The syntax and semantics of the ForTheL language[*]

Andrei Paskevich
Université Paris XII — Val de Marne, Créteil, France
Kiev National Taras Shevchenko University, Kiev, Ukraine

December 2007

Contents

[*] This text is excerpted from the English draft of the doctoral thesis "Méthodes de formalisation des connaissances et des raisonnements mathématiques: aspects appliqués et théoriques" ("Methods of formalization of mathematical knowledge and reasoning: applied and theoretical aspects") defended at Paris XII University, December 21, 2007.

Chapter 1

Formal Theory Language

1.1 Introduction

ForTheL, an acronym for "Formal Theory Language", is a formal language of mathematical texts, which imitates the natural (English) language of mathematical publications issued by human beings. There are two reasons to pursue a verbose "natural" style instead of basing on a terse unifying notation of some traditional language of logic.

First, a text composed with correct English sentences will hopefully be more readable than a collection of formulas built with quantifiers, parentheses, lambdas, junctors and so on. As for our own experience, it is also more pleasant to write. So, the first reason is to provide our framework with a user-friendly interface.

Second, we observe in a natural human text a lot of information that lies beyond logic as such and that usually vanishes in translation. In a natural speech, we meet nouns, which denote classes of entities; adjectives and verbs, which act as attributes and restrict classes; adjectives and verbs, which act as predicates and may relate different entities. In a traditional mathematical text, we meet definitions and axioms, important theorems and auxiliary lemmas, we meet various reasoning schemes. Where human language makes distinctions, the language of mathematical logic unifies: subjects, objects, attributes, predicates all become predicate symbols; axioms, definitions, theorems all become formulas-premises; case analysis, *reductio ad absurdum*, definition expansion all become applications of *modus ponens*, or resolution, or tableau rules.

We believe that the choice of reasoning fragments to write as separate lemmas, the choice of new notions to introduce with definitions, the choice of proof schemes and everything else what structures a mathematical discourse, bear significant knowledge about the problem in question, and should be taken into account together with a purely logical content. Our intention is to preserve the distinctions, the fine structure of a human text, in formalization. We want to understand how our mind makes use of this fine structure: how we treat definitions as compared with axioms or lemmas, class-nouns as compared with adjectives, proofs by case analysis as compared with proofs by definition expansion, and so on. Basing on these observations, we improve reasoning capabilities of a machine, we implement heuristic routines directing proof search or reducing a search space with the help of non-logical knowledge extracted from a formal, yet "human-like" input text.

The ForTheL language can be seen as a kind of controlled English. It has much in common with Attempto Controlled English (ACE) [2] (though the semantics of a sentence is different in ACE) and especially with Common Logic Controlled English (CLCE) [9]. In fact, many ForTheL statements are well-formed CLCE statements with the same meaning, and vice versa. However, ForTheL has a more elaborated concept of text which captures reasoning in a natural deduction style. On the level of text, ForTheL is quite similar to the language of Mizar [10] and Isar [12].

Another approach to formalization of the commonly used mathematical language is presented in Mathematical Vernacular of de Bruijn (see [7]) and its refinement, Weak Type Theory (WTT) [4]. It is noteworthy that notions ("nouns") and attributes ("adjectives") in WTT are particular weak types and participate in type derivations.

The following sections describe ForTheL, its syntax and semantics. Our description proceeds bottom-up with respect to ForTheL's syntactic and semantic structure:

1. At the bottom is the lexical structure, which determines how a sequence of ASCII-characters transforms to a chain of ForTheL *lexemes*.

2. At the next level come *units*: terms, notions, predicates, statements. Units are composed on top of predefined and user-defined *syntactic primitives* in accordance with the rules of ForTheL grammar. The meaning of a ForTheL statement can be given in terms of first-order language.

3. Then we consider ForTheL *sections*. The smallest section is a *sentence* which is merely an "enveloped" statement. Chains of sentences form proofs and top-level sections, such as axioms, definitions, signature extensions, and propositions. Semantics of a section is defined by a number of characterizing properties: its kind, first-order meaning, the set of declared variables, etc. These properties are determined both by the section's content and by the section's context: the set of sections that *logically precede* that section in the text.

4. Finally, a ForTheL *text* is a sequence of top-level sections. The semantics of a text depends on the task we wish to perform with it. The present work mostly focuses on mathematical verification, and the corresponding notion of text correctness will be introduced and studied in the next chapter.

1.2 Lexical structure

We use BNF notation to present syntax. Nonterminals are written in italic (e.g. *attribute*) and terminals in typewriter font (e.g. `therefore`). Grammar productions have the form:

$$\mathit{nonterm} \;\rightarrow\; \mathit{alt}_1 \mid \mathit{alt}_2 \mid \ldots \mid \mathit{alt}_n$$

and the following conventions are adopted:

$\mathit{pat}_1 \mid \mathit{pat}_2$	choice	$(\,\mathit{pattern}\,)$	grouping
$[\,\mathit{pattern}\,]$	optional	$\{\,\mathit{pattern}\,\}$	zero or more repetitions

The lexical structure of a ForTheL text is defined as follows:

```
     lexeme  →  word | symbol
       word  →  alphanum { alphanum }
   alphanum  →  alpha | numeric | _
      alpha  →  small | capital
      small  →  a | ... | z
    capital  →  A | ... | Z
    numeric  →  0 | ... | 9
     symbol  →  ( | ) | [ | ] | { | } | < | > | ‘ | ’ | " | / | \ | | | %
             |  ! | ? | @ | $ | & | % | ~ | + | - | * | = | : | ; | , | .
 whitespace  →  whitetoken { whitetoken }
 whitetoken  →  space | newline | comment
    comment  →  # {any character except newline} newline
      space  →  a space | a horizontal tab
    newline  →  a newline
```

In lexical analysis, the longest possible lexeme satisfying the grammar rules is read. In particular, a word lexeme is never followed by another word lexeme without being separated by a symbol lexeme or a whitespace.

Whitespaces serve as delimiters for lexemes and do not occur in the subsequent rules of ForTheL's grammar. There is one exception case where the presence or absence of a whitespace is taken into account: parsing a symbolic primitive (see 1.3.1) which has a token composed of several symbols, such as `->` or `!=`. According to the rules given above, the both strings consist of two consecutive lexemes. However, the same lexemes separated with the space character: `- >` will not be recognized as the token `->`.

Variables in ForTheL are denoted with Latin letters.

$$\textit{variable} \;\rightarrow\; \textit{alpha}$$

Case is significant, so that `x` and `X` are different variables.

1.3 Syntax of a statement

In order to give an idea about ForTheL statements, we begin with a couple of examples. For each statement we give its first-order meaning (the relevant notation is introduced in Section 2.2.1).

ForTheL: `every bird is a feathery animal`

Meaning: $\forall x\, (x \,\varepsilon\, \mathrm{Bird} \supset (x \,\varepsilon\, \mathrm{Animal} \wedge \mathrm{isFeathery}(x)))$

ForTheL: `every subset of some set S is equal to S`

Meaning: $\exists S\, (S \,\varepsilon\, \mathrm{Set} \wedge \forall x\, (x \,\varepsilon\, \mathrm{SubsetOf}(S) \supset x = S))$

ForTheL: `every natural number m greater than 0 divides m!`

Meaning: $\forall m\, ((m \,\varepsilon\, \mathrm{Number} \wedge m \,\varepsilon\, \mathrm{Natural} \wedge$
$\wedge\, \mathrm{isGreaterThan}(m, \mathrm{Zero})) \supset \mathrm{Divides}(m, \mathrm{Factorial}(m)))$

ForTheL: `if X and Y are close to Z then X and Y are close`

Meaning: $(\mathrm{isCloseTo}(X, Z) \wedge \mathrm{isCloseTo}(Y, Z)) \supset \mathrm{isCloseTo}(X, Y)$

ForTheL: `no equation in G has a positive solution`

Meaning: $\forall x\, ((x \,\varepsilon\, \mathrm{Equation} \wedge \mathrm{isIn}(x, G)) \supset$
$\supset \neg\exists y\, (y \,\varepsilon\, \mathrm{SolutionOf}(x) \wedge \mathrm{isPositive}(y)))$

ForTheL statements are composed of *units* of four general kinds:

- *notion* denotes a class of objects, possibly parametrized: `natural number`, `element of S`, `series that converges to N`.

- *term* denotes an object, either by pointing to a concrete value: `N`, `the complement of S`, `X * Y`; or by quantifying a class denoted with a notion: `every set`, `some divisor of M`.

- *predicate* denotes a property of an object: `empty`, `divides N`, `is a subset of S`. Applied to a term, a predicate forms a statement; applied to a notion as an attribute, it forms a new notion with a restricted class.

- *statement* denotes a logical expression which may be true or false, be atomic or composed from simpler statements: `X > Y`, `every divisor of N is a natural number`, `there exists a countable set`.

The third statement given above decomposes as follows:

$$\left.\text{term}\left\{\texttt{every}\ \underbrace{\overbrace{\texttt{natural}}^{\text{predicate}}\ \overbrace{\texttt{number m}}^{\text{notion}}\ \overbrace{\texttt{greater than 0}}^{\text{predicate}}}_{\text{notion}}\right.\ \underbrace{\texttt{divides}\ \overbrace{\texttt{m!}}^{\text{term}}}_{\text{predicate}}\right\}\text{statement}$$

Like any natural language, ForTheL is not (and is not intended to be) free from ambiguity. The syntax of ForTheL allows to write a unit that has several possible readings. For example, in the statement:

```
some point of any straight line that crosses L lies on L
```

the predicate `crosses L` can apply both to the point and to the line. Without certain reasoning capabilities, the parser has no way to choose the right variant, therefore, such units are rejected. Generally, it is not hard to find an unambiguous reformulation:

```
every straight line that crosses L has a point that lies on L
```

and, in any case, one can resort to the parentheses:

```
some point of (any straight line that crosses L) lies on L
```

1.3.1 Syntactic primitives

Syntactic primitives are the building bricks of units. At each point of parsing, we work with a certain collection of primitives that reflects the signature of the text's portion read so far. We call this collection a *current vocabulary* to indicate, first, that it is peculiar to the text under consideration and, second, that it is dynamic. There are six main groups of syntactic primitives, called *base primitives*:

- *class nouns*, to form notions: `element of` arg_1
- *definite nouns*, to form terms: `zero`, `power set of` arg_1
- *adjectives* and *verbs*, to form predicates: `converges`, `equal to` arg_1
- symbolic *operators*, including 0-ary ones, i.e. constants: `0`, arg_1 `+` arg_2, `min` arg_1
- symbolic *relations*: arg_1 `<=` arg_2, arg_1 `:` arg_2 `->` arg_3

Base primitives are introduced directly in the text with the help of definitions, signature extensions and declarations of synonyms. The syntax of these forms is described in Section 1.3.6. There also are supplementary groups of primitives that are automatically derived from the new-introduced base primitives. For example, noun primitives with an *of*- argument place, such as `subset of` arg_1 or `complement of` arg_1 `to` arg_2 produce special primitives to be used in possessive predicate units: `has an element`, `of an infinite cardinality`. Derived primitives will be considered in the subsequent sections.

To introduce a base primitive, the author must provide an appropriate *pattern* which determines its syntax. A pattern is a nonempty chain of *tokens* (which define terminals for the new primitive) alternated with variables (which denote argument places):

$$\begin{aligned}
\textit{pattern} &\rightarrow \textit{token}\ \{\,\textit{token}\,\}\ [\,\textit{variable}\ \{\,\textit{token}\ \{\,\textit{token}\,\}\ \textit{variable}\,\}\,] \\
\textit{token} &\rightarrow \textit{small}\ \{\,\textit{small}\,\} \\
\textit{symbPattern} &\rightarrow [\,\textit{variable}\,]\ \textit{symbToken}\ \{\,\textit{variable}\ \textit{symbToken}\,\}\ [\,\textit{variable}\,] \\
&\mid\ \textit{word}\ \texttt{(}\ \textit{variable}\ \{\ \texttt{,}\ \textit{variable}\,\}\ \texttt{)} \\
&\mid\ \textit{word}\ [\,\textit{variable}\,]
\end{aligned}$$

$$\mathit{symbToken} \;\rightarrow\; \mathit{symbol}\ \{\,\mathit{symbol}\,\}$$

Tokens and symbolic tokens should not contain spaces. The articles the forms of the verb “to be”, and one-letter words are not accepted as tokens.

Since nouns and verbs in English look different in singular and plural forms, ForTheL permits to “equate” tokens by joining them in a group:

$$\mathit{tokenGroup} \;\rightarrow\; \texttt{[}\ \mathit{token}\ \texttt{/}\ \mathit{token}\ \{\ \texttt{/}\ \mathit{token}\,\}\ \texttt{]}$$

For example, one can introduce the following groups:

```
[formula/formulas/formulae]
[cross/crosses]
```

A token group must appear in the text (at the top level) before a pattern using a token from that group.

Converting a pattern to a syntactical primitive, the parser transforms (grouped) tokens to (grouped choices of) terminals and replaces variables with the nonterminal *term* (in symbolic patterns, *symbTerm*) which “reads” the arguments of a primitive. In relation patterns which begin with a variable, this variable is replaced with the nonterminal *symbTerms*, to allow expressions like “`x,y <= z`”. In class nouns, the optional nonterminal *names* (Section 1.3.2) is inserted one token before the first argument place, or at the end of the pattern if there are no arguments. Consider the following examples of patterns together with the corresponding primitives (one line per base group, and assuming that the needed token groups have been introduced):

Pattern	Primitive
`subset of S`	$(\,\texttt{subset} \mid \texttt{subsets}\,)\ [\,\mathit{names}\,]\ (\,\texttt{of}\,)\ \mathit{term}$
`power set of S`	$(\,\texttt{power}\,)\ (\,\texttt{set} \mid \texttt{sets}\,)\ (\,\texttt{of}\,)\ \mathit{term}$
`infinite`	$(\,\texttt{infinite}\,)$
`divides n`	$(\,\texttt{divides} \mid \texttt{divide}\,)\ \mathit{term}$
`Pow x`	$(\,\texttt{Pow}\,)\ \mathit{symbTerm}$
`x <= y`	$\mathit{symbTerms}\ (\,\texttt{<=}\,)\ \mathit{symbTerm}$

When parsing non-symbolic primitives, the case of tokens is ignored. That is, `subset of S`, `Subset of S`, and `sUbSeT oF S` express the same notion. Tokens of symbolic primitives and variable names are case-sensitive. Thus, `sin x`, `Sin x`, `sin X` are three different terms.

The grammar rules responsible for parsing primitives are the only productions that are modified in the course of reading. In the subsequent sections, we will show these productions as hypothetical “snapshots” at some arbitrary moments of reading.

1.3.2 Notions

Any notion unit is built upon a *primary notion* to which a number of *attributes* is applied. Two types of syntactic primitives are used to form primary notions:

$$\begin{array}{rcl}
\mathit{primClassNoun} & \rightarrow & (\,\texttt{set} \mid \texttt{sets}\,)\ [\,\mathit{names}\,] \\
& \mid & (\,\texttt{element} \mid \texttt{elements}\,)\ [\,\mathit{names}\,]\ (\,\texttt{of}\,)\ \mathit{term} \\
& \mid & (\,\texttt{function} \mid \texttt{functions}\,)\ [\,\mathit{names}\,]\ (\,\texttt{from}\,)\ \mathit{term}\ (\,\texttt{to}\,)\ \mathit{term} \\
& \mid & \ldots \\
\\
\mathit{primClassRelation} & \rightarrow & \mathit{names}\ (\,\texttt{<<}\,)\ \mathit{symbTerm} \\
& \mid & \mathit{names}\ (\,\texttt{:}\,)\ \mathit{symbTerm}\ (\,\texttt{->}\,)\ \mathit{symbTerm} \\
& \mid & \ldots \\
\\
\mathit{names} & \rightarrow & \mathit{variable}\ \{\ \texttt{,}\ \mathit{variable}\,\}
\end{array}$$

Class relations are derived primitives. They are produced automatically from descriptive relations (see Sections 1.3.6 and 1.3.7). The first argument place in a class relation is taken by the nonterminal *names* and the rest is the same as in the corresponding relation. The class relations above could be produced by the following synonym declarations:

```
Let x << y stand for (x is an element of y).

Let f : D -> R denote (f is a function from D to R).
```

Every notion unit is characterized with a list of *names*. These identifiers refer to particular objects taken from the class and play the same role as the variable names attached to a quantifier. We already saw the use of names in the statement:

```
every natural number m greater than 0 divides m!
```

Constructions with more than one name may be convenient, too:

```
for all real numbers X,Y (X*Y) is a real number
```

In class nouns, the list of names may be empty:

```
every even natural number greater than 2 is compound
L,M are parallel straight lines
```

Attributes are used to restrict the class denoted by a notion. Syntactically, attributes are based on predicate and statement units:

$$\begin{aligned} \mathit{leftAttribute} &\rightarrow \mathit{primSimpleAdjective} \mid \mathit{primSimpleAdjectiveM} \\[1ex] \mathit{rightAttribute} &\rightarrow \mathit{isPredicate}\ \{\,\texttt{and}\ \mathit{isPredicate}\,\} \\ &\mid\ \texttt{that}\ \mathit{doesPredicate}\ \{\,\texttt{and}\ \mathit{doesPredicate}\,\} \\ &\mid\ \texttt{such that}\ \mathit{statement} \end{aligned}$$

Simple adjectives which form left attributes are derived primitives, too. They are produced from those adjectives and m-adjectives (1.3.4) which have no argument places: every such primitive is just copied to the group of simple (m-)adjectives:

$$\mathit{primSimpleAdjective} \rightarrow (\,\texttt{prime}\,) \mid (\,\texttt{empty}\,) \mid (\,\texttt{natural}\,) \mid \ldots$$

$$\mathit{primSimpleAdjectiveM} \rightarrow (\,\texttt{equal}\,) \mid (\,\texttt{parallel}\,) \mid (\,\texttt{disjoint}\,) \mid \ldots$$

Now we define the syntax of notion:

$$\begin{aligned} \mathit{notion} &\rightarrow \mathit{classNoun} \mid \mathit{classRelation} \\[1ex] \mathit{classNoun} &\rightarrow \{\,\mathit{leftAttribute}\,\}\ \mathit{primClassNoun}\ [\,\mathit{rightAttribute}\,] \\[1ex] \mathit{classRelation} &\rightarrow \{\,\mathit{leftAttribute}\,\}\ [\,\texttt{(}\,]\ \mathit{primClassRelation}\ [\,\texttt{)}\,]\ [\,\mathit{rightAttribute}\,] \end{aligned}$$

The following expressions are valid notion units:

```
cyclic group G

injective f : Nat -> Nat that maps 0 to 0

real number greater than 0 and less than 1

natural numbers q,r such that n = (q * m) + r and r < m
```

Note that the last unit actually denotes two different classes: one for `q` and one for `r`.

