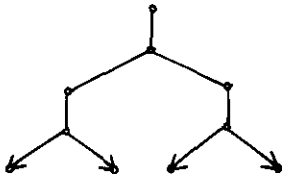
have been used for constant symbols and predicate symbols, and the input clauses labelling arcs have been omitted. Darkened nodes at the tips of the search tree contain selected conditions which match the conclusion of no input clause.



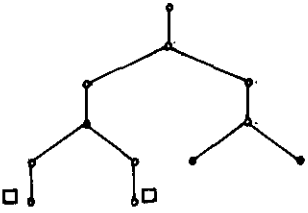
The search space is finite and can be searched completely in a finite amount of time. The two main kinds of search strategy are breadth-first and depth-first search. Breadth-first search explores all branches of the search tree to the same depth, n steps away from the root of the tree, before exploring them to the next depth, n+1 steps away from the root. Pictorially, breadth-first search explores the search space above in the following sequence:



Depth 4

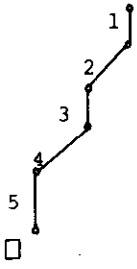


Depth 5

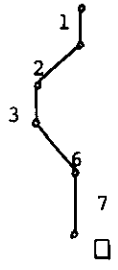


Depth-first search explores one branch of the search space at a time. When it reaches a tip of the tree it backtracks and tries an alternative branch as close to the tip as possible.

Branch 1

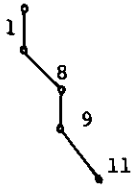


Branch 2



Branch 3

Branch 4

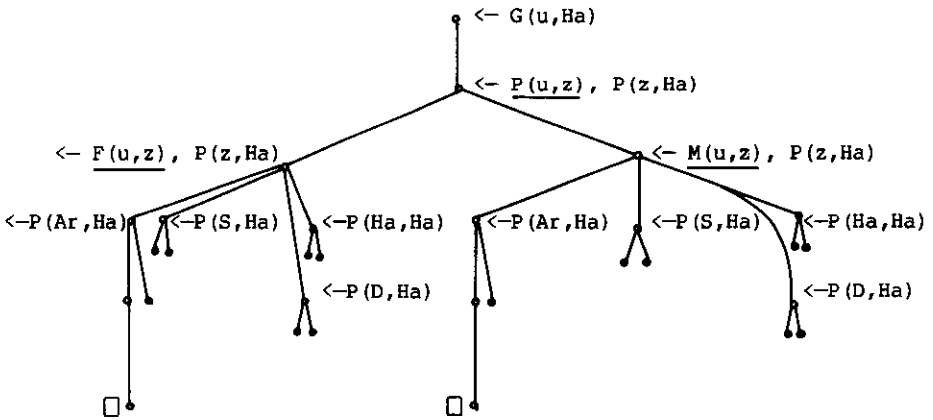


The numbers next to arcs indicate the sequence in which the arcs are generated. Here the first branch already contains a solution of the problem. If only one solution is required, then the rest of the search space need not be generated. The whole search space has to be generated, however, if all solutions are desired. In this case there are two

refutations, each of which determines a different answer to the question

Who is a grandparent u of Harmonia?
u = Zeus, u = Hera.

The search space for top-down inference is affected by the selection of conditions in denials. In the search space above, conditions were deliberately chosen with the intention of minimising the size of the search space. In the search space below, the selection of conditions maximises its size.

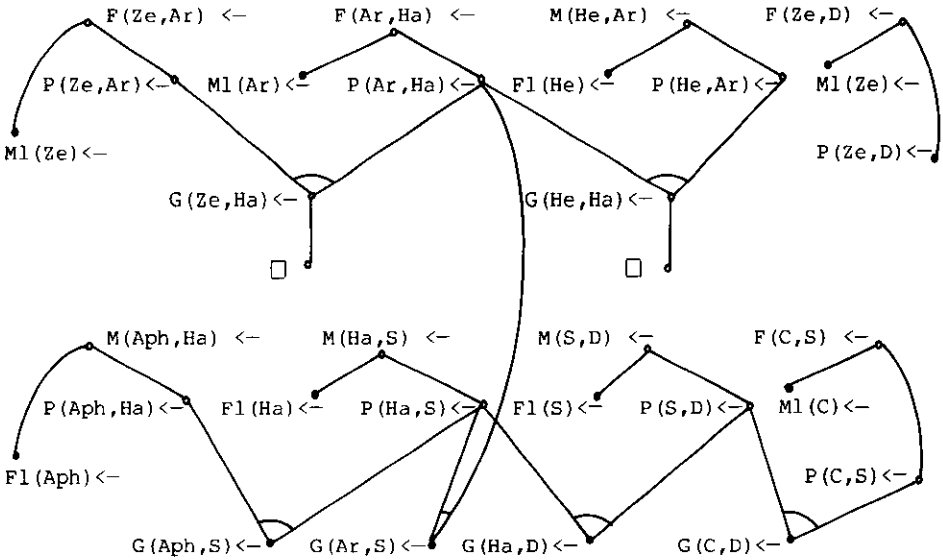


Both top-down search spaces are complete in the sense that they contain a refutation if the set of clauses is inconsistent. It suffices, therefore, to search either one search space or the other. In general, other things being equal, the larger the search space the more difficult it is for the search strategy to find a refutation.

