Appendix B

Tutorial

The ELFE system allows for verifying mathematical proofs. In this tutorial, we will take a look at its language features.

Formulas

In order to make mathematical statements, we use a language that is similar to first-order logic. Concretely, we have the following syntax constructs at our disposal:

for all *var*. *formula*	– an universally quantified statement
exists *var*. *formula*	– an existentially quantified statement
formula* iff *formula	– if and only if
formula* implies *formula	– an implication
formula* and *formula	– both should be true
formula* or *formula	– either one is true
not *formula*	– a negation
contradiction	– corresponds to a $\perp$ in first-order logic

We can use alphanumeric strings as variables. In order to introduce mathematical properties, we will use atoms inside a formula. We can construct atoms as follows:

var* is *predicate
var* is not *predicate
term* = *term
predicate(term, ... , term)

For example, we may say 'R is symmetric' or 'symmetric(R)'. Terms can be variables or more complex – for example, the union of two sets A and B. This can be written down similar with 'union(A,B)'. Alternatively, we can use syntactic sugar: $A \cup B$. The libraries contain several different sugars. Note that we need to insert brackets when nesting sugars: '$((A \cup B)^C) = ((A^C) \cap (B^C))$'.

Top level commands

On the top level, we have six possible commands. The following three allow us to introduce mathematical statements:

Definition *formula*.	– define a statement
Proposition *formula*.	– derive a statement without proof
Lemma *formula*. Proof: *proof* Qed.	– make a proved statement

Definitions and propositions can be used to introduce facts to a proof. Lemmas are the interesting part since here we can test our proofs – we will see in the next section how to write a proof.

Besides these three statements, we can use the following statements to shorten our proofs:

Include ***file.***	– includes a file of the library
Let ***var*** **be** ***predicate.***	– defines a meta-variable
Notation ***predicate*****:** ***pattern.***	– introduces a notation

The inclusion command allows us to use background libraries. With the second command we can assign a property to a variable for the whole text. For example, if we define several properties about sets, we will write in the beginning 'Let A be set' and use 'A' afterwards. The last command allows us to introduce own syntactic sugars.

```
Include relations.
Notation disjunion: A Ů B.
Let A,B be relation.
Definition disjunionDef: (A Ů B)[x,y] iff A[x,y] or B[x,y] and not (A[x,y] and B[x,y]).
Proposition: A Ů B is relation.
Lemma disjunionAssociative: A Ů B = B Ů A.
Proof:
    ...
qed.
```

TEXT B.1: Exemplary use of top sections

Proof structures

There are three ways to prove a goal within ELFE.

Splitting the goal

In order to simplify a goal, we will often make several proofs that imply the original goal:

Proof ***formula*****:** ***proof*** **Qed.**	– make several sub proofs
Case ***formula*****:** ***proof*** **Qed.**	– make case distinctions

The sub proofs can be completely unconnected – important is that the background theory proves that all sub proofs together imply the original goal. Within a case distinction, the specified formula is assumed and we want to derive the original goal.

Unfolding the structure of the goal

Often, we will want to make assumptions in a proof.

Assume ***formula.*** ***proof***	– proof an implication

Hence *formula*.

In order to prove an universally quantified statement, we can fix a particular element:

Fix *var*. – fix an universally quantified variable

Deriving intermediary steps

We can derive cornerstones for a proof with the following statement:

Then *formula*. – derive a cornerstone

We will often want to retrieve a specific element that we use afterwards in our proof. This can be done with this construction:

Take *var* such that *formula*. – fix an existing variable

```
...
Lemma disjunionAssociative: A Ů B = B Ů A.
Proof:
    Assume (A Ů B)[x,y].
    Case A[x,y]:
        Then not B[x,y].
        Then (B Ů A)[x,y].
    qed.
    Case B[x,y]:
        Then not A[x,y].
        Then (B Ů A)[x,y].
    qed.
    Hence (B Ů A)[x,y].
qed.
```

TEXT B.2: Exemplary proof