1.3.3 Terms

There are two kinds of term units in ForTheL: definite terms and quantified notions:

$$\begin{array}{rcl} \mathit{term} & \rightarrow & [\,\texttt{(}\,]\ \mathit{quantifiedNotion}\ [\,\texttt{)}\,] \\ & | & \mathit{definiteTerm} \end{array}$$

Quantified notions permit to formulate general statements about every object from the class denoted by a notion, or to state that there exists an object in the class having a certain property:

$$\begin{array}{rcl} \mathit{quantifiedNotion} & \rightarrow & (\,\texttt{every} \mid \texttt{each} \mid \texttt{all} \mid \texttt{any}\,)\ \mathit{notion} \\ & | & \texttt{some}\ \mathit{notion} \\ & | & \texttt{no}\ \mathit{notion} \end{array}$$

Remark. Quantified notions that occur as term arguments in syntactic primitives should have at most one name. Thus, the units `N divides some numbers X,Y,Z`, `a subset of finite sets A,B`, `the orders of groups G,H` are forbidden.

A *definite term* is formed by applying a function to term arguments. ForTheL allows to write such applications with English words as well as with symbolic operators. We adopt the following conventions about precedence and associativity of operators:

- *postfix* operators, whose primitives end with a token, have the highest precedence;
- *prefix* operators, whose primitives begin with a token and end with an argument place, have a lower precedence than the postfix operators;
- *infix* operators, whose primitives have argument places from the both ends, have the lowest precedence among operators and associate to the right.

Certainly, one may use parentheses to reorder a symbolic expression.

$$\begin{array}{rcl} \mathit{definiteTerm} & \rightarrow & [\,\texttt{(}\,]\ [\,\texttt{the}\,]\ \mathit{primDefiniteNoun}\ [\,\texttt{)}\,] \\ & | & \mathit{symbTerm} \\ & & \\ \mathit{symbTerm} & \rightarrow & \mathit{primInfixOperator} \mid \mathit{symbTermTighter} \\ & & \\ \mathit{symbTermTighter} & \rightarrow & \mathit{primPrefixOperator} \mid \mathit{symbTermTightest} \\ & & \\ \mathit{symbTermTightest} & \rightarrow & \mathit{primPostfixOperator} \mid \texttt{(}\ \mathit{symbTerm}\ \texttt{)} \mid \mathit{variable} \end{array}$$

Here follow typical primitives for definite nouns and operators:

$$\begin{array}{rcl} \mathit{primDefiniteNoun} & \rightarrow & (\,\texttt{zero} \mid \texttt{zeroes}\,) \\ & | & (\,\texttt{order} \mid \texttt{orders}\,)\ (\,\texttt{of}\,)\ \mathit{term} \\ & | & \ldots \\ & & \\ \mathit{primInfixOperator} & \rightarrow & \mathit{symbTermTighter}\ (\,\texttt{*}\,)\ \mathit{symbTerm} \\ & | & \ldots \\ & & \\ \mathit{primPrefixOperator} & \rightarrow & (\,\texttt{min}\,)\ \mathit{symbTermTighter} \\ & | & \ldots \\ & & \\ \mathit{primPostfixOperator} & \rightarrow & (\,0\,) \\ & | & (\,\texttt{exp}\,)\ (\,\texttt{(}\,)\ \mathit{symbTerm}\ (\,\texttt{,}\,)\ \mathit{symbTerm}\ (\,\texttt{)}\,) \\ & | & \mathit{symbTermTightest}\ (\,\texttt{(}\,)\ \mathit{symbTerm}\ (\,\texttt{)}\,) \\ & | & \ldots \end{array}$$

Note the last primitive in the group of postfix operators. It introduces function application as an abstract binary operation, using the same syntax as for signature functions like `exp`.

A *plain term* is a term that does not contain quantified notions inside. In other words, a plain term is a definite term whose arguments are plain terms:

$$\begin{array}{rcl} \mathit{plainTerm} & \rightarrow & [\,\texttt{(}\,]\ [\,\texttt{the}\,]\ \mathit{primPlainNoun}\ [\,\texttt{)}\,] \\ & | & \mathit{symbTerm} \end{array}$$

$$\begin{array}{rcl} \mathit{primPlainNoun} & \rightarrow & (\,\texttt{zero} \mid \texttt{zeroes}\,) \\ & | & (\,\texttt{order} \mid \texttt{orders}\,)\ (\,\texttt{of}\,)\ \mathit{plainTerm} \\ & | & \ldots \end{array}$$

Each definite noun primitive automatically produces a twin plain noun primitive by replacing *term*'s with *plainTerm*'s at argument places.

1.3.4 Predicates

Terms give us objects; *predicates* express their properties and relations that hold between them. There are three kinds of *primary* (i.e. non-compound) predicates: ones built on top of primitive verbs and adjectives, predicates that state membership in the class denoted by some notion ("is a"-predicates), and predicates that express existence of a certain object related to the subject ("has"-predicates). Primary predicates may be negated and composed in conjunctions to form predicate units:

$$\begin{array}{rcl} \mathit{doesPredicate} & \rightarrow & [\,\texttt{does} \mid \texttt{do}\,]\ [\,\texttt{not}\,]\ \mathit{primVerb} \\ & | & [\,\texttt{does} \mid \texttt{do}\,]\ [\,\texttt{not}\,]\ [\,\texttt{pairwise}\,]\ \mathit{primVerbM} \\ & | & (\,\texttt{has} \mid \texttt{have}\,)\ \mathit{hasPredicate} \\ & | & (\,\texttt{is} \mid \texttt{are} \mid \texttt{be}\,)\ \mathit{isPredicate}\ \{\,\texttt{and}\ \mathit{isPredicate}\,\} \\ & | & (\,\texttt{is} \mid \texttt{are} \mid \texttt{be}\,)\ \mathit{is\_aPredicate}\ \{\,\texttt{and}\ \mathit{is\_aPredicate}\,\} \end{array}$$

$$\begin{array}{rcl} \mathit{isPredicate} & \rightarrow & [\,\texttt{not}\,]\ \mathit{primAdjective} \\ & | & [\,\texttt{not}\,]\ [\,\texttt{pairwise}\,]\ \mathit{primAdjectiveM} \\ & | & (\,\texttt{with} \mid \texttt{of} \mid \texttt{having}\,)\ \mathit{hasPredicate} \end{array}$$

$$\begin{array}{rcl} \mathit{is\_aPredicate} & \rightarrow & [\,\texttt{not}\,]\ [\,\texttt{a} \mid \texttt{an}\,]\ \mathit{classNoun} \\ & | & [\,\texttt{not}\,]\ \mathit{definiteTerm} \end{array}$$

$$\begin{array}{rcl} \mathit{hasPredicate} & \rightarrow & [\,\texttt{a} \mid \texttt{an} \mid \texttt{the}\,]\ \mathit{possessedNoun}\ \{\,\texttt{and}\ [\,\texttt{a} \mid \texttt{an} \mid \texttt{the}\,]\ \mathit{possessedNoun}\,\} \\ & | & \texttt{no}\ \mathit{possessedNoun} \end{array}$$

Primitive adjectives and verbs look as follows:

$$\begin{array}{rcl} \mathit{primVerb} & \rightarrow & (\,\texttt{converges} \mid \texttt{converge}\,) \\ & | & (\,\texttt{divides} \mid \texttt{divide}\,)\ \mathit{term} \\ & | & (\,\texttt{belongs} \mid \texttt{belong}\,)\ (\,\texttt{to}\,)\ \mathit{term} \\ & | & (\,\texttt{joins} \mid \texttt{join}\,)\ \mathit{term}\ (\,\texttt{with}\,)\ \mathit{term} \\ & | & \ldots \end{array}$$

$$\begin{array}{rcl} \mathit{primAdjective} & \rightarrow & (\,\texttt{prime}\,) \\ & | & (\,\texttt{dividing}\,)\ \mathit{term} \\ & | & (\,\texttt{equal}\,)\ (\,\texttt{to}\,)\ \mathit{term} \\ & | & (\,\texttt{less}\,)\ (\,\texttt{than}\,)\ \mathit{term} \\ & | & \ldots \end{array}$$

Primitive adjective `equal to` is preintroduced in ForTheL. Note that we can define a present continuous tense of a primitive verb as a primitive adjective. It should be done explicitly, with a separate synonym declaration (like "`Let x is dividing y stand for (x divides y).`"), since the system would not determine the corresponding form of the word correctly.

Primitive verbs and adjectives may automatically produce *multisubject primitives* or *m-primitives*. The rule of production is as follows. Take a primitive with at least one argument from *primVerb* or *primAdjective*. If the first argument place is preceded by a preposition token and is not succeeded by the token (`and`), then we produce a new primitive in *primVerbM* (resp., *primAdjectiveM*) by removing the first argument place together with the preceding token.

Thus, the primitive adjective $((\,\texttt{parallel}\,)\ (\,\texttt{to}\,)\ \textit{term})$ will produce the m-adjective $((\,\texttt{parallel}\,))$ added both to *primAdjectiveM* and to *primSimpleAdjectiveM*; and the verb primitive $((\,\texttt{commutes} \mid \texttt{commute}\,)\ (\,\texttt{with}\,)\ \textit{term}\ (\,\texttt{wrt}\,)\ \textit{term})$ will produce the m-verb $((\,\texttt{commutes} \mid \texttt{commute}\,)\ (\,\texttt{wrt}\,)\ \textit{term})$.

Remark. Predicate units containing m-primitives (*m-predicates*) ought to be used with several subjects: `X,Y,Z commute wrt N`, `parallel lines l,m`. If such a predicate is employed in an attribute, the notion unit must have several names. Thus, the units `an equal number X`, `a line N that is parallel`, `a set S such that S is disjoint` are forbidden.

New m-primitives are produced for all primitive adjectives and verbs. Symmetry and transitivity of multisubject predicates is taken for granted. Thus, the statement `A,B,C are equal` will be translated to the formula `A is equal to B and B is equal to C`. For non-transitive predicates, the optional adverb `pairwise` should be used. Then the statement `A,B,C are pairwise disjoint` will be correctly translated to `A is disjoint with B and A is disjoint with C and B is disjoint with C`.

$$\begin{aligned} \textit{primVerbM} &\to (\,\texttt{collides} \mid \texttt{collide}\,) \\ &\mid\ (\,\texttt{commutes} \mid \texttt{commute}\,)\ (\,\texttt{wrt}\,)\ \textit{term} \\ &\mid\ \dots \end{aligned}$$

$$\begin{aligned} \textit{primAdjectiveM} &\to (\,\texttt{equal}\,) \\ &\mid\ (\,\texttt{adjacent}\,)\ (\,\texttt{in}\,)\ \textit{term} \\ &\mid\ \dots \end{aligned}$$

With the help of *"is a"-predicates* we say that an object is either "described" by a class noun or equals to a definite term: `X is a natural number`, `X is the order of G`.

Remark. Class nouns in "is a"-predicates should have at most one name.

For the *"has"-predicates*, we also maintain a special group of primitives that are derived from primitive functions and notions:

$$\textit{possessedNoun} \to \{\,\textit{leftAttribute}\,\}\ \textit{primPossessedNoun}\ [\,\textit{rightAttribute}\,]$$

$$\begin{aligned} \textit{primPossessedNoun} &\to (\,\texttt{element} \mid \texttt{elements}\,)\ [\,\textit{names}\,] \\ &\mid\ (\,\texttt{solution} \mid \texttt{solutions}\,)\ [\,\textit{names}\,] \\ &\mid\ (\,\texttt{order} \mid \texttt{orders}\,)\ [\,\textit{names}\,] \\ &\mid\ \dots \end{aligned}$$

These primitives are produced automatically by the following rule. Take a noun primitive with at least one argument from *primClassNoun* or *primDefiniteNoun*. If the first argument place is preceded by the token (`of`) and is not succeeded by the token (`and`), then we produce a new possessed noun primitive by removing the first argument place together with the preceding (`of`) and adding the optional $[\,\textit{names}\,]$ if needed.

Thus, the class noun primitive

$$(\,\texttt{ambassador} \mid \texttt{ambassadors}\,)\ [\,\textit{names}\,]\ (\,\texttt{of}\,)\ \textit{term}\ (\,\texttt{in}\,)\ \textit{term}$$

produces the new possessed noun primitive

$$(\,\texttt{ambassador}\mid\texttt{ambassadors}\,)\ [\,\mathit{names}\,]\ (\,\texttt{in}\,)\ \mathit{term}$$

while the definite noun primitive `union of _ and _` does not produce any new primitive.

Here follow some examples of statements with "has"-predicates:

```
X has no elements

every set of a finite cardinality is finite

F has the domain D and the range R such that D is a subset of R
```

In the second statement the "has"-predicate appears as a right attribute of a notion.

1.3.5 Statements

Statements are ForTheL counterparts of logic formulas. First, we consider non-compound, *primary* statement units (we avoid calling them "atomic"). Such a statement may be of four kinds:

$$\begin{array}{rcl}
\mathit{primaryStatement} & \rightarrow & \mathit{simpleStatement} \\
& \mid & \mathit{thereIsStatement} \\
& \mid & [\,\texttt{we have}\,]\ \mathit{symbStatement} \\
& \mid & [\,\texttt{we have}\,]\ \mathit{constStatement}
\end{array}$$

Simple statements apply predicates to terms:

$$\mathit{simpleStatement} \rightarrow \mathit{terms}\ \mathit{doesPredicate}\ \{\,\texttt{and}\ \mathit{doesPredicate}\,\}$$

$$\mathit{terms} \rightarrow \mathit{term}\ \{\,(\,\texttt{,}\mid\texttt{and}\,)\ \mathit{term}\,\}$$

So called *"there is"-statements* affirm or deny nonemptiness of some notions' classes:

$$\begin{array}{rcl}
\mathit{thereIsStatement} & \rightarrow & \texttt{there}\ (\,\texttt{exists}\mid\texttt{exist}\,)\ \mathit{notions} \\
& \mid & \texttt{there}\ (\,\texttt{exists}\mid\texttt{exist}\,)\ \texttt{no}\ \mathit{notion}
\end{array}$$

$$\mathit{notions} \rightarrow [\,\texttt{a}\mid\texttt{an}\,]\ \mathit{notion}\ \{\,(\,\texttt{,}\mid\texttt{and}\,)\ [\,\texttt{a}\mid\texttt{an}\,]\ \mathit{notion}\,\}$$

Symbolic statements are composed in a traditional first-order syntax from symbolic relations and terms. Non-symbolic statements enclosed in parentheses are also allowed inside symbolic statements. In the rule for *symbStatement*, `<=>` stays for equivalence, `=>` for implication, `\/` for disjunction, and `/\` for conjunction. Productions for these propositional connectives are sorted by increase of precedence.

$$\begin{array}{rcl}
\mathit{symbStatement} & \rightarrow & \texttt{forall}\ \mathit{classRelation}\ \mathit{symbStatement} \\
& \mid & \texttt{exists}\ \mathit{classRelation}\ \mathit{symbStatement} \\
& \mid & \mathit{symbStatement}\ \texttt{<=>}\ \mathit{symbStatement} \\
& \mid & \mathit{symbStatement}\ \texttt{=>}\ \mathit{symbStatement} \\
& \mid & \mathit{symbStatement}\ \texttt{\textbackslash/}\ \mathit{symbStatement} \\
& \mid & \mathit{symbStatement}\ \texttt{/\textbackslash}\ \mathit{symbStatement} \\
& \mid & \texttt{not}\ \mathit{symbStatement} \\
& \mid & \texttt{(}\ \mathit{statement}\ \texttt{)} \\
& \mid & \mathit{primRelation}
\end{array}$$

$$\begin{array}{rcl}
\mathit{primRelation} & \to & \mathit{symbTerms}\ (\,\texttt{=}\,)\ \mathit{symbTerm} \\
 & | & \mathit{symbTerms}\ (\,\texttt{!=}\,)\ \mathit{symbTerm} \\
 & | & \mathit{symbTerms}\ (\,\texttt{-<-}\,)\ \mathit{symbTerm} \\
 & | & \mathit{symbTerms}\ (\,\texttt{:}\,)\ \mathit{symbTerm}\ (\,\texttt{->}\,)\ \mathit{symbTerm} \\
 & | & (\,\texttt{Nat}\,)\ (\,\texttt{(}\,)\ \mathit{symbTerm}\ (\,\texttt{)}\,) \\
 & | & \ldots \\
 & & \\
\mathit{symbTerms} & \to & \mathit{symbTerm}\ \{\ \texttt{,}\ \mathit{symbTerm}\,\}
\end{array}$$

Binary relations `=`and `!=` are preintroduced in ForTheL. They are synonyms for, respectively, the equality statement and its negation. The binary relation `-<-` also has a special semantics: it stands for an abstract well-founded relation and is used for induction proofs (1.5.3). This relation has to be introduced explicitly in the text, though.