In the problem-solving interpretation of top-down inference, the selection of a condition in a denial is the selection for solution of a subgoal from a set of subgoals. It is one of the most important considerations of problem-solving strategy and a major topic of the next two chapters.

The structure of bottom-up search spaces is more complex than that of top-down search spaces. Consequently, they are more difficult to search. The figure below illustrates the bottom-up search space for the family relationships example. Nodes are labelled by assertions. A bundle of arcs connects the assertions which match the conditions of an input clause with the new assertion derived by bottom-up inference. The input clause which ought to label the bundle is omitted to save space. Darkened nodes indicate assertions to which no bottom-up inference step applies. The same abbreviations are used as before. In addition, we use

Ml for Male and
Fl for Female.



Not included in the figure are the input assertions, such as

God(Zeus) <- and Fairy-Princess(Harmonia) <-

which match no conditions. Notice that the assertion

Male(Zeus) <-

is derived in two different ways, giving rise to two nodes labelled by the same assertion. In the next chapter, we consider representations of search spaces in which different nodes are labelled by different clauses.

In practice, few strategies other than breadth-first search have been applied to bottom-up search spaces. As in top-down search spaces, breadth-first search explores all assertions of depth n before generating any of depth n+1. The depth of an assertion is one greater than the maximum of the depths of its parent assertions.

Search strategies are an important part of all problem-solving systems and are investigated in greater detail in the next chapter.

Infinite search spaces: natural numbers

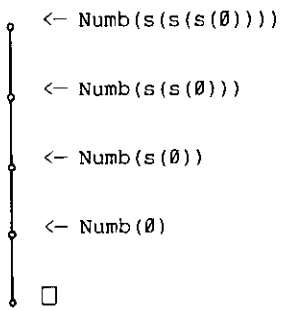
The search spaces for the parsing problem and the family relationships problems are both finite. Infinite search spaces are normally associated with clauses containing function symbols. The definition of natural number using the successor function symbol is a simple example.

```
Numb(∅) <-
Numb(s(x)) <- Numb(x)
```

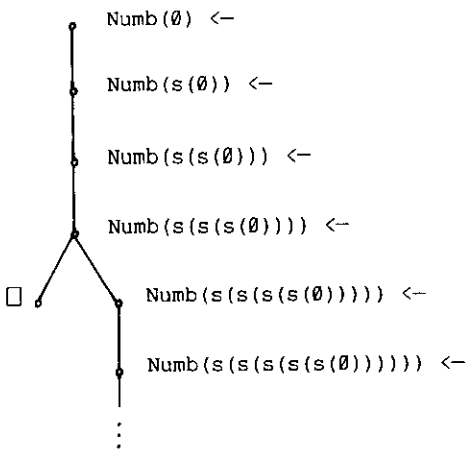
Suppose the problem is to show that three is a number.

```
<- Numb(s(s(s(∅))))
```

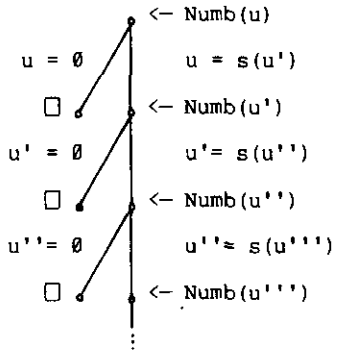
The top-down search space is finite



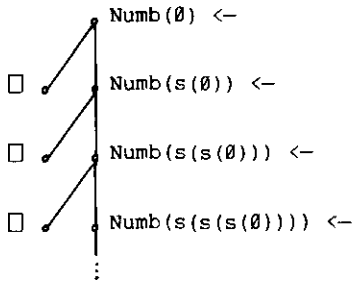
and contains only the solution of the problem. The bottom-up search space, however, is infinite.



For the problem of finding a number, however, both search spaces are infinite. Moreover, both spaces contain an infinite number of solutions.



Here each arc of the top-down search space is labelled by that part of the matching substitution which is needed to find the number u whose existence is denied in the initial statement of the problem.



