The ability to compute with function symbols (as well as logical connectives and quantifiers) has some distinct advantages. It enables a flexible form of overloading and other powerful notational customizations. These are features that we will be using extensively in this book.

Keep in mind that **define** is a top-level directive, not a procedure. It can only appear as a direct command to Athena; it cannot appear inside a phrase.

We can use square-bracket list notation to define several values at the same time. For instance:

```
> define [x y z] := [1 2 3]

Term x defined.

Term y defined.

Term z defined.

> [z y x]

List: [3 2 1]
```

This feature is similar to the use of patterns inside **let** bindings, discussed in Section 2.16.

2.6 Assumption bases

At all times Athena maintains a global set of sentences called the *assumption base*. We can think of the elements of the assumption base as our premises—sentences that we regard (at least provisionally) as true. Initially the system starts with a small assumption base. Every time an axiom is postulated or a theorem is proved at the top level,[12] the corresponding sentence is inserted into the assumption base.

The assumption base can be inspected with the procedure `show-assumption-base`. It is a nullary procedure (i.e., it does not take any arguments), so it is invoked as

```
(show-assumption-base).
```

A related nullary procedure, `get-ab`, returns a list of all the sentences in the current assumption base. Because this is a very commonly used procedure, it also goes by the simpler name `ab`.

When Athena is first started, the assumption base has a fairly small number of sentences in it, mostly axioms of some very fundamental built-in theories, such as pairs and options.

12 A "theorem" is simply a sentence produced by a deduction *D*. We say that the deduction proves or derives the corresponding theorem (which is also the deduction's *conclusion*).

```
> (length (ab))

Term: 20
```

A sentence can be inserted in the assumption base with the top-level directive **assert**:

```
> assert (joe =/= father joe)

The sentence
(not (= joe
        (father joe)))
has been added to the assumption base.

Unit: ()
```

The unary procedure holds? can be used to determine whether a sentence is in the current assumption base. We can confirm that the previous sentence was inserted in the assumption base by checking that it holds afterward:

```
> (holds? (joe =/= father joe))

Term: true
```

Multiple sentences can be asserted at the same time; the general syntax form is

assert $p_1, \ldots, p_n$

Lists of sentences can also appear as arguments to **assert**. A list of sentences $[p_1 \cdots p_n]$ can be given a name with a definition, and then asserted by that name. This is often the most convenient way to make a group of assertions:

```
define facts := [(1 = 1) (joe = joe)]

assert facts
```

Alternatively, we can combine definition with assertion in one fell swoop:

```
assert facts := [(1 = 1) (joe = joe)]
```

All of these features work with **assert*** as well, which is equivalent to **assert** except that it first universally quantifies each given sentence over all of its free variables before inserting it into the assumption base (see page 7).

There are also two mechanisms for removing sentences from the global assumption base:

clear-assumption-base

and **retract**. The former will empty the assumption base, while the second will only remove the specified sentence:

```
> clear-assumption-base

Assumption base cleared.

> assert (1 = 2)

The sentence
(= 1 2)
has been added to the assumption base.

> retract (1 = 2)

The sentence
(= 1 2)
has been removed from the assumption base.
```

All three of these (**assert**, **clear-assumption-base**, and **retract**) are top-level directives; they cannot appear inside other code.

The last two constructs are powerful and should be used sparingly. A directive

retract *p*

in particular, simply removes *p* from the assumption base without checking to see what other sentences might have been derived from *p* in the interim (between its assertion and its retraction), so a careless removal may well leave the assumption base in an incorrect state. This tool is meant to be used only when we assert a sentence *p* and then right away realize that something went wrong, for instance, that *p* contains certain free variables that it should not contain, in which case we can promptly remove it from the assumption base with **retract**.

2.7 Datatypes

A **datatype** is a special kind of domain. It is special in that it is *inductively generated*, meaning that every element of the domain can be built up in a finite number of steps by applying certain operations known as the *constructors* of the datatype. A datatype *D* is specified by giving its name, possibly followed by some sort parameters (if *D* is polymorphic; we discuss that in Section 2.8), and then a nonempty sequence of constructor profiles separated by the symbol **|**. A constructor profile without selectors[13] is of the form

$$(c \;\; S_1 \cdots S_n), \tag{2.3}$$

consisting of the name of the constructor, *c*, along with *n* sorts $S_1 \cdots S_n$, where S_i is the sort of the i^{th} argument of *c*. The range of *c* is not explicitly mentioned—it is tacitly understood

13 Selectors are discussed in Section A.5.