The *constant statements* `thesis` and `contrary` denote, respectively, the current thesis statement and its negation. The notion of thesis will be explained in Section 2.3.2. The constant statement `contradiction` stands for $\bot$, the logical falsity.

$$\begin{array}{rcl}
\mathit{constStatement} & \to & [\,\texttt{the}\,]\ \texttt{thesis} \\
 & | & [\,\texttt{the}\,]\ \texttt{contrary} \\
 & | & [\,\texttt{a}\,|\,\texttt{an}\,]\ \texttt{contradiction}
\end{array}$$

Primary statements are composed with prepositions and conjunctions as follows:

$$\begin{array}{rcl}
\mathit{statement} & \to & \mathit{headStatement}\ |\ \mathit{chainStatement} \\
 & & \\
\mathit{headStatement} & \to & \texttt{for}\ \mathit{quantifiedNotion}\ \{\,\texttt{and}\ \mathit{quantifiedNotion}\,\}\ \mathit{statement} \\
 & | & \texttt{if}\ \mathit{statement}\ \texttt{then}\ \mathit{statement} \\
 & | & \texttt{it is wrong that}\ \mathit{statement} \\
 & & \\
\mathit{chainStatement} & \to & \mathit{andChain}\ [\,\texttt{and}\ \mathit{headStatement}\,] \\
 & | & \mathit{orChain}\ [\,\texttt{or}\ \mathit{headStatement}\,] \\
 & | & (\,\mathit{andChain}\,|\,\mathit{orChain}\,)\ \texttt{iff}\ \mathit{statement} \\
 & & \\
\mathit{andChain} & \to & \mathit{primaryStatement}\ \{\,\texttt{and}\ \mathit{primaryStatement}\,\} \\
 & & \\
\mathit{orChain} & \to & \mathit{primaryStatement}\ \{\,\texttt{or}\ \mathit{primaryStatement}\,\}
\end{array}$$

1.3.6 Special statements and synonym declarations

Special ForTheL statements are used in definitions and signature extensions (see Section 1.5.1). Their syntax is described by the following productions:

$$\begin{array}{rcl}
\mathit{defStatement} & \to & \mathit{notionDef}\ |\ \mathit{functionDef}\ |\ \mathit{predicateDef} \\
 & & \\
\mathit{sigStatement} & \to & \mathit{notionSig}\ |\ \mathit{functionSig}\ |\ \mathit{predicateSig} \\
 & & \\
\mathit{notionDef} & \to & \mathit{notionHead}\ \texttt{is}\ [\,\texttt{a}\,|\,\texttt{an}\,]\ \mathit{classNoun} \\
 & & \\
\mathit{notionSig} & \to & \mathit{notionHead}\ \texttt{is}\ [\,\texttt{a}\,|\,\texttt{an}\,]\ (\,\mathit{classNoun}\,|\,\texttt{notion}\,) \\
 & & \\
\mathit{functionDef} & \to & \mathit{functionHead}\ \texttt{is}\ [\,\texttt{equal to}\,]\ \mathit{plainTerm} \\
 & | & \mathit{functionHead}\ \texttt{is}\ [\,\texttt{equal to}\,]\ [\,\texttt{a}\,|\,\texttt{an}\,|\,\texttt{the}\,]\ \mathit{classNoun} \\
 & & \\
\mathit{functionSig} & \to & \mathit{functionHead}\ \texttt{is}\ [\,\texttt{a}\,|\,\texttt{an}\,]\ (\,\mathit{classNoun}\,|\,\texttt{term}\,|\,\texttt{constant}\,)
\end{array}$$

$$\begin{array}{rcl}
\mathit{predicateDef} & \rightarrow & \mathit{predicateHead}\ \texttt{iff}\ \mathit{statement} \\
\\
\mathit{predicateSig} & \rightarrow & \mathit{predicateHead}\ \texttt{implies}\ \mathit{statement} \\
 & | & \mathit{predicateHead}\ \texttt{is}\ [\,\texttt{a}\,|\,\texttt{an}\,]\ \texttt{atom}
\end{array}$$

In every special statement S, there is a so called *main junction* which may be connection `is`, equality, equivalence, or implication. The unit to the left of the main junction is called the *head unit* of the statement (and the corresponding section). The syntax of the head is as follows:

$$\begin{array}{rcl}
\mathit{notionHead} & \rightarrow & [\,\texttt{a}\,|\,\texttt{an}\,]\ \mathit{primClassNoun} \\
 & | & \mathit{notionPattern} \\
\\
\mathit{notionPattern} & \rightarrow & (\texttt{a}\,|\,\texttt{an})\ \mathit{pattern} \\
\\
\mathit{functionHead} & \rightarrow & [\,\texttt{the}\,]\ \mathit{primDefiniteNoun} \\
 & | & \mathit{primInfixOperator} \\
 & | & \mathit{primPrefixOperator} \\
 & | & \mathit{primPostfixOperator} \\
 & | & \mathit{functionPattern} \\
\\
\mathit{functionPattern} & \rightarrow & \texttt{the}\ \mathit{pattern} \\
 & | & \mathit{symbPattern} \\
\\
\mathit{predicateHead} & \rightarrow & \mathit{variable}\ \texttt{is}\ \mathit{primAdjective} \\
 & | & \mathit{variable}\ \texttt{,}\ \mathit{variable}\ \texttt{are}\ \mathit{primAdjectiveM} \\
 & | & \mathit{variable}\ \mathit{primVerb} \\
 & | & \mathit{variable}\ \texttt{,}\ \mathit{variable}\ \mathit{primVerbM} \\
 & | & \mathit{primRelation} \\
 & | & \mathit{predicatePattern} \\
\\
\mathit{predicatePattern} & \rightarrow & \mathit{variable}\ \texttt{is}\ \mathit{pattern} \\
 & | & \mathit{variable}\ \mathit{pattern} \\
 & | & \mathit{symbPattern}
\end{array}$$

A special statement S is well-formed if the following conditions hold:

- in the head unit, all the terms in argument places are variables (head unit is flat);
- no variable occurs twice in the head unit (head unit is linear);
- each variable that occurs free in S (see 1.4 for definition) occurs in the head unit;
- the primitive in the head unit should not be already introduced as a synonym (see below), though it may be already introduced by some definition or signature extension above the current one (this is why we allow not only *pattern*'s in the head, but also established primitives);
- if S introduces a notion then the head notion unit has at most one name;
- a class noun to the right of the main junction has at most one name.

Unless the head unit is based on some syntactic primitive which is already in the current vocabulary, the parser will construct and add a new primitive. In Section 1.3.1, we explained how patterns are converted into base syntactic primitives.

Another possibility to extend the current vocabulary is to write a *synonym declaration*:

$$synonym \;\rightarrow\; notionSyn \mid functionSyn \mid predicateSyn$$

$$notionSyn \;\rightarrow\; \texttt{let}\; notionPattern\; (\texttt{stand for} \mid \texttt{denote})\; [\,\texttt{a} \mid \texttt{an}\,]\; classNoun\;\; \texttt{.}$$

$$functionSyn \;\rightarrow\; \texttt{let}\; functionPattern\; (\texttt{stand for} \mid \texttt{denote})\; plainTerm\;\; \texttt{.}$$

$$predicateSyn \;\rightarrow\; \texttt{let}\; predicatePattern\; (\texttt{stand for} \mid \texttt{denote})\; statement\;\; \texttt{.}$$

The pattern to the left of the main junction (`stand for` | `denote`) is still called the *head unit*. The unit to the right is called *target*. As in special statements, the head in a synonym declaration must be flat and linear, and must contain all the variables which occurs free in the target. Unlike special statements, the head in a synonym declaration cannot be an already introduced primitive.

Note that synonym declarations are not statements but instructions to the parser, just like token groups (see Section 1.3.1). Roughly, synonyms relate to defined symbols as preprocessor macros relate to subroutines in a programming language.

1.3.7 Variable description

We say that a statement S *describes* a variable v whenever S implies that v belongs to a class denoted by some notion. The following mutually recursive definitions give a formal explication for this.

A term t is called *positive* whenever t is not a quantified notion of the form `no` *notion* and the arguments of t are positive terms.

A syntactic primitive I is called *descriptive* if one of the following conditions holds:

- I is the preintroduced adjective `equal to` arg_1;
- I is introduced as a predicate synonym; the pattern in the head of the synonym declaration begins with some variable v and the target statement describes v (see below).

A predicate unit P is *descriptive* if P is non-negated and one of the following holds:

- P is an "is a"-predicate built upon a positive definite term;
- P is an "is a"-predicate built upon a noun notion with positive terms in arguments;
- P is built upon a descriptive adjective or verb primitive with positive terms in arguments;
- P is composed of several primary predicates one of which is descriptive.

Finally, a statement S *describes* a variable v if one of the following conditions holds:

- S is a symbolic statement built upon a descriptive relation primitive; the first argument of S is a sequence of variables containing v;
- S is a simple statement such that the subject of S is a sequence of variables containing v and the predicate of S is descriptive;
- S is a conjunction of several statements one of which describes v.

The recursive descent in the definitions above is terminating, since a syntactic primitive introduced as a synonym can occur neither in the target unit of the corresponding synonym declaration nor above it in the text.

For example, after the following token group and two synonym declarations

```
[belongs/belong]

Let x belongs to y stand for (x is an element of y).

Let x << y stand for (x belongs to y).
```

the statement

```
u,v belong to some finite (s << W) and Q is a subset of no set
```

is well-formed and describes the variables `u`, `v` but not the variable `Q`. Note that the introduction of `<<` produces both a base symbolic relation primitive and a derived class relation primitive. This is because the target statement `x belongs to y` describes the variable `x`.

1.4 Formula image of a statement

Parsing a string of lexemes, we identify parts of the string with the nonterminals of the grammar in accordance with the grammar rules. A substring that correspond to some nonterminal N in the overall parsing will be called *an occurrence* of N. Note that the context of a substring is significant. Consider two statements:

```
some natural number N divides 1
for some natural number N (N divides 1)
```

The string `some natural number N` is an occurrence of *term* in the first statement but not in the second one. Also, the string `natural number` is not an occurrence of *notion* in the both statements (though it is a valid notion unit by itself), because the actual occurrences on *notion* also include the name `N`.

Using the correspondence between substrings and nonterminals, we translate a statement unit S to a certain first-order formula $|S|$, called *the formula image* of S. To this purpose we subsequently apply the transformation rules described in the following subsections. Each rule must be applied as many times as possible before the next rule could be applied for the first time.

We define the set of free variables of a statement S as the set of free variables of the formula image of S: $\mathcal{FV}(S) = \mathcal{FV}(|S|)$.

1.4.1 Desugaring syntax

First of all, we somewhat normalize our syntax in order to simplify formulation of the subsequent transformation rules.

1. For each occurrence of *primClassNoun* or *primPossessedNoun*, where the subunit $[\,names\,]$ is empty (i.e. no name was given), we put a single fresh variable into the corresponding position.

2. We split "and"-chains of notions in quantified statements. The following rule is applied to occurrences of *headStatement*:

   $\texttt{for}\ (quantifiedNotion)_O$
   $\quad\texttt{and}\ (quantifiedNotion\ \{\,\texttt{and}\ quantifiedNotion\,\})_T\ \ (statement)_S$
   $\Rightarrow\ \ \texttt{for}\ O\ \texttt{for}\ T\ \ S$

3. Articles `a`, `an`, `the` and the prefix `we have` are removed. Negation `it is wrong that` is replaced with `not`. The special statement `contrary` is replaced with `not thesis`. Verbs `be`, `are`, `do`, `have`, `exist` are converted to the third person singular. Quantifier words `all`, `each`, `any` are replaced with `every`. In occurrences of *terms* and *notions*, `and`'s are replaced with commas.

We denote with $\mathbf{N}$ any of the nonterminals *primClassNoun*, *primPossessedNoun*, or *primClassRelation*[1]. For each occurrence O of $\mathbf{N}$, we denote with $\mathbf{n}(O)$ the list of variables contained in the subunit $[\,names\,]$ of the corresponding primitive. We extend this notation to the nonterminals *notion*, *classNoun*, *classRelation*, *quantifiedNotion*, and *possessedNoun* in an obvious manner. As the rule (1.4.1(1)) has been exhaustively applied, the list of names is now nonempty for all occurrences of $\mathbf{N}$ in the unit under consideration.

[1]thus, $\mathbf{N}$ is a kind of a meta-nonterminal

1.4.2 Resolving quantified notions I

The following rules remove quantified notions from the occurrences of *term* in subjects of simple statements and in quantifiers. Roughly, every quantified notion O that occurs as such a term unit is replaced with its name list $\mathbf{n}(O)$ and the appropriate quantifier is set over the statement.

Consider two arbitrary occurrences S and O. We call O a *proper occurrence* in S if O occurs properly inside S. We call O a *native occurrence* in S if O is proper in S and does not belong to any occurrence of *rightAttribute* in S.

For example, the occurrences X, Y and U of *quantifiedNotion* are native in the following simple statement, whereas the occurrence V is not native:

$$\texttt{(every member M of (some committee C)}_Y\texttt{)}_X\texttt{ declares}$$
$$\texttt{(some opinion O such that (every member N of C)}_V\texttt{ supports O)}_U$$

Two rules below apply to occurrences of *simpleStatement* and *headStatement*, respectively. In premises, O is the leftmost native occurrence of *quantifiedNotion* in S. In conclusions, O is replaced in S with $\mathbf{n}(O)$:

1. $(terms\lceil(quantifiedNotion)_O\rfloor)_S\ \ (doesPredicate\ \{\ \texttt{and}\ doesPredicate\ \})_T$
   $\Rightarrow\quad \texttt{for}\ O\ \ S\lceil O \to \mathbf{n}(O)\rfloor\ \ T$

2. $\texttt{for}\ \ (quantifiedNotion\lceil(quantifiedNotion)_O\rfloor)_S\ \ (statement)_T$
   $\Rightarrow\quad \texttt{for}\ O\ \ \texttt{for}\ S\lceil O \to \mathbf{n}(O)\rfloor\ \ T$

For the example above, these rules produce the following chain of transformations (we add parentheses for readability):

```
every member M of some committee C declares some opinion O
          such that every member N of C supports O
```
$\Downarrow$ (1.4.2(1))
```
for (every member M of some committee C) M declares some opinion O
          such that every member N of C supports O
```
$\Downarrow$ (1.4.2(1))
```
for (every member M of some committee C) M declares some opinion O
          such that for (every member N of C) N supports O
```
$\Downarrow$ (1.4.2(2))
```
for (some committee C) for (every member M of C) M declares
some opinion O such that for (every member N of C) N supports O
```

1.4.3 Unifying attributes

Now we transform attribute subunits to a common unified form. The following rules apply to occurrences of the nonterminals *notion*, *classNoun*, and *possessedNoun*.