When search spaces are infinite, depth-first search strategies are subject to the possibility of following the wrong branch of the search space and thus failing to find a refutation. In the present example, this happens in the top-down search space if the clause

Numb(s(x)) ← Numb(x)

is always used before the assertion

Numb(∅) ←

and in the bottom-up search space if

Numb(s(x)) ← Numb(x)

is always used before the denial

← Numb(u).

To guarantee the completeness of a proof procedure, not only must the search space be complete, but the search strategy must be exhaustive: eventually investigating every node of the search space.

Definitions

Some of the concepts introduced in this chapter are defined more precisely below:

Let S be a set of Horn clauses and let there be given a selection strategy which picks a condition from any denial. A sequence of denials

$$C_1, C_2, \ldots, C_n$$

is a top-down derivation of C_n from S if

1) the first clause C_1 belongs to S and
2) every denial in the sequence, other than the first, is obtained from the preceding denial by an application of top-down inference, using a clause in S.

A derivation of the empty clause from S is a refutation of S.

Given a denial

$$\leftarrow A_1,\ldots,A_{i-1},A_i,A_{i+1},\ldots,A_m \qquad m \geq 1$$

with selected atom A_i and an implication

$$B \leftarrow B_1,\ldots, B_n \qquad n \geq \emptyset$$

which shares no variables with the denial, a new denial can be obtained by top-down inference if the selected atom A_i matches the conclusion B of the implication. The new denial consists of all the conditions of the original denial (except for the selected condition) together with all the conditions of the implication, with the matching substitution Θ applied:

$$\leftarrow (A_1,\ldots,A_{i-1},B_1,\ldots,B_n,A_{i+1},\ldots,A_m)\Theta$$

If the denial and the implication contain variables in common, then they have to be renamed, giving equivalent clauses which share no variables, before top-down inference is attempted. Thus to apply top-down inference to the denial

$$\leftarrow \underline{Np(y,u)}, Vp(u,z)$$

using the clause

$$Np(x,y) \leftarrow Det(x,u), Noun(u,y)$$

it is necessary to rename variables first, using, for example, the variant implication

$$Np(x',y') \leftarrow Det(x',u'), Noun(u',y')$$

to obtain the new denial

$$\leftarrow Det(y,u'), Noun(u',u), Vp(u,z)$$

where the matching substitution is

$$\{x' = y,\ y' = u\}.$$

In general, any condition can be selected in a denial. The selection strategy is of the last-in-first-out kind if the selected condition is always one of the conditions most recently introduced into the denial, in particular one of the conditions

$$B_1\theta,\ldots,B_n\theta$$

in the new denial

$$\leftarrow A_1\theta,\ldots,A_{i-1}\theta,B_1\theta,\ldots,B_n\theta,A_{i+1}\theta,\ldots,A_m\theta .$$

A top-down derivation can be represented as a graph by associating a node with every denial C_j in the derivation and by inserting an arc, from it to the next denial C_{j+1}, labelled by the implication used in the inference step.

The definition of matching substitution is needed to define both top-down and bottom-up inference and will be presented after the top-level of the definition of bottom-up inference.

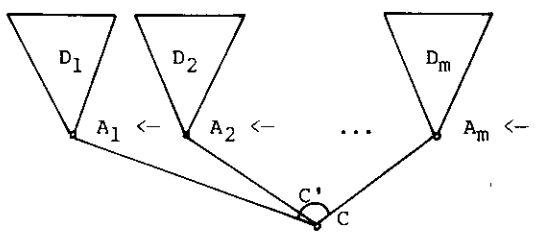
It is convenient to define a graph-representation of bottom-up inference from the outset. Let S be a set of Horn clauses. A graph D with nodes labelled by assertions is a bottom-up derivation of a clause C from S if

1) D consists of a single node labelled by C, belonging to S and C is either an assertion or the empty clause, or

2) D consists of subderivations,

D_1 of $A_1 \leftarrow$ from S,
D_2 of $A_2 \leftarrow$ from S,
.
.
.
D_m of $A_m \leftarrow$ from S,

whose root nodes are connected by arcs to a new node labelled by C and C is obtained from $A_1 \leftarrow$, $A_2 \leftarrow$, ..., $A_m \leftarrow$ by bottom-up inference using a clause C' in S.



The clause C' labels the bundle of arcs associated with the inference step.

