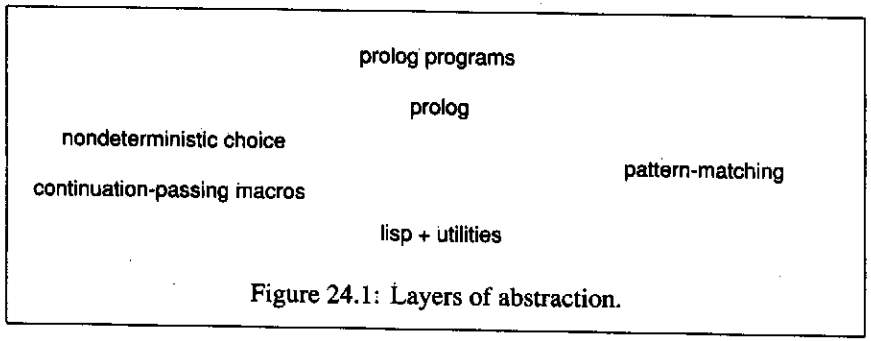

Figure 24.1: Layers of abstraction.

```
> (fact painter reynolds)
(REYNOLDS)
> (fact painter gainsborough)
(GAINSBOROUGH)
> (with-answer (painter ?x)
    (print ?x))
GAINSBOROUGH
REYNOLDS
NIL
```

Conceptually, Prolog is the database program with the addition of *rules*, which make it possible to satisfy a query not just by looking it up in the database, but by inferring it from other known facts. For example, if we have a rule like:

```
If   (hungry ?x) and (smells-of ?x turpentine)
Then (painter ?x)
```

then the query `(painter ?x)` will be satisfied for `?x` = `raoul` when the database contains both `(hungry raoul)` and `(smells-of raoul turpentine)`, even if it doesn't contain `(painter raoul)`.

In Prolog, the if-part of a rule is called the *body*, and the then-part the *head*. (In logic, the names are *antecedent* and *consequent*, but it is just as well to have separate names, to emphasize that Prolog inference is not the same as logical implication.) When trying to establish bindings[1] for a query, the program looks first at the head of a rule. If the head matches the query that the program is trying to answer, the program will then try to establish bindings for the body of the rule. Bindings which satisfy the body will, by definition, satisfy the head.

The facts used in the body of the rule may in turn be inferred from other rules:

[1]Many of the concepts used in this chapter, including this sense of bindings, are explained in


```
If   (gaunt ?x) or (eats-ravenously ?x)
Then (hungry ?x)
```

and rules may be recursive, as in:

```
If   (surname ?f ?n) and (father ?f ?c)
Then (surname ?c ?n)
```

Prolog will be able to establish bindings for a query if it can find some path through the rules which leads eventually to known facts. So it is essentially a search engine: it traverses the tree of logical implications formed by the rules, looking for a successful path.

Though rules and facts sound like distinct types of objects, they are conceptually interchangeable. Rules can be seen as virtual facts. If we want our database to reflect the discovery that big, fierce animals are rare, we could look for all the x such that there are facts `(species` x`)`, `(big` x`)`, and `(fierce` x`)`, and add a new fact `(rare` x`)`. However, by defining a rule to say

```
If   (species ?x) and (big ?x) and (fierce ?x)
Then (rare ?x)
```

we get the same effect, without actually having to add all the `(rare` x`)` to the database. We can even define rules which imply an infinite number of facts. Thus rules make the database smaller at the expense of extra processing when it comes time to answer questions.

Facts, meanwhile, are a degenerate case of rules. The effect of any fact F could be duplicated by a rule whose body was always true:

```
If   true
Then F
```

To simplify our implementation, we will take advantage of this principle and represent facts as bodyless rules.

24.2 An Interpreter

Section 18.4 showed two ways to define `if-match`. The first was simple but inefficient. Its successor was faster because it did much of its work at compile-time. We will follow a similar strategy here. In order to introduce some of the topics involved, we will begin with a simple interpreter. Later we will show how