1. $(\{\ leftAttribute\ \})_L\ \ (\mathbf{N})_O\ \ (isPredicate\ \{\ \texttt{and}\ isPredicate\ \})_A$
   $\Rightarrow\quad L\ O\ \texttt{such that}\ \mathbf{n}(O)\ \texttt{is}\ A$

2. $(\{\ leftAttribute\ \})_L\ \ (\mathbf{N})_O\ \ \texttt{that}\ \ (doesPredicate\ \{\ \texttt{and}\ doesPredicate\ \})_V$
   $\Rightarrow\quad L\ O\ \texttt{such that}\ \mathbf{n}(O)\ V$

3. $(\{\ leftAttribute\ \})_L\ \ (leftAttribute)_A\ \ (\mathbf{N})_O\ \ [\ \texttt{such that}\ \ (statement)_S\ ]$
   $\Rightarrow\quad L\ O\ \texttt{such that}\ \mathbf{n}(O)\ \texttt{is}\ A\ [\ \texttt{and}\ S\ ]$

The purpose of these rules is to move all the predicate units to simple statements. As an example, consider the following chain of transformations:

```
prime natural number X that divides N
```
$\Downarrow$ (1.4.3(2))
```
prime natural number X such that X divides N
```
$\Downarrow$ (1.4.3(3))

`prime number X such that X is natural and X divides N`

$\Downarrow$ (1.4.3(3))

`number X such that X is prime and X is natural and X divides N`

1.4.4 Splitting compound predicates

Now we remove conjunctions from simple statements. The following rules apply to occurrences of *simpleStatement*:

1. $(\textit{terms})_O$ $(\textit{doesPredicate})_P$ `and` $(\textit{doesPredicate}\ \{$ `and` $\textit{doesPredicate}\ \})_T$
   $\Rightarrow$ $O\ P$ `and` $O\ T$
2. $(\textit{terms})_O$ `is` $(\textit{isPredicate})_P$ `and` $(\textit{isPredicate}\ \{$ `and` $\textit{isPredicate}\ \})_T$
   $\Rightarrow$ O `is` P `and` O `is` T
3. $(\textit{terms})_O$ `is` $(\textit{is\_aPredicate})_P$ `and` $(\textit{is\_aPredicate}\ \{$ `and` $\textit{is\_aPredicate}\ \})_T$
   $\Rightarrow$ O `is` P `and` O `is` T

Note that in the above rules, all the terms in $(\textit{terms})_O$ are plain.

1.4.5 Elimination of "there exists"

Below, $\bar{O}$ denotes an occurrence O of *notion*, *primClassNoun*, or *primPossessedNoun* with the list of names $\mathbf{n}(O)$ reset to the empty list. Here, we transform existential atomic statements to quantified statements with unrestricted quantifiers. The following rules apply to occurrences of *thereIsStatement*:

1. `there exists` $(\textit{classRelation})_O$ $[$ `such that` $(\textit{statement})_S\ ]$ $[$ `,` $(\textit{notions})_T\ ]$
   $\Rightarrow$ `for some` $\mathbf{n}(O)$ `(`O $[$ `and` $S\ ]$ $[$ `and there exists` $T\ ]$ `)`
2. `there exists no` $(\textit{classRelation})_O$ $[$ `such that` $(\textit{statement})_S\ ]$
   $\Rightarrow$ `for every` $\mathbf{n}(O)$ `(not (`O $[$ `and` $S\ ]$ `))`
3. `there exists` $(\textit{notion})_O$ $[$ `,` $(\textit{notions})_T\ ]$
   $\Rightarrow$ `for some` $\mathbf{n}(O)$ `(`$\mathbf{n}(O)$ `is` $\bar{O}$ $[$ `and there exists` $T\ ]$ `)`
4. `there exists no` $(\textit{notion})_O$
   $\Rightarrow$ `for every` $\mathbf{n}(O)$ `(`$\mathbf{n}(O)$ `is not` $\bar{O}$`)`

Let us illustrate these rules with the following transformations:

`there exists number E and prime divisors G,H of E`

$\Downarrow$ (1.4.1(3),1.4.3(3))

`there exists number E, divisors G,H of E such that G,H is prime`

$\Downarrow$ (1.4.5(3))

```
for some E (E is a number and there exists
divisors G,H of E such that G,H is prime)
```

$\Downarrow$ (1.4.5(3))

```
for some E (E is a number and for some G,H
(G,H is divisors of E such that G,H is prime))
```

1.4.6 Resolving quantified notions II

The following group of rules removes quantified notions from the rest of the occurrences of *term*. Like the rules (1.4.2(1-2)) above, the following transformation rules apply to occurrences of *simpleStatement* and *headStatement*, respectively. In premises, O is the leftmost native occurrence of *quantifiedNotion* in S. In conclusions, O is replaced in S with $\mathbf{n}(O)$:

1. $(\textit{terms})_T$ $(\textit{doesPredicate}\lceil(\textit{quantifiedNotion})_O\rfloor)_S$
   $\Rightarrow$ `for` $O\ T\ S\lceil O \rightarrow \mathbf{n}(O)\rfloor$

2. `for` $(quantifiedNotion\lceil(quantifiedNotion)_O\rfloor)_S$ $(statement)_T$
   $\Rightarrow$ `for` O `for` $S\lceil O \to \mathbf{n}(O)\rfloor$ T

The example above can thus be continued as follows:

```
     for (some committee C) for (every member M of C) M declares
   some opinion O such that for (every member N of C) N supports O
```
$$\Downarrow (1.4.6(1))$$
```
  for (some committee C) for (every member M of C) for (some opinion O
     such that for (every member N of C) N supports O) M declares O
```

Remark. We apply the rules (1.4.5(1-4)) before the rules (1.4.6(1-2)), or else the statement `there exists a subset of every set` would be transformed to `for every set X there exists a subset Y of X`, while the intended meaning is `for some Y for every set X (Y is a subset of X)`.

1.4.7 Handling quantifiers

The previous group of rules transformed all the term units in our statement to plain terms. Here we transform restricted ForTheL quantifiers to unrestricted first-order quantifiers over implications and conjunctions. The transformation rules below apply to occurrences of *headStatement* (1.4.7(1-6)) and *symbStatement* (1.4.7(7-8)):

1. `for` [`(`] `every` $(classRelation)_O$ [`such that` $(statement)_T$] [`)`] $(statement)_S$
   $\Rightarrow$ `for every` $\mathbf{n}(O)$ `(if` O [`and` T] `then` S`)`

2. `for` [`(`] `some` $(classRelation)_O$ [`such that` $(statement)_T$] [`)`] $(statement)_S$
   $\Rightarrow$ `for some` $\mathbf{n}(O)$ `(`O [`and` T] `and` S`)`

3. `for` [`(`] `no` $(classRelation)_O$ [`such that` $(statement)_T$] [`)`] $(statement)_S$
   $\Rightarrow$ `for every` $\mathbf{n}(O)$ `(if` O [`and` T] `then (not` S`))`

4. `for` [`(`] `every` $(notion)_O$ [`)`] $(statement)_S$
   $\Rightarrow$ `for every` $\mathbf{n}(O)$ `(if` $\mathbf{n}(O)$ `is` $\bar{O}$ `then` S`)`

5. `for` [`(`] `some` $(notion)_O$ [`)`] $(statement)_S$
   $\Rightarrow$ `for some` $\mathbf{n}(O)$ `(`$\mathbf{n}(O)$ `is` $\bar{O}$ `and` S`)`

6. `for` [`(`] `no` $(notion)_O$ [`)`] $(statement)_S$
   $\Rightarrow$ `for every` $\mathbf{n}(O)$ `(if` $\mathbf{n}(O)$ `is` $\bar{O}$ `then (not` S`))`

7. `forall` $(classRelation)_O$ $(symbStatement)_S$
   $\Rightarrow$ `forall` $\mathbf{n}(O)$ `(`O `=>` S`)`

8. `exists` $(classRelation)_O$ $(symbStatement)_S$
   $\Rightarrow$ `exists` $\mathbf{n}(O)$ `(`O `/\`S`)`

1.4.8 Eliminating "has"-predicates

Given an occurrence O of *primPossessedNoun* and an occurrence N of *term*, $O\lceil N\rfloor$ will stand for the original class-noun or definite noun primitive with N put back in the first argument place. For example, if O is `subset` and N is `S` then $O\lceil N\rfloor$ `is subset of S`.

The following rules apply to occurrences of *simpleStatement*:

1. $(terms)_N$ `is` (`with` | `of` | `having`) $(hasPredicate)_S$
   $\Rightarrow$ N `has` S

2. $(term)_N$ `,` $(term$ { `,` $term$ }$)_T$ `has` $(hasPredicate)_S$
   $\Rightarrow$ N `has` S `and` T `has` S

3. $(term)_N$ `has` $(primPossessedNoun)_O$ [`such that` $(statement)_S$]
   [`and` $(possessedNoun$ { `and` $possessedNoun$ }$)_T$]
   $\Rightarrow$ `for some` $\mathbf{n}(O)$ `(`$\mathbf{n}(O)$ `is` $\bar{O}\lceil N \rfloor$ [`and` S] [`and` N `has` T] `)`

4. $(term)_N$ `has no` $(primPossessedNoun)_O$ [`such that` $(statement)_S$]
   $\Rightarrow$ `for every` $\mathbf{n}(O)$ `(not (`$\mathbf{n}(O)$ `is` $\bar{O}\lceil N \rfloor$ [`and` S] `))`

These rules are very similar to those ones handling existential atomic statements. We can illustrate them with the following chain of transformations:

```
              A has a wife W and a salary not enough for W
                  ⇓ (1.4.1(1), 1.4.1(3),1.4.3(1))
         A has wife W and salary S such that S is not enough for W
                          ⇓ (1.4.8(3))
                 for some W (W is wife of A and
           A has salary S such that S is not enough for W)
                          ⇓ (1.4.8(3))
                 for some W (W is wife of A and
        for some S (S is salary of A and S is not enough for W))
```

Here, `wife of _` is a class-noun primitive that forms a notion, and `salary of _` is a definite noun primitive that forms a function. Thus, in the final statement we have occurrences of *is_aPredicate* of the both kinds.

1.4.9 Handling "is a"-predicates

Consider an occurrence of *classNoun* of the form

$$((primClassNoun)_O \ \texttt{such that} \ (statement)_S)$$

According to the grammar, $\mathbf{n}(O)$ contains at most one name and this property is preserved by translation rules. Let N be an occurrence of *terms*. If $\mathbf{n}(O)$ is empty then the expression $S\lceil \mathbf{n}(O) \to N \rfloor$ is just S. If $\mathbf{n}(O)$ contains a name v then $S\lceil \mathbf{n}(O) \to N \rfloor$ is the result of substitution of N in place of each occurrence of v in S.

The following rules apply to occurrences of *simpleStatement*:

1. $(term)_N$ `,` $(term$ { `,` $term$ }$)_T$ `is` $([$ `not` $])_C$ $(classNoun)_O$
   $\Rightarrow$ N `is` C O `and` T `is` C O
   (if O does not contain in the attribute any m-predicate applied to $\mathbf{n}(O)$)

2. $(terms)_N$ `is` $([$ `not` $])_C$ $(primClassNoun)_O$ [`such that` $(statement)_S$]
   $\Rightarrow$ `(`C `(`N `is` $\bar{O}$ [`and` $S\lceil \mathbf{n}(O) \to N \rfloor$] `))`

3. $(term)_N$ `,` $(term$ { `,` $term$ }$)_T$ `is` $(primClassNoun)_O$
   $\Rightarrow$ N `is` O `and` T `is` O

4. $(terms)_N$ `is` $([$ `not` $])_C$ $(definiteTerm)_O$
   $\Rightarrow$ N `is` C `equal to` O

To see, why the condition in the first rule is needed, consider two transformation chains. In the first example, an "is a"-predicate does not contain m-primitives in attributes. Therefore, it applies independently to each subject:

```
                    A,B are not natural numbers
                  ⇓ (1.4.1(1),1.4.1(3),1.4.3(3))
               A,B is not number X such that X is natural
                          ⇓ (1.4.9(1))
                 A is not number X such that X is natural
              and B is not number X such that X is natural
                          ⇓ (1.4.9(2))
                  (not (A is number and A is natural))
              and (not (B is number and B is natural))
```

In the second example, an "is a"-predicate does use m-primitives:

```
A,B are not equal numbers
⇓ (1.4.1(1),1.4.1(3),1.4.3(3))
A,B is not number X such that X is equal
⇓ (1.4.9(2))
not (A,B is number and A,B is equal)
⇓ (1.4.9(3))
not (A is number and B is number and A,B is equal)
```

Also note that the occurrence N in the rule (1.4.9(2)) consists of more than one term only if S contains m-predicates applied to $\mathbf{n}(O)$. In this case, the original noun notion unit had no name at all and $\mathbf{n}(O)$ was introduced by the rule (1.4.1(1)). Therefore, $\mathbf{n}(O)$ occurs in S only as a subject of a simple statement (due to the rules (1.4.3(1-3))) and hence the expression $S\lceil \mathbf{n}(O) \to N \rfloor$ is well-formed.

1.4.10 Eliminating m-predicates

Given an occurrence S of an m-primitive and an occurrence N of *term*, we denote with $S\lceil N \rfloor$ the original verb or adjective primitive with N put back in the first argument place. For example, if S is `parallel` and N is `X` then $S\lceil N \rfloor$ is `parallel to X`.

The following rules apply to occurrences of *simpleStatement*:

1. $(\mathit{term})_N$ [`does`] [`not`] [`pairwise`] $\mathit{primVerbM}$ $\Rightarrow$ syntax error

2. $(\mathit{term})_N$ `is` [`not`] [`pairwise`] $\mathit{primAdjectiveM}$ $\Rightarrow$ syntax error

3. $(\mathit{terms})_N$ [`does`] `not` ([`pairwise`])$_P$ $(\mathit{primVerbM})_S$
   $\Rightarrow$ `not` (N P S)

4. $(\mathit{terms})_N$ `is not` ([`pairwise`])$_P$ $(\mathit{primAdjectiveM})_S$
   $\Rightarrow$ `not` (N `is` P S)

5. $(\mathit{term})_{N_1}$, … , $(\mathit{term})_{N_n}$ [`does`] `pairwise` $(\mathit{primVerbM})_S$
   $\Rightarrow$ (N_1 $S\lceil N_2 \rfloor$) `and` … (N_i $S\lceil N_{i+j} \rfloor$) … `and` (N_{n-1} $S\lceil N_n \rfloor$)

6. $(\mathit{term})_{N_1}$, … , $(\mathit{term})_{N_n}$ [`does`] $(\mathit{primVerbM})_S$
   $\Rightarrow$ (N_1 $S\lceil N_2 \rfloor$) `and` … (N_i $S\lceil N_{i+1} \rfloor$) … `and` (N_{n-1} $S\lceil N_n \rfloor$)

7. $(\mathit{term})_{N_1}$, … , $(\mathit{term})_{N_n}$ `is pairwise` $(\mathit{primAdjectiveM})_S$
   $\Rightarrow$ (N_1 `is` $S\lceil N_2 \rfloor$) `and` … (N_i `is` $S\lceil N_{i+j} \rfloor$) … `and` (N_{n-1} `is` $S\lceil N_n \rfloor$)

8. $(\mathit{term})_{N_1}$, … , $(\mathit{term})_{N_n}$ `is` $(\mathit{primAdjectiveM})_S$
   $\Rightarrow$ (N_1 `is` $S\lceil N_2 \rfloor$) `and` … (N_i `is` $S\lceil N_{i+1} \rfloor$) … `and` (N_{n-1} `is` $S\lceil N_n \rfloor$)

1.4.11 Handling ordinary predicates and symbolic relations

Then we transform the rest of atoms containing *terms* or *symbTerms*. The rules below apply to occurrences of *simpleStatement* (1.4.11(1-2)) and *primRelation* (1.4.11(3)):

1. $(\mathit{term})_N$ [, $(\mathit{terms})_T$] [`does`] ([`not`])$_C$ $(\mathit{primVerb})_S$
   $\Rightarrow$ (C (N S)) [`and` T C S]

2. $(\mathit{term})_N$ [, $(\mathit{terms})_T$] `is` ([`not`])$_C$ $(\mathit{primAdjective})_S$
   $\Rightarrow$ (C (N `is` S)) [`and` T `is` C S]

3. $(\mathit{primRelation}\lceil(\mathit{symbTerm})_N$, $(\mathit{symbTerms})_T\rfloor)_S$
   $\Rightarrow$ $S\lceil N \rfloor$ `/\`$S\lceil T \rfloor$

1.4.12 Resolving synonyms

The resulting unit S is essentially a first-order formula with atoms of six possible forms:

term [`does`] *primVerb*	*term* `is` *primAdjective*	`contradiction`
term `is` *primClassNoun*	*primRelation*	`thesis`

All the terms in this formula are composed of definite nouns and symbolic operators and hence are plain. Converting n-ary adjectives and verbs to predicate symbols of the arity $(n+1)$, the word `contradiction` to $\perp$, and the word `thesis` to $\mathfrak{T}$, we will obtain a well-formed formula in the language described in Section 2.2.1.

The following rule applies to those primitives in S which were introduced as synonyms. We replace every such a primitive with the appropriate instantiation of the target unit and reapply the whole translation procedure to the resulting statement. This recursion never falls in an endless loop, since synonym declarations are linearly ordered in the ForTheL text.

For example, the synonym declarations:

```
Let a relation on D stand for a relation with the domain equal to D.
Let U ** V stand for the intersection of U with V.
```

induce the following transformation chain:

```
                X is relation on A ** B ** C
                       ⇓ (1.4.12)
             X is a relation with the domain equal to
       the intersection of A with the intersection of B with C
                       ⇓ (1.4.1–1.4.12)
             X is relation and domain of X is equal to
           intersection of A with intersection of B with C
```

The transformation is now finished with a well-formed first-order formula in the result. An example of a full transformation chain for a ForTheL statement is given in Figure 1.1.