It is convenient to define the bottom-up inference of clause C from m assertions

$$A_1 \leftarrow , \quad A_2 \leftarrow , \ldots, \quad A_m \leftarrow$$

using clause C' by decomposing the inference into a sequence of m simpler inference steps. Suppose that C' has the form

$$B \leftarrow B_1, B_2, \ldots, B_m \quad \text{or}$$
$$\leftarrow B_1, B_2, \ldots, B_m.$$

The clause C is obtained by bottom-up inference using C' from

$$A_1 \leftarrow , \quad A_2 \leftarrow , \ldots, \quad A_m \leftarrow$$

1) by selecting a condition, say B_1, of C', matching it with an assertion, say $A_1 \leftarrow$, and deriving the intermediate clause C"

$$(B \leftarrow B_2, \ldots, B_m)\theta \quad \text{or}$$
$$(\leftarrow B_2, \ldots, B_m)\theta$$

where θ is the matching substitution and

2) deriving C by bottom-up inference from $A_2 \leftarrow, \ldots, A_m \leftarrow$ using C".

3) If m=1 then C = C".

4) In step (1) the variables in $A_1 \leftarrow$ need to be distinct from those in C'. If necessary, variables need to be renamed to make them distinct.

It can be shown that the conditions in C' can be selected in any order without affecting the clause C which is finally derived.

The assertions $A_1 \leftarrow, \quad A_2 \leftarrow, \ldots, \quad A_m \leftarrow$ to which bottom-up inference is applied need not all be distinct. For example, the assertion

Friends(Narcissus, Narcissus) <-

can be derived in one step of bottom-up inference from two copies of the assertion

Likes(Narcissus, Narcissus) <-

using the clause

Friends(x,y) <- Likes(x,y), Likes(y,x).

Substitution and matching

It remains to define the notions of substitution and matching.

A substitution

$$\{x_1=t_1, \ldots, x_m=t_m\}$$

is a set of substitution components of the form

$$x_i = t_i$$

where x_i is a variable and t_i is a term. Distinct substitution components of a substitution

$$x_i = t_i \quad \text{and} \quad x_j = t_j$$

have distinct variables x_i and x_j. Thus a substitution can be regarded as a function which maps variables onto terms. If E is an expression (term, atom, or clause) then the result of applying the substitution

$$\theta = \{x_1=t_1, \ldots, x_m=t_m\}$$

to E is a new expression

$$E\theta$$

which is identical to E except that for every component $x_i=t_i$ which belongs to θ, wherever E contains an occurrence of x_i, $E\theta$ contains an occurrence of t_i. The new expression $E\theta$ is said to be an instance of E.

A substitution σ unifies the two expressions E_1 and E_2 if it makes them identical, i.e.

$$E_1\sigma = E_2\sigma.$$

$E_1\sigma$ is the common instance of E_1 and E_2 determined by σ. A substitution θ matches E_1 and E_2 (is a most general unifier of E_1 and E_2) if

1) θ unifies E_1 and E_2 and

2) the common instance
$$E_1\sigma$$
determined by any other unifier σ of E_1 and E_2 is an instance of the common instance
$$E_1\theta$$
determined by θ. Thus
$$E_1\sigma = (E_1\theta)\lambda$$
for some substitution λ.

Every pair of expressions which can be unified can also be matched. Moreover, all matching substitutions are equivalent, in the sense that the common instances they determine are variants.

Correctness and completeness of inference systems

A system of inference rules is correct (or sound) if every set of clauses which has a refutation constructed in accordance with the inference rules is inconsistent. The system is complete if every inconsistent set has a refutation. The notions of correctness and completeness connect semantics with the part of syntax concerned with proof theory. An inference system which is both correct and complete is one for which the semantic notion of inconsistency coincides with the proof theoretic notion of refutability. The correctness of top-down and bottom-up inference is easy to verify.

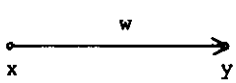
Bottom-up inference is a special case of the hyper-resolution rule defined and proved complete by Robinson [1965b]. Top-down inference is a form of the model elimination rule introduced and proved by Loveland [1968, 1969]. Like hyper-resolution, model elimination applies to arbitrary sets of clauses. In both cases for non-Horn clauses, however, an additional rule of inference, the factoring rule, discussed in Chapter 7, is needed for completeness.

Many forms of top-down inference have been developed, notably linear resolution [Loveland 1970], [Luckham 1970], ordered linear resolution [Reiter 1971], SL-resolution [Kowalski and Kuehner 1971], G-deduction [Michie et al 1972], inter-connectivity graph resolution [Sickel 1976] and analytic resolution [Brand 1976]. Linear resolution employs no restriction on the selection of atoms for top-down inference. Given a denial containing n atoms it potentially investigates the n! redundant sequences in which the atoms can be selected. The other systems, including model elimination, employ last-in-first-out selection procedures. The importance of selecting atoms in a more flexible manner will be studied in the next two chapters. Completeness for top-down inference systems employing arbitrary selection procedures has been proved by several authors including Brown [1973] and Hill [1974].

