Partial Functions

Partial functions arise naturally.

Partial functions arise in a number of situations. CASL provides means for the declaration of partial functions, the specification of their domains of definition, and more generally the specification of system properties involving partial functions. The aim of this chapter is to discuss and illustrate how to handle partial functions in CASL specifications.

4.1 Declaring Partial Functions

Partial functions are declared differently from total functions.

```plaintext
spec Set_Partial_Choose [sort Elem] =
    Generated_Set [sort Elem]
then op choose : Set →? Elem
end
```

The choose function on sets is naturally a partial function, expected to be undefined on the empty set. In CASL, a partial function is declared similarly to a total one, but for the question mark ‘?’ following the arrow in the profile. It is therefore quite easy to distinguish the functions declared to be total from the ones declared to be partial.

A function declared to be partial may happen to be total in some of the models of the specification. For instance, the above specification Set_Partial_Choose does not exclude models where the function symbol choose is interpreted by a total function, defined on all set values. Axioms can be
used to specify the domain of definition of a partial function, and how to do this is detailed later in this chapter.

Terms containing partial functions may be undefined, i.e., they may fail to denote any value.

For instance, the (value of the) term \( \text{choose}(\text{empty}) \) may be undefined.\(^1\) This is more natural than insisting that \( \text{choose}(\text{empty}) \) has to denote some arbitrary but fixed element of \( \text{Elem} \).

Note that variables range only over defined values, and therefore a variable always denotes a value, in contrast to terms containing partial functions.

Functions, even total ones, propagate undefinedness.

If the term \( \text{choose}(S) \) is undefined for some value of \( S \), then the term \( \text{insert}(\text{choose}(S), S') \) is undefined as well for this value of \( S \), although \( \text{insert} \) is a total function.

Predicates do not hold on undefined arguments.

\( \text{CASL} \) is based on classical two-valued logic. A predicate symbol is interpreted by a relation, and when the value of some argument term is undefined, the application of a predicate to this term does not hold. For instance, if the term \( \text{choose}(S) \) is undefined, then the atomic formula \( \text{choose}(S) \text{ is in } S \) does not hold.

Equations hold when both terms are undefined.

In \( \text{CASL} \), equations are by default \textit{strong}, which means that they hold not only when both sides denote equal values, but also when both sides are simultaneously undefined. For instance, let us consider the equation:

\[
\text{insert}(\text{choose}(S), \text{insert}(\text{choose}(S), \text{empty})) = \text{insert}(\text{choose}(S), \text{empty})
\]

\(^1\) Note that the term \( \text{choose}(\text{empty}) \) is well-formed and therefore is a ‘correct term’. It is \textit{its value} which may be undefined. To avoid unnecessary pedantry, in the following we will simply write that a term is undefined to mean that its value is so. Obviously, a term with variables may be defined for some values of the variables and undefined for other values.