1.4.13 Handling special statements

Instead of integrating the following rules into the common transformation sequence presented above, we just transform a given occurrence of *defStatement* or *sigStatement* into an ordinary statement unit and then apply the general procedure.

For an occurrence of *defStatement*, the formula image is calculated according to the following equations:

$$\begin{aligned}
|\,(\mathit{notionHead})_H \ \texttt{is}\ (\mathit{notion})_B\,| &= \forall v\,|\,v\ \texttt{is}\ \bar{H}\ \texttt{iff}\ v\ \texttt{is}\ B\,| \\
|\,(\mathit{functionHead})_H \ \texttt{is equal to}\ (\mathit{plainTerm})_B\,| &= \forall v\,|\,v\ \texttt{is equal to}\ \bar{H}\ \texttt{iff}\ v\ \texttt{is}\ B\,| \\
|\,(\mathit{functionHead})_H \ \texttt{is equal to}\ (\mathit{classNoun})_B\,| &= \forall v\,|\,v\ \texttt{is equal to}\ \bar{H}\ \texttt{iff}\ v\ \texttt{is}\ B\,| \\
|\,(\mathit{predicateHead})_H \ \texttt{iff}\ (\mathit{statement})_B\,| &= |\,H\ \texttt{iff}\ B\,|
\end{aligned}$$

The formula image of a *sigStatement* is calculated as follows:

$$\begin{aligned}
|\,(\mathit{notionHead})_H \ \texttt{is}\ (\mathit{notion})_B\,| &= \forall v\,|\,\texttt{if}\ v\ \texttt{is}\ \bar{H}\ \texttt{then}\ v\ \texttt{is}\ B\,| \\
|\,(\mathit{notionHead})_H \ \texttt{is notion}\,| &= \forall v\,|\,\texttt{if}\ v\ \texttt{is}\ \bar{H}\ \texttt{then}\ \top\,| \\
|\,(\mathit{functionHead})_H \ \texttt{is}\ (\mathit{notion})_B\,| &= \forall v\,|\,\texttt{if}\ v\ \texttt{is equal to}\ \bar{H}\ \texttt{then}\ v\ \texttt{is}\ B\,| \\
|\,(\mathit{functionHead})_H \ \texttt{is}\ (\texttt{term}\,|\,\texttt{constant})\,| &= \forall v\,|\,\texttt{if}\ v\ \texttt{is equal to}\ \bar{H}\ \texttt{then}\ \top\,| \\
|\,(\mathit{predicateHead})_H \ \texttt{implies}\ (\mathit{statement})_B\,| &= |\,\texttt{if}\ H\ \texttt{then}\ B\,| \\
|\,(\mathit{predicateHead})_H \ \texttt{is}\ (\texttt{atom})_B\,| &= |\,\texttt{if}\ H\ \texttt{then}\ \top\,|
\end{aligned}$$

Everywhere above, the variable v is either $\mathbf{n}(H)$ or a fresh variable if $\mathbf{n}(H)$ is empty. Also, $\top$ denotes the logical truth.

for all nonequal points A,B there exists a straight line L such that A and B lie on L and any straight line that contains A and contains B is L

$\Downarrow$ (1.4.1(1),1.4.1(3))

for every nonequal points A,B there exists straight line L such that A,B lies on L and every straight line M that contains A and contains B is L

$\Downarrow$ (1.4.2(1))

for every nonequal points A,B there exists straight line L such that A,B lies on L and for every straight line M that contains A and contains B M is L

$\Downarrow$ (1.4.3(2))

for every nonequal points A,B there exists straight line L such that A,B lies on L and for every straight line M such that M contains A and contains B M is L

$\Downarrow$ (1.4.3(3))

for every points A,B such that A,B is nonequal there exists line L such that L is straight and A,B lies on L and for every line M such that M is straight and M contains A and contains B M is L

$\Downarrow$ (1.4.4(1))

for every points A,B such that A,B is nonequal there exists line L such that L is straight and A,B lies on L and for every line M such that M is straight and M contains A and M contains B M is L

$\Downarrow$ (1.4.5(3))

for every points A,B such that A,B is nonequal for some L (L is line such that L is straight and A,B lies on L and for every line M such that M is straight and M contains A and M contains B M is L)

$\Downarrow$ (1.4.7(4),1.4.7(5))

for every A,B (if A,B is points such that A,B is nonequal then for some L (L is line such that L is straight and A,B lies on L and for every M (if M is line such that M is straight and M contains A and M contains B then M is L)))

$\Downarrow$ (1.4.9(2),1.4.9(3),1.4.9(4))

for every A,B (if A is points and B is points and A,B is nonequal then for some L (L is line and L is straight and A,B lies on L and for every M (if M is line and M is straight and M contains A and M contains B then M is equal to L)))

$\Downarrow$ (1.4.10(8))

for every A,B (if A is points and B is points and A is nonequal to B then for some L (L is line and L is straight and A,B lies on L and for every M (if M is line and M is straight and M contains A and M contains B then M is equal to L)))

$\Downarrow$ (1.4.11(1))

for every A,B (if A is points and B is points and A is nonequal to B then for some L (L is line and L is straight and A lies on L and B lies on L and for every M (if M is line and M is straight and M contains A and M contains B then M is equal to L)))

$\Downarrow$ (1.4.12, given [x contains/contain y @ y lies on x])

for every A,B (if A is points and B is points and A is nonequal to B then for some L (L is line and L is straight and A lies on L and B lies on L and for every M (if M is line and M is straight and A lies on M and B lies on M then M is equal to L)))

$\Downarrow$

$$\begin{aligned}&\forall\, A,B\ ((A\,\varepsilon\,\text{Point}\wedge B\,\varepsilon\,\text{Point}\wedge A\neq B)\supset\\&\qquad\supset\exists\, L\ (L\,\varepsilon\,\text{Line}\wedge\text{isStraight}(L)\wedge\text{LiesOn}(A,L)\wedge\text{LiesOn}(B,L)\wedge\\&\qquad\wedge\forall\, M\ ((M\,\varepsilon\,\text{Line}\wedge\text{isStraight}(M)\wedge\text{LiesOn}(A,M)\wedge\text{LiesOn}(B,M))\supset M=L)))\end{aligned}$$

Figure 1.1: Full Translation Chain

1.5 Sections of ForTheL

We begin with an example of a well-formed ForTheL text (line numbers do not belong to the text). Throughout this section we will refer to it as to the Empty Set Text.

```
 1:    [set/sets] [element/elements]
 2:    [belongs/belong] [subset/subsets]

 3:    Signature SetSort.
 4:      A set is a notion.

 5:    Signature ElmSort.
 6:      Let S be a set.
 7:      An element of S is a notion.

 8:    Let x belongs to y stand for (x is an element of y).
 9:    Let x is in y stand for (x belongs to y).

10:    Definition DefSubset.
11:      Let S be a set.
12:      A subset of S is a set T such that all elements of T are in S.

13:    Definition DefEmpty.
14:      Let S be a set.
15:      S is empty iff S has no elements.

16:    Axiom ExEmpty.
17:      There exists an empty set.

18:    Proposition.
19:      Let S be a set.
20:      S is a subset of every set iff S is empty.
21:    Proof.
22:      If S is empty then S is a subset of every set.
23:      Indeed if S is empty then any element of S belongs to any set T.
24:      Assume that S is a subset of every set.
25:      Let us show that S is empty.
26:        Let z belong to S.
27:        Take an empty set E.
28:        z is an element of E (by DefSubset).
29:        We have a contradiction (by DefEmpty).
30:      end.
31:    qed.
```

1.5.1 Top-level sections

A ForTheL text is a sequence of parser instructions (token groups and synonym declarations) and *top-level sections*. Top-level sections are *axioms*, *definitions*, *signature extensions* and *propositions*. There are six top-level sections in the Empty Set Text, namely, a signature extension for the notion of set (lines 3–4), a signature extension for the notion of set element (lines 5–7), a definition of the subset relation (lines 10–12), a definition of the emptiness property (lines 13–15), an axiom (lines 16–17), and a proposition with a proof (lines 18–31).

Every top-level section has a *header* that describes the type of the section and may contain a *label* to use in references (as in the lines 28 and 29). The header of a top-level section is followed by the section's contents, the *body*.

$$
\begin{aligned}
\mathit{text} &\rightarrow \{\ \mathit{toplevel} \mid \mathit{tokenGroup} \mid \mathit{synonym}\ \} \\
\mathit{toplevel} &\rightarrow \mathit{axiom} \mid \mathit{definition} \mid \mathit{signature} \mid \mathit{proposition} \\
\mathit{axiom} &\rightarrow \mathit{axmHeader}\ \{\mathit{assume}\}\ \mathit{axmAffirm} \\
\mathit{axmHeader} &\rightarrow \texttt{Axiom}\ [\mathit{label}]\ \texttt{.} \\
\mathit{definition} &\rightarrow \mathit{defHeader}\ \{\mathit{assume}\}\ \mathit{defAffirm} \\
\mathit{defHeader} &\rightarrow \texttt{Definition}\ [\mathit{label}]\ \texttt{.} \\
\mathit{signature} &\rightarrow \mathit{sigHeader}\ \{\mathit{assume}\}\ \mathit{sigAffirm} \\
\mathit{sigHeader} &\rightarrow \texttt{Signature}\ [\mathit{label}]\ \texttt{.} \\
\mathit{proposition} &\rightarrow \mathit{prpHeader}\ \{\mathit{assume}\}\ \mathit{affirm} \\
\mathit{prpHeader} &\rightarrow (\,\texttt{Proposition} \mid \texttt{Theorem} \mid \texttt{Lemma} \mid \texttt{Corollary}\,)\ [\mathit{label}]\ \texttt{.} \\
\mathit{label} &\rightarrow \mathit{word}
\end{aligned}
$$

1.5.2 Sentences

Inside the body of any compound section we meet *sentences*: *assumptions* (lines 6, 11, 14, 16, 19, 24, 26 in the Empty Set Text), *choices* (line 27), *affirmations* of various kinds (lines 4, 7, 12, 15, 17, 20–31, 22–23, 23, 25–30, 28, 29), and *case hypotheses* (not included in the example). As we see in the grammar rules above, the body of any top-level section consists of zero or more assumptions followed by an affirmation of an appropriate kind.

Assumptions, affirmations, and case hypotheses are built on top of statement units. In what follows, we say that an affirmation (assumption) *affirms* (respectively, *assumes*) the corresponding statement. Choices are composed of notion units and state the nonemptiness of correspondent classes.

$$
\begin{aligned}
\mathit{assume} &\rightarrow \mathit{asmPrefix}\ \mathit{statement}\ \texttt{.} \\
\mathit{asmPrefix} &\rightarrow \texttt{let} \mid [\,\texttt{let us} \mid \texttt{we can}\,]\ (\,\texttt{assume} \mid \texttt{suppose}\,)\ [\,\texttt{that}\,] \\
\mathit{case} &\rightarrow \texttt{case}\ \mathit{statement}\ \texttt{.}\ \mathit{proof}\ \mathit{qed} \\
\mathit{axmAffirm} &\rightarrow \mathit{statement}\ \texttt{.} \\
\mathit{defAffirm} &\rightarrow \mathit{defStatement}\ \texttt{.} \\
\mathit{sigAffirm} &\rightarrow \mathit{sigStatement}\ \texttt{.} \\
\mathit{affirm} &\rightarrow \mathit{affPrefix}\ \mathit{statement}\ [\mathit{ref}]\ \texttt{.}\ [\mathit{prfHeader}\ \mathit{proof}\ \mathit{qed}] \\
&\mid\ \mathit{affPrefix}\ \mathit{statement}\ [\mathit{ref}]\ \texttt{.}\ \texttt{indeed}\ \mathit{shortProof} \\
&\mid\ \mathit{prfPrefix}\ \mathit{statement}\ [\mathit{ref}]\ \texttt{.}\ \mathit{proof}\ \mathit{qed} \\
\mathit{affPrefix} &\rightarrow [\,\texttt{then} \mid \texttt{therefore} \mid \texttt{hence}\,] \\
\mathit{prfPrefix} &\rightarrow [\,\texttt{let us} \mid \texttt{we can}\,]\ (\,\texttt{prove} \mid \texttt{show}\,)\ [\,\texttt{by}\ \mathit{method}\,]\ [\,\texttt{that}\,]
\end{aligned}
$$

$$\begin{array}{rcl}
\mathit{choose} & \to & \mathit{chsPrefix}\ \mathit{notions}\ [\,\mathit{ref}\,]\ \texttt{.}\ [\,\mathit{prfHeader}\ \mathit{proof}\ \mathit{qed}\,] \\
 & | & \mathit{chsPrefix}\ \mathit{notions}\ [\,\mathit{ref}\,]\ \texttt{.}\ \texttt{indeed}\ \mathit{shortProof} \\
\mathit{chsPrefix} & \to & [\,\texttt{then} \mid \texttt{therefore} \mid \texttt{hence}\,]\ [\,\texttt{let us} \mid \texttt{we can}\,]\ (\,\texttt{take} \mid \texttt{choose}\,) \\
\mathit{prfHeader} & \to & \texttt{proof}\ [\,\texttt{by}\ \mathit{method}\,]\ \texttt{.} \mid \texttt{indeed} \\
\mathit{method} & \to & \texttt{contradiction} \mid \texttt{case analysis} \mid \texttt{induction}\ [\,\texttt{on}\ \mathit{plainTerm}\,] \\
\mathit{ref} & \to & \texttt{(}\ \texttt{by}\ \mathit{label}\ \{\ \texttt{,}\ \mathit{label}\,\}\ \texttt{)} \\
\mathit{qed} & \to & \texttt{end.} \mid \texttt{qed.} \mid \texttt{obvious.} \mid \texttt{trivial.}
\end{array}$$

Note that proofs make a structural part of an affirmation or choice section. That is why the affirmations in the lines 20, 22 and 25 span several lines. Similarly, the examination of a proof case is a structural part of a case hypothesis. That is why we will often call the whole section a *case section*, identifying it with the case hypothesis sentence.

1.5.3 Proofs

In a logically correct text, any affirmation, choice or proof case (except the affirmations inside axioms, definitions, and signature extensions) should be grounded: is has to follow from its logical predecessors in the text. The author can make this implication more evident either with a reference to the relevant top-level sections or by supplying a *proof* which will make an additional implicit logical predecessor. In the Empty Set Text, there are three proofs: one for the affirmation in the line 20 (lines 21–31), one for the affirmation in the line 22 (line 23), and one for the affirmation in the line 25 (lines 26–30).

Proofs are not sections by themselves. A proof is a list of sentences (possibly with proofs of their own) and is considered as the *body* of the corresponding affirmation, choice, or case hypothesis. Short proofs (line 23) are single-element lists, containing just an affirmation. The main proof in the Empty Set Text is composed of an affirmation, an assumption, and another affirmation.

$$\begin{array}{rcl}
\mathit{proof} & \to & [\,\{\,\mathit{prfBody}\,\}\ \mathit{prfLast}\,] \\
\mathit{prfBody} & \to & \mathit{affirm} \mid \mathit{choose} \mid \mathit{assume} \\
\mathit{prfLast} & \to & \mathit{affirm} \mid \mathit{choose} \mid \mathit{case}\,\{\,\mathit{case}\,\} \\
\mathit{shortProof} & \to & \mathit{affirm}
\end{array}$$

One may explicitly mention the proof method that will be applied: currently, proofs by contradiction, by case analysis, and by induction are supported. The mentions “by contradiction” and “by case analysis” are optional, since the parser is able to guess that one of these methods is applied by analyzing the form of the proof. Conversely, the mention “by induction” is necessary as it instructs the verifier to formulate the *induction hypothesis* and to insert it into the proof in the appropriate place. This will be explained informally in Section 1.6.2 and then in detail in Section 2.3.2.

1.5.4 Precedence and declaration

Occurrences of sections in a ForTheL text relate to each other through *subordination* and *precedence*. In the following definitions, we will say “section” instead of “occurrence of a section in the text under consideration”.

We set the boundaries of sections in a text according to the grammar rules given above. For example, any top-level section starts with the first character of its header and ends where the section's last affirmation ends. Also, a sentence supplied with a proof ends where the proof ends, and not with the sentence-ending dot.

We say that a section $\mathbb{A}$ is a *subsection* of a section $\mathbb{B}$ if $\mathbb{A}$ lies within the boundaries of $\mathbb{B}$ and $\mathbb{B}$ is not $\mathbb{A}$. Thus, an affirmation supplied with a proof is a supersection of every section occurring in the proof. We call $\mathbb{A}$ an *immediate subsection* of $\mathbb{B}$ if $\mathbb{B}$ is the smallest supersection of $\mathbb{A}$, that is, $\mathbb{A}$ is an element of the body of $\mathbb{B}$.

A section $\mathbb{A}$ *textually precedes* a section $\mathbb{B}$ if $\mathbb{B}$ starts after the end of $\mathbb{A}$ in the text. A section $\mathbb{A}$ *logically precedes* a section $\mathbb{B}$ if $\mathbb{A}$ textually precedes $\mathbb{B}$ and any supersection of $\mathbb{A}$ is also a supersection of $\mathbb{B}$. By default, "predecessor" means "logical predecessor".