Top-down and bottom-up inference are special cases of the resolution rule [Robinson 1965a]. A system which mixes top-down and bottom-up inference for Horn clauses has been described by Kuehner [1972]. The connection graph proof procedure [Kowalski 1974a] investigated in Chapter 8 combines both directions of inference for non-Horn clauses as well. A non-resolution system which uses the standard form of logic rather than clausal form has been developed for applications in mathematical theorem-proving by Bledsoe and his colleagues [1971, 1977]. His system also combines bottom-up reasoning forwards from assumptions together with top-down reasoning backwards from conclusions.

Exercises

1) A string of items can be regarded as a directed graph whose nodes are spaces and whose arcs are labelled by items connecting one space to the next. An arc labelled by an item connecting space x to space y



can be represented by means of a three place relationship

Conn(x,w,y).

Thus the assertions

Conn(4,D,2) <-
Conn(2,A,3) <-
Conn(3,D,f) <-

represent the string

D A D

whose spaces are arbitrarily named

4, 2, 3, f .

A string is a palindrome if it reads the same backwards as it does forwards. Express the following more precise definition by means of Horn clauses.

a) A string from space x to space y is a palindrome if the item from x to x' is the same as the item from y' to y and the string from x' to y' is a palindrome.

b) A string from x to y is a palindrome if there is an item from x to y.

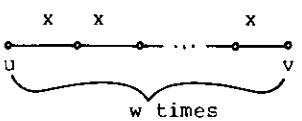
c) A string from x to x is a palindrome.

Construct both top-down and bottom-up solutions for the problem of showing that the string D A D is a palindrome.

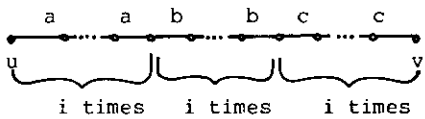
2) Let strings be represented by means of the three place Conn relation as in exercise (1).

a) Define by means of Horn clauses the relationships

Identical(w,x,u,v) which holds when the string from u to v consists of w copies of the same item x, i.e.



Admissible(u,v) which holds when, for some i, the string from u to v consists of i copies of item a followed by i copies of item b followed by i copies of c, i.e. has the form



b) Exhibit the entire top-down and bottom-up search spaces for the problem of showing that the string a b c is admissible. In the case of the top-down search space select conditions in a manner which minimises the size of the search space.

3) Using the clause

Distance(x,y,w) <- Distance(x,z,u), Distance(z,y,v), Plus(u,v,w)

and any assertions such as

Plus(3,2,5) <-
Plus(5,4,9) <-

which are necessary for the Plus relation, construct top-down and bottom-up solutions to the problem

<- Distance(A,M,w)

for the graph shown in exercise (7) of Chapter 1. How many distinct solutions does the top-down search space contain? Is the problem

<- Distance(x,x,w)

solvable?

4) The relation $x \leq y$ can be defined by the Horn clauses

$\emptyset \leq x$ <-
$s(x) \leq s(y)$ <- $x \leq y$.

Generate the top-down and bottom-up search spaces (where they are finite) for the following problems.

a) <- $s(s(\emptyset)) \leq s(s(s(\emptyset)))$

b) <- $s(s(\emptyset))) \leq w$

c) <- $w \leq s(s(\emptyset))$

d) <- $s(s(w)) \leq s(w)$

e) <- $s(s(w)) \leq s(\emptyset)$

5) Define the relation Plus(x,y,z) which holds when x+y = z. You can use two clauses, one for the case x is $\emptyset$, the other for the case x is s(x').

6) Assume that the relations

Plus(x,y,z) and Times(u,v,w)

are defined by variable-free assertions and hold whenever x+y = z and u*v = w respectively.

a) Let Exp(x,y,z) stand for the relation x to the exponent y is z, written x↑y = z. Express the following sentences in clausal form, without using function symbols.

x↑1 = x for all x.
x↑(u+v) = y*z if x↑u = y and x↑v = z.
x↑u = z if x↑(u+v) = w and x↑v = y and y*z = w.

b) Using the clauses from part (a) solve the following problems by means of both top-down and bottom-up refutations.

If 2↑a = 10 and a+a = b, then find w such that 2↑b = w.
If 3↑c = 12 and b+1 = c then find w such that 3↑b = w.
Show that for every x there is a z such that x↑0 = z.

You may need to assume such obvious facts about multiplication as Times(1,x,x) <— .