In the Empty Set Text, the assumption in the line 23 has the following logical predecessors: the both signature extensions, the both definitions, the axiom `ExEmpty`, the assumption in the line 19 and the affirmation in the line 22. The affirmations in the lines 20 and 23 are not predecessors of the assumption in question (the former is a supersection of it and the latter is a subsection of a logical predecessor).

We use logical precedence to determine the set of variables declared by an assumption or choice in a ForTheL text (no other sections can declare variables).

An assumption is said to *describe* a variable v whenever the assumed statement describes v (Section 1.3.7). A choice is said to *describe* a variable v whenever v is a name of any notion O in the chain, i.e. $v \in \mathbf{n}(O)$.

An assupmtion or choice $\mathbb{A}$ *declares* a variable v whenever $\mathbb{A}$ describes v but no logical predecessor of $\mathbb{A}$ does. If a variable is declared by a logical predecessor of $\mathbb{A}$, it is said to be *known* in $\mathbb{A}$. So, a known variable cannot be declared.

In a well-formed ForTheL text, any variable that occurs freely in a sentence (see the next section for the definition of free occurrence) must be either known or declared in that sentence. Also, ForTheL prohibits re-descriptions in choices. That is, every variable described in a choice must not be known in it.

1.5.5 Formula image of sections

Similarly to the ForTheL units, any section $\mathbb{A}$ in a normalized ForTheL text can be translated to a first-order formula $|\mathbb{A}|$, called the *formula image of* $\mathbb{A}$.

- Assumptions: $|\, asmPrefix\ (statement)_S\ \texttt{.}\, | = |S|$
- Affirmations:

$$\begin{aligned}
&|\,(statement)_S\ \texttt{.}\,| = |\,(defStatement)_S\ \texttt{.}\,| = |\,(sigStatement)_S\ \texttt{.}\,| = |S|\\
&|\,affPrefix\ (statement)_S\ [\,ref\,]\ \texttt{.}\ [\,prfHeader\ proof\ qed\,]\,| = |S|\\
&|\,affPrefix\ (statement)_S\ [\,ref\,]\ \texttt{.}\ \texttt{indeed}\ shortProof\,| = |S|\\
&|\,prfPrefix\ (statement)_S\ [\,ref\,]\ \texttt{.}\ proof\ qed\,| = |S|
\end{aligned}$$

- Choices:

$$\begin{aligned}
&|\,chsPrefix\ (notions)_N\ [\,ref\,]\ \texttt{.}\ [\,prfHeader\ proof\ qed\,]\,| = \overline{|\,\texttt{there exist}\ N\,|}\\
&|\,chsPrefix\ (notions)_N\ [\,ref\,]\ \texttt{.}\ \texttt{indeed}\ shortProof\,| = \overline{|\,\texttt{there exist}\ N\,|}
\end{aligned}$$

  where, for every variable v described by the choice (i.e. for every name of every notion in N), the existential quantifier on v is erased from the overlined formula, thus making v free. Consider the following examples:

$$\begin{aligned}
&|\ \texttt{Take a set S and an element of S.}\ | = (\text{aSet}(S) \wedge \exists x\, (\text{aElement}(x, S)))\\
&|\ \texttt{Take a set S and an element x of S.}\ | = (\text{aSet}(S) \wedge \text{aElement}(x, S))\\
&|\ \texttt{Take a set and an element x of S.}\ | = \exists z\, (\text{aSet}(z) \wedge \text{aElement}(x, S))
\end{aligned}$$

- Cases: $|\,\texttt{case}\ (statement)_S\ \texttt{.}\ proof\ qed\,| = |S| \supset \texttt{thesis}$

- Top-level sections:

$$\begin{aligned}
&|\, axmHeader\ \ (\{\, assume \,\}\ \ axmAffirm)_\Delta \,| = |\Delta| \\
&|\, defHeader\ \ (\{\, assume \,\}\ \ defAffirm)_\Delta \,| = |\Delta| \\
&|\, sigHeader\ \ (\{\, assume \,\}\ \ sigAffirm)_\Delta \,| = |\Delta| \\
&|\, prpHeader\ \ (\{\, assume \,\}\ \ affirm)_\Delta \,| = |\Delta|
\end{aligned}$$

where $|\Delta|$, the formula image of a sequence Δ of sentences, is calculated according to the following rules:

$$\begin{array}{lll}
|\ \mathbb{A}\ \ \Delta\ | = \forall\vec{x}\,(|\mathbb{A}| \supset |\Delta|) & \text{where} & \mathbb{A} \text{ is an assumption} \\
 & & \vec{x} \text{ is the set of variables declared in } \mathbb{A} \\
|\ \mathbb{A}\ \ \Delta\ | = \exists\vec{x}\,(|\mathbb{A}| \wedge |\Delta|) & \text{where} & \mathbb{A} \text{ is an affirmation, choice, or case} \\
 & & \vec{x} \text{ is the set of variables declared in } \mathbb{A} \\
 & & (\vec{x} = \varnothing, \text{ if } \mathbb{A} \text{ is not a choice}) \\
|\ \epsilon\ | = \top & & \text{the image of the empty sequence is truth}
\end{array}$$

Note that a proof supplied for an affirmation, choice, or proof case does not influence its formula image. The formula image expresses the indented meaning of a section, and it is left to a verifier to check that the written proof supports that indented meaning.

We define the set of free variables of a section $\mathbb{A}$ as the set of free variables of the formula image of $\mathbb{A}$: $\mathcal{FV}(\mathbb{A}) = \mathcal{FV}(|\mathbb{A}|)$. We denote by $\mathcal{DV}(\mathbb{A})$ the set of variables declared in $\mathbb{A}$. If $\mathbb{A}$ is not an assumption or choice, then $\mathcal{DV}(\mathbb{A})$ is empty. Given a sequence of sections Δ, the sets $\mathcal{FV}(\Delta)$ and $\mathcal{DV}(\Delta)$ are defined as unions of the corresponding sets for individual sections.

We stress again that one must consider a ForTheL sentence in view of its logical predecessors in order to determine the set of declared variables. As a result, the formula image of a sequence of sentences depends on its logical predecessors, too. To make this explicit, we introduce the following context-dependent notation.

Let Γ be a sequence of ForTheL sections and F, a formula. The set $\mathcal{DV}_\Gamma(F)$ is defined to be the difference $\mathcal{FV}(F)\backslash\mathcal{FV}(\Gamma)$. Accordingly, $\mathcal{DV}_\Gamma(\mathbb{A}) = \mathcal{FV}(|\mathbb{A}|)\backslash\mathcal{FV}(\Gamma)$.

Now, the context-dependent image of a sequence Δ, denoted $|\Delta|_\Gamma$, is defined as follows:

$$\begin{array}{ll}
|\ \mathbb{A}\ \ \Delta\ |_\Gamma = \forall\vec{x}\,(|\mathbb{A}| \supset |\Delta|_{\Gamma,\mathbb{A}}) & \mathbb{A} \text{ is an assumption and } \vec{x} = \mathcal{DV}_\Gamma(\mathbb{A}) \\
|\ \mathbb{A}\ \ \Delta\ |_\Gamma = \exists\vec{x}\,(|\mathbb{A}| \wedge |\Delta|_{\Gamma,\mathbb{A}}) & \mathbb{A} \text{ is an affirmation, choice, or case, and } \vec{x} = \mathcal{DV}_\Gamma(\mathbb{A}) \\
|\ \epsilon\ |_\Gamma = \top & \text{the image of the empty sequence is truth}
\end{array}$$

In a well-formed ForTheL text, by definitions given in Section 1.5.4, the set $\mathcal{DV}(\mathbb{A})$ is precisely the set $\mathcal{DV}_\Gamma(\mathbb{A})$ for Γ, the set of logical predecessors of $\mathbb{A}$. Therefore, $|\Delta|$ is the same as $|\Delta|_\Gamma$, too.

1.6 Formalization example

The formal proof of the fact that $\sqrt{2}$ is not a rational number is a classical touchstone problem formalized in many mathematical assistants [13]. We have chosen its generalized version to demonstrate step by step the process of formalization in ForTheL. We show how the initial problem is transformed to a self-contained ForTheL text, and how to refine the text in order to fill the gaps in a proof that the verification system can not jump over by itself.

1.6.1 Comprehending the problem

The existence of numbers which cannot be represented as a fraction of two integers was proved by mathematicians of the Pythagorean School who demonstrated that the diagonal of a square with side 1 is not such a fraction. Here is the theorem in its traditional form:

THEOREM. $\sqrt{2}$ is not rational.

Proof. Assume that $\sqrt{2}$ is rational. Then there exist relatively prime natural numbers n,m such that $\sqrt{2} = n/m$, and therefore, $2m^2 = n^2$. Since n^2 is even, n also is; say $n = 2k$. Then $2m^2 = 4k^2$, that is $m^2 = 2k^2$. Thus, m^2 is even too, whence m is even. So, n and m have a common nontrivial divisor in contradiction to our hypothesis. □

The claim can be easily generalized as follows:

THEOREM. The square root of any prime number is not rational.

It is not surprising that the proof of this more general fact is in a manner more logical though less straightforward (an auxiliary lemma is required):

Proof. Let p be a prime such that $\sqrt{p}$ is rational. Then there exist relatively prime natural numbers n,m such that $\sqrt{p} = n/m$, and therefore, $pm^2 = n^2$. Since p is prime, p divides n; say $n = kp$. Then $pm^2 = p^2k^2$, that is $m^2 = pk^2$. Thus, p divides m^2 and m. So, n and m have a common nontrivial divisor and we have a contradiction. □

The fact that $2|n^2$ implies $2|n$ can be proved directly: $(2k + 1)^2 = 2(2k^2 + 2k) + 1$. The generalized statement $(p|n^2 \supset p|n)$ is not that obvious. So, in order to make our proof verifiable, we involve the following lemma (PDP stands for "Prime that Divides a Product"):

LEMMA PDP. Let m,n be natural numbers and p be a prime. If p divides mn then p divides m or n.

Proof. We proceed by induction on $(m + n + p)$. If $m \geqslant p$ then $p|mn$ implies $p|(m - p)n$ and we apply the induction hypothesis ($p|m - p$ obviously implies $p|m$). In the same manner we treat the case $n \geqslant p$.

Say $m, n < p$ and $pk = mn$. If k is zero or one, the lemma's statement is immediately proved. Otherwise, let r be a prime divisor of k. Then r divides mn and, therefore $r|m$ or $r|n$, using the induction hypothesis and the fact that $r \leqslant k < p$.

If r divides m then $p(k/r) = (m/r)n$ and the induction hypothesis is applicable. Obviously, $p|(m/r)$ implies $p|m$. We treat the case $r|n$ in the same way. □

What we have, is not yet a self-contained text. We must complete it with an ensemble of preliminary facts, including, in the order of introduction: basic properties of natural numbers (addition, subtraction, and ordering), some Euclidean arithmetic (divisibility, quotient, primes and relative primes), definitions of a rational number and square root. These *preliminaries* are well-known properties of well-known mathematical objects and we can take them for granted.

1.6.2 ForTheL'ization

The mathematical notation used in our example is quite simple. Thus it suffices just to reduce the syntax to ASCII character set and to get rid of some syntactic sugar in order to write down this text in ForTheL. For the sake of brevity, we do not cite the preliminaries (see Section 2.4 for some typical examples).

```
Lemma PDP.  For all natural numbers n,m,p
  if p is prime and p | n * m then p | n or p | m.
Proof by induction on ((n + m) + p).
  Let n,m,p be natural numbers.
  Assume that p is prime and p divides n * m.
  Case p <= n.
    p divides (n - p) * m and n - p < n.
    Then p divides n - p or p divides m.
    If p divides n - p then p divides n.
    end.
    Case p <= m.
      p divides n * (m - p) and m - p < m.
      Then p divides n or p divides m - p.
      If p divides m - p then p divides m.
    end.
    Case n < p and m < p.
      Take a natural number k such that n * m = p * k.
      Case k = 0 \/ k = 1. Obvious.
      Case k != 0 /\ k != 1.
        Take a prime divisor r of k.
        r divides n * m and r <= k and k < p.
        Then r divides n or r divides m.
        Case r divides n.
          p divides (n / r) * m and (n / r) < n.
          Then p divides (n / r) or p divides m.
          If p divides (n / r) then p divides n.
        end.
        Case r divides m.
          p divides n * (m / r) and (m / r) < m.
          Then p divides n or p divides (m / r).
          If p divides (m / r) then p divides m.
        end.
      end.
    end.
  qed.

  Theorem Main.  Let p be a prime natural number.
    For no rational number q the square of q is p.
  Proof by contradiction.
    Let q be a rational number such that q * q = p.
    Take relatively prime natural numbers n,m such that q * m = n.
    Then p * (m * m) = (n * n).
    Hence p divides n * n and p divides n.
    Choose a natural number k such that n = p * k.
    Then we have p * (m * m) = p * (k * n).
    The square of m is equal to (p * k) * k.
    Hence p divides m * m and p divides m.
    We have a contradiction.
  qed.
```

The text above (in the rest of this section, we will refer to it as to the Square Root Text) is a syntactically valid ForTheL text. Moreover, being supplied with suitable preliminary facts, this text is probably correct, meaning that every statement in it can be deduced from its logical predecessors. (The formal definition of a correct text will be given in the next chapter.) We say "probably", because this ForTheL fragment just reflects the proofs written above in natural language and we believe those proofs to be correct; however, we cannot be completely sure in any one of these two claims, so the formal verification is required.

Let us explain how a text in natural language is converted to ForTheL. First of all, we add explicit structural markup. Proofs, subproofs, cases should have explicit headers and endings. Note that SAD does not support proof by analogy, so we have to write twice almost the same reasoning in the proof of Lemma PDP (compare the cases "`p <= n`" and "`p <= m`"; "`r divides n`" and "`r divides m`").

Since our motivation is not to show the existence of irrational numbers, but just to prove a property of primes, we can slightly simplify the reasoning by eliminating some operations which we use sporadically, such as square root or division on rational numbers.

We remove some reminders for a human reader — such as "Since p is prime" — since they do not help a reasoner to find a proof (indeed, the system will not forget that p is prime), and therefore are not included in the syntax of ForTheL (they may be easily added on the user's request). On the other hand, we may have to add some supplementary statements which would seem superfluous to a reader but make the automated verification possible (see Section 1.6.3).

Finally, since ForTheL formalizes just a small fragment of English (even "mathematical English"), there are some natural expressions that should be reformulated. For example, we unfold English statements where a verb applies to a list of objects: in ForTheL, we have to say "p divides m^2 and p divides m" instead of "p divides m^2 and m".

In conclusion, let us give some informal explanations on using induction (see Section 2.3.2 for precise definitions). Proofs by induction do not have any particular structure in ForTheL; for example, there is no special markup for the base case and the step case. To write a proof by induction, the author must explicitly mention the proof scheme: "`proof by induction` [`on` *term*]". The statement to prove must be a universally quantified formula of the form $\forall \vec{v} F$ and the free variables of the induction term (say, t) must be in $\vec{v}$. If the induction term is omitted then the variable under the outermost universal quantifier is taken as t. These variables must be declared at the beginning of the proof section in the same way as they are described in the goal. If these requirements are satisfied then the verifier formulates and inserts in the proof the appropriate induction hypothesis:

$$\forall \vec{u}\, (t[\vec{u}/\vec{v}] \prec t \;\supset\; F[\vec{u}/\vec{v}])$$

Here, the binary relation $\prec$ stands for an abstract well-founded ordering, as explained below. Considering the free variables of t as arbitrary but fixed constants introduced at some point above in the proof, and calling t the *current value* of the induction term, the induction hypothesis can be formulated as follows: *the claim holds for every value of the induction term that is less than the current value with respect to a fixed well-founded ordering*.

If we succeed to prove the claim for the current value of the induction term in presence of the induction hypothesis, we can conclude that the claim holds for all values of that term and the initial goal is thus proved. In other words, the induction proof scheme in ForTheL follows the *general induction principle*.

Since there is no finite axiomatization of well-foundness in first-order logic, the verifier can by no means check whether a given user-defined ordering is well-founded. Instead, ForTheL provides a special binary predicate symbol `-<-` which is used as an abstract a priori well-founded relation in induction arguments. It is left to the author to write down the axioms defining particular properties of the relation `-<-`, and the system will take for granted that the supplied axioms are compatible with well-foundness.

In the current example, the corresponding axiom is:

```
Axiom IH.   For all natural numbers n,m (n < m => n -<- m).
```

and the induction hypothesis for the proof of PDP is as follows:

```
For all natural numbers N,M,P if ((N + M) + P) -<- ((n + m) + p)
  then if P is prime and P | N * M then P | N or P | M.
```

1.6.3 Filling the gaps

Despite our confidence in the correctness of the Square Root Text, SAD fails to verify it. The reasoning capabilities of the system are too weak to establish automatically the truth of the very first affirmation in the proof of Lemma PDP: "`p divides (n - p) * m`", the fourth line of the proof. In fact, almost all the affirmations in the Square Root Text will not be proved as they are.

Generally, there are several ways to make an affirmation more "evident" to the system. The simplest one is to provide it with *references*, a list of top-level premises to use in verification. The rest of top-level facts (axioms, definitions, lemmas) will be excluded from the proof search. When your problem has a large set of preliminaries, such an explicit filtering of premises can be very helpful, reducing the search time from half an hour to half a second. Note that a top-level fact mentioned in references is not required to actually contribute to the found proof.

However, the exact list of needed premises can be sometimes hard to find (in fact, the author has to foresee the proof) and can turn out to be quite long, making the text unnatural. Finally, it may happen that even if the set of premises is filtered, the prover still fails to find the proof. The needed proof step may be just too large.

Another way is to insert an intermediate statement which is easier to verify than the affirmation in question (say, A) and which may shorten the proof of A. The author may provide a proof section for A and put there not just a single statement but the whole intermediate reasoning. The rule of thumb is as follows: if your intermediate argument $\mathcal{P}$ is going to be used merely to support verification of A then place $\mathcal{P}$ in a subproof; if it can help to verify other claims in the current proof section, too, then put $\mathcal{P}$ above A, so that $\mathcal{P}$ will be a premise for the rest of the section; if $\mathcal{P}$ reappears several times in your proof under different instantiations, then make it a top-level lemma.

The needed level of detail is usually found by trial and error, until the problematic affirmation becomes checkable. Of course, this level strongly depends on the reasoner and the prover being used. Therefore, some knowledge about the deductive machinery of the verifier can help in preparation of a verifiable text. In hard cases, the author may have to revise the whole proof and the set of preliminaries.

Consider the first case section in the proof of Lemma PDP in its final, verifiable state:

```
Case p <= n.
  Let us show that p divides (n - p) * m and n - p < n.
    n = p + (n - p) and n * m = p * ((n * m) / p).
    n * m = (p * m) + ((n - p) * m) (by AMDistr).
    p divides (n - p) * m (by DefDiv, DivMin).
  end.
  Then p divides n - p or p divides m (by IH).
  Indeed ((n - p) + m) + p < (n + m) + p (by MonAdd).
  If p divides n - p then p divides n (by DefDiv,DivSum).
end.
```

Let us compare the case section above with the corresponding one in the Square Root Text. It consists of the same three affirmations which are now provided with subproofs and references. Note that purely algebraic tasks turn to be difficult for a generic first-order prover (especially in presence of the axioms of associativity, commutativity, and distributivity), so that we give three intermediate statements in order to deduce $p|(n-p)m$ from $p|nm$.

Another subgoal, $n - p < n$, does not require any support from us. On the other hand, the system fails to deduce $((n-p)+m)+p < (n+m)+p$ from $n-p < n$ on its own, and we have to mention it explicitly in a subproof in order to apply the induction hypothesis.

Other "hard" affirmations in the text are processed in a similar way. At the end, the proof of Lemma PDP becomes twice longer than the initial one. At the same time, the proof of the main theorem does not grow, since we can make it verifiable just by referencing the necessary preliminary facts:

```
Theorem Main.  Let p be a prime natural number.
  For no rational number q the square of q is p.
Proof by contradiction.
  Let q be a rational number such that q * q = p.
  Take relatively prime natural numbers n,m such that q * m = n.
  Then p * (m * m) = (n * n) (by MulAsso,MulComm).
  Hence p divides n * n and p divides n (by DefDiv,PDP).
  Choose a natural number k such that n = p * k (by DefDiv).
  Then we have p * (m * m) = p * (k * n) (by MulAsso).
  The square of m is equal to (p * k) * k (by MulComm,MulCanc).
  Hence p divides m * m and p divides m (by MulAsso,DefDiv,PDP).
  We have a contradiction (by DefRelPr).
```

```
qed.
```

In its final state, the ForTheL text about square roots of prime numbers consists of 205 non-blank lines: 126 lines of preliminaries and 79 lines for Lemma PDP and the main theorem. The text contains 95 goals (lemmas in preliminaries taken into account, too) and is fully verified by SAD using SPASS [11] as the background prover in 45 seconds (0.5 sec/goal on average, 4.7 seconds for the longest session). Statistics on using different provers with different texts will be given in Appendix **??**.

Chapter 2

Correctness of a ForTheL text

2.1 Introduction

The mathematical text (which we aspire to approximate with the ForTheL text) is a complex object containing axioms, definitions, theorems, and proofs of various kinds. What “correctness” of such an object might stand for?

The formal semantics of a text can be given by packing the whole text in a single statement and considering the corresponding logical formula, the formula image of the text. Then the text could be declared correct whenever its formula image is deducible in the underlying logic. This approach is simple, theoretically transparent but absolutely impracticable. The precise notion of correctness obtained in this way can hardly be considered as a formal specification of a verifier. That’s why we develop a specific notion of text correctness that, though being less straightforward, can be formalized with the help of a logical calculus on one hand and can serve as a specification on the other.

In our approach, we distinguish ontological and logical correctness of a formal text.

A *logically correct* text is a text, one might say, «truthful». Everything is grounded: every affirmation made in the text can be deduced from its logical predecessors.

An *ontologically correct* text is a text, one might say, «sensible». Nothing comes from nowhere and everything is used properly: every non-logical symbol occurring in the text is introduced somewhere above that occurrence (in ForTheL, with a definition or signature extension), and is employed within the domain established by the introduction (in ForTheL, satisfies the assumptions of that definition or signature extension). In other words, all terms and formulas are well-defined in an ontologically correct text.

Ontological correctness is closely related to type correctness in typed languages (especially, in weakly typed systems like WTT [4]). It allows to spot formalization errors which otherwise could hardly be detected. Indeed, an accidental ontological incorrectness most often implies logical incorrectness (i.e. presence of false or unprovable claims). And it is much harder to trace a failure log of a prover back to an invalid occurrence than to discover it in the first place.

Moreover, during ontological verification, we obtain information about applicability of definitions, signature extensions and other preliminary facts at individual positions in the text in question. As long as subsequent transformations (e.g. during the logical verification phase) preserve ontological correctness and other local properties (and that should always be the case) we can unfold definitions and apply lemmas without further checking.

The proposed approach can be regarded as a way to handle partial relations and functions in a mathematical text. Instead of introducing special individual or truth values for undefinedness (as in Kleene’s strong logic [5]), ontological correctness requires every term or atom to be well-defined *a priori*. Using deductive techniques which preserve local properties, we can guarantee that the text under consideration always stays well-defined. In our opinion, this corresponds well to the usual mathematical practice.

There is one technical problem, however. How can we formally define ontological correctness, given that it refers to properties of particular occurrences of atomic formulas and terms inside complex propositions in the text?

Consider a formula of the form $(\cdots\forall x\,(x \in \mathbb{R}^+ \supset (\cdots \frac{xy}{x} \cdots))\cdots)$. It is obvious that the fraction is well-defined and we can reduce it to y, but how can we justify that? The task itself seems absurd: as soon as a term depends on bound variables, we can not reason about it. In the traditional fashion, we should first split our formula up to the quantifier that binds x, make a substitution (or skolemization), separate $x \in \mathbb{R}^+$, and only then we can reduce the fraction. Yet, we would like to judge the ontological correctness of a formula before we touch it at all.

While the statement "x is non-zero" is surely meaningless, we can say that "x is non-zero in this occurrence of $\frac{xy}{x}$". Our intuition suggests that along with the usual notion of validity, a certain *local validity* of a proposition can be defined with respect to some position in a formula. A statement that is generally false or just meaningless can become locally valid being considered in the corresponding context. In what follows, we call such a statement a *local property* of the position in question.

We begin by introducing and investigating the locally valid statements. Then we use our results to define ontological correctness and formula transformations which preserve it.

2.2 Mathematical grounds

2.2.1 Preliminary notions

We consider a one-sorted first-order language whose signature consists of three disjoint sets of symbols — those of functions, predicates, and notions. The logical symbols of the language are: equality ($\approx$), class membership (ε), the standard propositional connectives $\neg$, $\wedge$, $\vee$, $\supset$, $\equiv$ and the quantifier symbols $\forall$ and $\exists$.

Terms in this language are ordinary first-order terms. Atomic formulas are of five possible forms: $\top$, $\mathfrak{T}$, $s_1 \approx s_2$, $P(s_1,\ldots,s_n)$, and $s\,\varepsilon\,N(s_1,\ldots,s_n)$, where $\top$ stands for the boolean truth, $\mathfrak{T}$ denotes the propositional variable `thesis` (a placeholder for the statement being proved at a given position in a text), $s, s_1, \ldots, s_n$ are terms, P is a predicate symbol, and N is a notion symbol. Recall that notions denote (parametrized) classes of objects, so that the atom $s\,\varepsilon\,N(s_1,\ldots,s_n)$ means that the term s belongs to the particular class $N(s_1,\ldots,s_n)$.

The respective order of subformulas is significant for some of our definitions, therefore we consider $F \wedge G$ and $G \wedge F$ as different formulas (the same is true for $\vee$, $\equiv$, and $\approx$).

The symbol $\sim$ is the abbreviation for the double implication: $F \sim G$ stands for $(F \supset G) \wedge (G \supset F)$. We write the negated equality $\neg(s_1 \approx s_2)$ as $s_1 \not\approx s_2$ and the negated truth $\neg\top$ as $\bot$.

The *complement* of a formula H, denoted $H^{\neg}$, is defined as follows:

$$\begin{array}{llll}(F \equiv G)^{\neg} = (F \sim G)^{\neg} & (F \wedge G)^{\neg} = \neg F \vee \neg G & (\forall x\, F)^{\neg} = \exists x\, \neg F & (\neg F)^{\neg} = F \\ (F \supset G)^{\neg} = F \wedge \neg G & (F \vee G)^{\neg} = \neg F \wedge \neg G & (\exists x\, F)^{\neg} = \forall x\, \neg F & A^{\neg} = \neg A\end{array}$$

We consider substitutions as functions which map variables to terms. For any substitution ϕ, if $x\phi$ is different from x, we call the term $x\phi$ a *substitute* of x in ϕ. A substitution is *finite* whenever the set of its substitutes is finite. We write finite substitutions as sequences of the form $[t_1/x_1, \ldots, t_n/x_n]$.

Position is a word in the alphabet $\{0, 1, \ldots\}$. In what follows, positions are denoted by Greek letters τ, μ and ν; the letter ϵ denotes the null position (the empty word). Positions point to particular subformulas and subterms in a formula or term.

The *set of positions* in an atomic formula or a term E, denoted $\Pi(E)$, is defined as follows (below $i.\Pi$ stands for $\{\, i.\tau \mid \tau \in \Pi \,\}$):

$$\begin{array}{rl}\Pi(s_0\,\varepsilon\,N(s_1,\ldots,s_n)) = \{\epsilon\} \cup \bigcup i.\Pi(s_i) & \Pi(s \approx t) = \{\epsilon\} \cup 0.\Pi(s) \cup 1.\Pi(t) \\ \Pi(P(s_0,\ldots,s_n)) = \{\epsilon\} \cup \bigcup i.\Pi(s_i) & \Pi(\top) = \{\epsilon\} \\ \Pi(f(s_0,\ldots,s_n)) = \{\epsilon\} \cup \bigcup i.\Pi(s_i) & \Pi(\mathfrak{T}) = \{\epsilon\}\end{array}$$

The *set of positions* in a formula H, denoted $\Pi(H)$, is the disjoint union

$$\Pi(F) = \Pi^+(F) \sqcup \Pi^-(F) \sqcup \Pi^\circ(F)$$

of the *set of positive positions* $\Pi^+(H)$, the *set of negative positions* $\Pi^-(H)$, and the *set of neutral positions* $\Pi^\circ(H)$ (in what follows, A stands for an atomic formula):

$$\begin{aligned}
\Pi^+(F \equiv G) &= \{\epsilon\} & \Pi^+(F \supset G) &= \{\epsilon\} \cup 0.\Pi^-(F) \cup 1.\Pi^+(G)\\
\Pi^+(F \wedge G) &= \{\epsilon\} \cup 0.\Pi^+(F) \cup 1.\Pi^+(G) & \Pi^+(F \vee G) &= \{\epsilon\} \cup 0.\Pi^+(F) \cup 1.\Pi^+(G)\\
\Pi^+(\forall x\, F) &= \{\epsilon\} \cup 0.\Pi^+(F) & \Pi^+(\exists x\, F) &= \{\epsilon\} \cup 0.\Pi^+(F)\\
\Pi^+(\neg F) &= \{\epsilon\} \cup 0.\Pi^-(F) & \Pi^+(A) &= \Pi(A)
\end{aligned}$$

$$\begin{aligned}
\Pi^-(F \equiv G) &= \varnothing & \Pi^-(F \supset G) &= 0.\Pi^+(F) \cup 1.\Pi^-(G)\\
\Pi^-(F \wedge G) &= 0.\Pi^-(F) \cup 1.\Pi^-(G) & \Pi^-(F \vee G) &= 0.\Pi^-(F) \cup 1.\Pi^-(G)\\
\Pi^-(\forall x\, F) &= 0.\Pi^-(F) & \Pi^-(\exists x\, F) &= 0.\Pi^-(F)\\
\Pi^-(\neg F) &= 0.\Pi^+(F) & \Pi^-(A) &= \varnothing
\end{aligned}$$

$$\begin{aligned}
\Pi^\circ(F \equiv G) &= 0.\Pi(F) \cup 1.\Pi(G) & \Pi^\circ(F \supset G) &= 0.\Pi^\circ(F) \cup 1.\Pi^\circ(G)\\
\Pi^\circ(F \wedge G) &= 0.\Pi^\circ(F) \cup 1.\Pi^\circ(G) & \Pi^\circ(F \vee G) &= 0.\Pi^\circ(F) \cup 1.\Pi^\circ(G)\\
\Pi^\circ(\forall x\, F) &= 0.\Pi^\circ(F) & \Pi^\circ(\exists x\, F) &= 0.\Pi^\circ(F)\\
\Pi^\circ(\neg F) &= 0.\Pi^\circ(F) & \Pi^\circ(A) &= \varnothing
\end{aligned}$$

For the sake of consistency, we set $\Pi^+(t) = \Pi(t)$ and $\Pi^-(t) = \Pi^\circ(t) = \varnothing$ for any term t.

Among positions, we distinguish those of formulas ($\Pi_{\mathbf{F}}$), those of atomic formulas ($\Pi_{\mathbf{A}}$), and those of terms ($\Pi_{\mathbf{t}}$). Obviously, $\Pi(F) = \Pi_{\mathbf{t}}(F) \cup \Pi_{\mathbf{F}}(F)$, $\Pi_{\mathbf{A}}(t) = \Pi_{\mathbf{F}}(t) = \varnothing$, $\Pi_{\mathbf{A}}(F) \subseteq \Pi_{\mathbf{F}}(F)$, $\Pi(t) = \Pi_{\mathbf{t}}(t)$. We split the sets $\Pi_{\mathbf{t}}$, $\Pi_{\mathbf{A}}$, and $\Pi_{\mathbf{F}}$ into positive, negative, and neutral parts, too.

Given a formula H and a position $\pi \in \Pi(H)$, the position $\widehat{\pi}$ is the maximal prefix of π in $\Pi_{\mathbf{F}}(H)$. In what follows, we will often use this conversion to extend notions and operations defined on positions from $\Pi_{\mathbf{F}}$ to the whole Π.

A formula or a term E coupled with a position $\tau \in \Pi(E)$ defines an *occurrence*.

Let us say that τ is a *subposition* (subordinate position) of π whenever $\tau = \pi.i.\mu$ for some natural i and a position μ. We will call τ an *immediate subposition* if μ is empty, that is, $\tau = \pi.i$. We call τ an *outer position* with respect to π whenever τ is not a subposition of π. Note that any position is outer with respect to itself.

Let us say that π is a *textual predecessor* of τ whenever $\pi = \omega.i.\mu$ and $\tau = \omega.j.\eta$ and $i < j$. If $\mu = \epsilon$, we will say that π is a *logical predecessor* of τ. By default, "predecessor" means "logical predecessor".

Given a formula or a term E and a position τ in $\Pi(E)$, we will denote by $E|_\tau$ the subformula or subterm occurring in that position. In what follows, $(*F)$ stands for $(\neg F)$, $(\forall x\, F)$, or $(\exists x\, F)$; and $(F * G)$ stands for $(F \equiv G)$, $(F \supset G)$, $(F \wedge G)$, or $(F \vee G)$:

$$\begin{aligned}
F|_\epsilon &= F & (*\,F)|_{0.\tau} &= F|_\tau\\
(s_0 \,\varepsilon\, N(s_1, \ldots, s_n))|_{i.\tau} &= s_i|_\tau & (F * G)|_{0.\tau} &= F|_\tau\\
P(s_0, \ldots, s_n)|_{i.\tau} &= s_i|_\tau & (F * G)|_{1.\tau} &= G|_\tau\\
f(s_0, \ldots, s_n)|_{i.\tau} &= s_i|_\tau & (s \approx t)|_{0.\tau} &= s|_\tau\\
t|_\epsilon &= t & (s \approx t)|_{1.\tau} &= t|_\tau
\end{aligned}$$

Given a formula or a term E, a position τ in $\Pi(E)$, and a formula or a term e, we will denote by $E[e]_\tau$ the result of replacement of $E|_\tau$ with e. Of course, e must be a term if $\tau \in \Pi_{\mathbf{t}}(E)$, and a formula otherwise:

$$\begin{aligned}
E[e]_\epsilon &= e & (*\,F)[e]_{0.\tau} &= *\,F[e]_\tau\\
(s_0 \,\varepsilon\, N(s_1, \ldots, s_n))[e]_{0.\tau} &= s_0[e]_\tau \,\varepsilon\, N(s_1, \ldots, s_n) & (F * G)[e]_{0.\tau} &= F[e]_\tau * G\\
(s_0 \,\varepsilon\, N(s_1, \ldots, s_n))[e]_{i.\tau} &= s_0 \,\varepsilon\, N(s_1, \ldots, s_i[p]_\tau, \ldots, s_n) & (F * G)[e]_{1.\tau} &= F * G[e]_\tau\\
P(s_0, \ldots, s_n)[e]_{i.\tau} &= P(s_0, \ldots, s_i[p]_\tau, \ldots, s_n) & (s \approx t)[e]_{0.\tau} &= s[e]_\tau \approx t\\
f(s_0, \ldots, s_n)[e]_{i.\tau} &= f(s_0, \ldots, s_i[p]_\tau, \ldots, s_n) & (s \approx t)[e]_{1.\tau} &= s \approx t[e]_\tau
\end{aligned}$$

Note that free variables of e may become bound in $F[e]_\tau$.

2.2.2 Local validity and local properties

Given a formula F, a position $\pi \in \Pi_{\mathbf{F}}(F)$, and a formula U, we define the *local image* of U w.r.t. F and π, denoted $\langle U \rangle^F_\pi$, as follows:

$$\begin{array}{lll}
\langle U \rangle^{F \equiv G}_{0.\pi} = \langle U \rangle^F_\pi & \langle U \rangle^{F \equiv G}_{1.\pi} = \langle U \rangle^G_\pi & \langle U \rangle^{\forall x F}_{0.\pi} = \forall x \, \langle U \rangle^F_\pi \\
\langle U \rangle^{F \supset G}_{0.\pi} = G \vee \langle U \rangle^F_\pi & \langle U \rangle^{F \supset G}_{1.\pi} = F \supset \langle U \rangle^G_\pi & \langle U \rangle^{\exists x F}_{0.\pi} = \forall x \, \langle U \rangle^F_\pi \\
\langle U \rangle^{F \wedge G}_{0.\pi} = G \supset \langle U \rangle^F_\pi & \langle U \rangle^{F \wedge G}_{1.\pi} = F \supset \langle U \rangle^G_\pi & \langle U \rangle^{\neg F}_{0.\pi} = \langle U \rangle^F_\pi \\
\langle U \rangle^{F \vee G}_{0.\pi} = G \vee \langle U \rangle^F_\pi & \langle U \rangle^{F \vee G}_{1.\pi} = F \vee \langle U \rangle^G_\pi & \langle U \rangle^F_\varepsilon = U
\end{array}$$

Roughly, the formula $\langle U \rangle^F_\pi$ says "U is true at the position π in F". Note that this formula does not depend on the subformula $F|_\pi$. Also note that an existential quantifier in F produces a universal quantifier in the local image. This is not accidental: a locally valid statement about a bound variable must hold for every possible value of the variable, regardless of the binder.

For a position $\pi \in \Pi_{\mathbf{t}}(F)$, we define $\langle U \rangle^F_\pi$ to be $\langle U \rangle^F_{\widehat{\pi}}$.

Example 2.2.1. Let F be the formula

$$\begin{aligned}
&\forall x \, (x \in \mathbb{N} \supset \forall n \, (n \in \mathbb{N} \supset (x \approx \mathbf{fib}(n) \equiv \\
&\qquad\qquad \equiv ((n \leq 1 \wedge x \approx 1) \vee x \approx (\mathbf{fib}(n-1) + \mathbf{fib}(n-2))))))
\end{aligned}$$

This formula represents a recursive definition. We want to verify that the arguments $(n-1)$ and $(n-2)$ satisfy the guards of the definition and are strictly less than n.

Consider the second argument. Let us denote by π its position, 0.1.0.1.1.1.1.1.0. We want to prove $\langle (n-2) \in \mathbb{N} \wedge (n-2) < n \rangle^F_\pi$. The latter formula is equal to

$$\forall x \, (x \in \mathbb{N} \supset \forall n \, (n \in \mathbb{N} \supset ((n \leq 1 \wedge x \approx 1) \vee ((n-2) \in \mathbb{N} \wedge (n-2) < n))))$$

But this formula is false given $n = x = 0$. And that reveals an error in our definition: $x = 0$ makes false the left side of the disjunction $F|_{0.1.0.1.1}$, so we have to consider the right side with $n = 0$ in order to evaluate the truth value of the whole disjunction. Now it is easy to build a good definition F' of **fib**:

$$\begin{aligned}
&\forall x \, (x \in \mathbb{N} \supset \forall n \, (n \in \mathbb{N} \supset (x \approx \mathbf{fib}(n) \equiv \\
&\qquad\qquad \equiv ((n \leq 1 \wedge x \approx 1) \vee (n \geq 2 \wedge x \approx (\mathbf{fib}(n-1) + \mathbf{fib}(n-2)))))))
\end{aligned}$$

Obviously, the local image $\langle (n-2) \in \mathbb{N} \wedge (n-2) < n \rangle^{F'}_{0.1.0.1.1.1.1.1.1.0}$ is a valid formula:

$$\begin{aligned}
&\forall x \, (x \in \mathbb{N} \supset \forall n \, (n \in \mathbb{N} \supset \\
&\qquad\qquad \supset ((n \leq 1 \wedge x \approx 1) \vee (n \geq 2 \supset ((n-2) \in \mathbb{N} \wedge (n-2) < n)))))
\end{aligned}$$

Lemma 2.2.2. *For any F, $\pi \in \Pi(F)$, and a closed formula U, $U \vdash \langle U \rangle^F_\pi$.*

Proof. The formula $\langle U \rangle^F_\pi$ is equivalent to a universally quantified disjunction and U is a part of this disjunction. □

Lemma 2.2.3. (a) $\vdash \langle U \supset V \rangle^F_\pi \supset (\langle U \rangle^F_\pi \supset \langle V \rangle^F_\pi)$ (b) $\vdash \langle \forall x \, U \rangle^F_\pi \supset \langle U[t/x] \rangle^F_\pi$

The both statements of Lemma 2.2.3 can be proved by a simple induction on the length of π, in the same way as Theorem 2.2.7 below.

The lemmas above show that we can consistently reason about local properties. They are powerful enough to prove some interesting corollaries.

Corollary 2.2.4. $\vdash \langle U \equiv V \rangle^F_\pi \supset (\langle U \rangle^F_\pi \equiv \langle V \rangle^F_\pi)$

Proof. By Lemma 2.2.2 we have $\vdash \langle (U \equiv V) \supset (U \supset V) \rangle^F_\pi$. Hence by Lemma 2.2.3(a), $\vdash \langle (U \equiv V) \rangle^F_\pi \supset (\langle U \supset V \rangle^F_\pi)$. Again by local *modus ponens*, $\vdash \langle (U \equiv V) \rangle^F_\pi \supset (\langle U \rangle^F_\pi \supset \langle V \rangle^F_\pi)$. In the same way, $\vdash \langle (U \equiv V) \rangle^F_\pi \supset (\langle V \rangle^F_\pi \supset \langle U \rangle^F_\pi)$. □

Corollary 2.2.5. $\vdash \langle U \wedge V\rangle_\pi^F \equiv (\langle U\rangle_\pi^F \wedge \langle V\rangle_\pi^F)$

Proof. In order to prove the necessity, we take the propositional tautologies $(U \wedge V) \supset U$ and $(U \wedge V) \supset V$. In order to prove the sufficiency, we take the propositional tautology $U \supset (V \supset (U \wedge V))$. Then we "immerse" a chosen tautology inside the formula F by Lemma 2.2.2 and apply Lemma 2.2.3(a). $\square$

Corollary 2.2.6. *For any quantifier-free "form" $C[\,]$,*

$$\vdash \big(\langle U_1 \equiv V_1\rangle_\pi^F \wedge \cdots \wedge \langle U_n \equiv V_n\rangle_\pi^F \wedge \langle t_1 \approx s_1\rangle_\pi^F \wedge \cdots \wedge \langle t_m \approx s_m\rangle_\pi^F\big) \supset$$
$$\supset \langle C[U_1,\dots,U_n,t_1,\dots,t_m] \equiv C[V_1,\dots,V_n,s_1,\dots,s_m]\rangle_\pi^F$$

The term "form" stands here for a formula with "holes", in which formulas or terms can be inserted, completing the form up to a well-formed formula. The corollary can be proved similarly to previous statements.

The key property of local images is given by the following theorem.

Theorem 2.2.7. *For any formulas F, U, V*

$$\begin{aligned}
\pi \in \Pi_{\mathbf{F}}(F) \quad &\Rightarrow \quad \vdash \langle U \equiv V\rangle_\pi^F \supset (F[U]_\pi \equiv F[V]_\pi)\\
\pi \in \Pi_{\mathbf{F}}^+(F) \quad &\Rightarrow \quad \vdash \langle U \supset V\rangle_\pi^F \supset (F[U]_\pi \supset F[V]_\pi)\\
\pi \in \Pi_{\mathbf{F}}^-(F) \quad &\Rightarrow \quad \vdash \langle V \supset U\rangle_\pi^F \supset (F[U]_\pi \supset F[V]_\pi)
\end{aligned}$$

Proof. We will prove only the first statement, the other two can be demonstrated in a similar fashion. We proceed by induction on the length of π. In the base case ($\pi = \epsilon$), the claim is obvious. Otherwise, we consider four main cases. In every case we formally deduce the claim from the induction hypothesis by definition of a local image.

Case $F = (\neg G)$, $\pi = 0.\tau$ (similarly for $F = (G \equiv H)$):

$$\begin{aligned}
&\vdash \langle U \equiv V\rangle_\tau^G \supset (G[U]_\tau \equiv G[V]_\tau) \quad \Rightarrow\\
&\qquad \Rightarrow \quad \vdash \langle U \equiv V\rangle_\tau^G \supset (\neg G[U]_\tau \equiv \neg G[V]_\tau) \quad \Rightarrow\\
&\qquad\qquad \Rightarrow \quad \vdash \langle U \equiv V\rangle_\pi^F \supset (F[U]_\pi \equiv F[V]_\pi)
\end{aligned}$$

Case $F = (G \supset H)$, $\pi = 0.\tau$ (similarly for $F = (G \vee H)$):

$$\begin{aligned}
&\vdash \langle U \equiv V\rangle_\tau^G \supset (G[U]_\tau \equiv G[V]_\tau) \quad \Rightarrow\\
&\qquad \Rightarrow \quad \vdash (H \vee \langle U \equiv V\rangle_\tau^G) \supset (H \vee (G[U]_\tau \equiv G[V]_\tau)) \quad \Rightarrow\\
&\qquad \Rightarrow \quad \vdash \langle U \equiv V\rangle_\pi^F \supset ((G[U]_\tau \supset H) \equiv (G[V]_\tau \supset H)) \quad \Rightarrow\\
&\qquad\qquad \Rightarrow \quad \vdash \langle U \equiv V\rangle_\pi^F \supset (F[U]_\pi \equiv F[V]_\pi)
\end{aligned}$$

Case $F = (H \supset G)$, $\pi = 1.\tau$ (similarly for $F = (G \wedge H)$):

$$\begin{aligned}
&\vdash \langle U \equiv V\rangle_\tau^G \supset (G[U]_\tau \equiv G[V]_\tau) \quad \Rightarrow\\
&\qquad \Rightarrow \quad \vdash (H \supset \langle U \equiv V\rangle_\tau^G) \supset (H \supset (G[U]_\tau \equiv G[V]_\tau)) \quad \Rightarrow\\
&\qquad \Rightarrow \quad \vdash \langle U \equiv V\rangle_\pi^F \supset ((H \supset G[U]_\tau) \equiv (H \supset G[V]_\tau)) \quad \Rightarrow\\
&\qquad\qquad \Rightarrow \quad \vdash \langle U \equiv V\rangle_\pi^F \supset (F[U]_\pi \equiv F[V]_\pi)
\end{aligned}$$

Case $F = (\exists x\, G)$, $\pi = 0.\tau$ (similarly for $F = (\forall x\, G)$):

$$\begin{aligned}
&\vdash \langle U \equiv V\rangle_\tau^G \supset (G[U]_\tau \equiv G[V]_\tau) \quad \Rightarrow\\
&\qquad \Rightarrow \quad \vdash \forall x\, \langle U \equiv V\rangle_\tau^G \supset \forall x\, (G[U]_\tau \equiv G[V]_\tau) \quad \Rightarrow\\
&\qquad \Rightarrow \quad \vdash \langle U \equiv V\rangle_\pi^F \supset (\exists x\, G[U]_\tau \equiv \exists x\, G[V]_\tau) \quad \Rightarrow\\
&\qquad\qquad \Rightarrow \quad \vdash \langle U \equiv V\rangle_\pi^F \supset (F[U]_\pi \equiv F[V]_\pi)
\end{aligned}$$

$\square$

Thus we can safely replace subformulas not only by equivalent formulas but by locally equivalent ones as well. Note that the inverse of Theorem 2.2.7 holds in the propositional logic: $\vdash_0 \langle U \equiv V\rangle_\pi^F \equiv (F[U]_\pi \equiv F[V]_\pi)$. The local equivalence is there a *criterion* of the substitutional equivalence. It is not the case for the first-order logic, where $(\exists x\, x \approx 0)$ is equivalent to $(\exists x\, x \approx 1)$.

Corollary 2.2.8. *For any formula F, a position $\pi \in \Pi_{\mathbf{t}}(F)$, and terms s and t,*

$$\vdash \langle s \approx t\rangle_\pi^F \supset (F[s]_\pi \equiv F[t]_\pi)$$

Follows from Theorem 2.2.7 and Corollary 2.2.6.

Corollary 2.2.9. *For any formula F, a position $\pi \in \Pi_{\mathbf{F}}(F)$, and formulas U and V*

$$\vdash \langle U\rangle_\pi^F \supset (F[V]_\pi \equiv F[U \wedge V]_\pi) \qquad \vdash \langle U\rangle_\pi^F \supset (F[V]_\pi \equiv F[U \supset V]_\pi)$$
$$\vdash \langle V \supset U\rangle_\pi^F \supset (F[V]_\pi \equiv F[U \wedge V]_\pi) \qquad \vdash \langle U \supset V\rangle_\pi^F \supset (F[V]_\pi \equiv F[U \vee V]_\pi)$$

Consider a closed formula H of the form $\forall \vec{x}\,(C \supset (A \equiv D))$, where A is an atomic formula. Consider a formula F and a position $\pi \in \Pi_{\mathbf{A}}(F)$ such that $F|_\pi = A\sigma$ for some substitution σ. If we can prove $\langle C\sigma\rangle_\pi^F$, then we have $\langle A\sigma \equiv D\sigma\rangle_\pi^F$ by Lemma 2.2.2 and Corollary 2.2.3 (provided that H is among the premises). Then we can replace $A\sigma$ with $D\sigma$ by Theorem 2.2.7.

Returning to Example 2.2.1, we can guarantee that such an expansion is always possible (since $\langle n-1 \in \mathbb{N} \wedge n-2 \in \mathbb{N}\rangle_\pi^F$ holds) and is never infinite (since $\langle n-1 < n \wedge n-2 < n\rangle_\pi^F$ holds).

Directed local validity

The notion of a local image introduced above has a disadvantage: it is not invariant w.r.t. transformations at adjacent positions.

Example 2.2.10. Since $\langle A\rangle_0^{A\wedge A}$ is true, $(A \wedge A)$ is equivalent to $(\top \wedge A)$ by Theorem 2.2.7. But $\langle A\rangle_1^{A\wedge A}$ is also true, whereas $\langle A\rangle_1^{\top\wedge A}$ is not.

Generally, we can build a formula F whose two subformulas U and V assure certain local properties for each other. Using these properties, we replace U with a locally equivalent formula U'. But thus we can lose the local properties of V.

This would not play an important role if we used local properties just for one-time transformations: say, simplifications. Indeed, one should check that simplification is possible just before applying it.

But let us recall (and also anticipate) that ontological correctness of the formulas and parts of a formal text is formulated in terms of local properties. Moreover, evidence collection, definition expansion and a lot of other methods of the reasoner (described in the next chapter) rely on sets of local properties once collected and assigned to particular occurrences. If some transformation of a formula containing that occurrence (say, inserting its instance into some logical successor down the text — which happens during definition expansion or lemma application) falsifies some important local properties, we would have to discard what we once knew about that occurrence and reevaluate it in its new form and context. This is something that we definitely want to avoid.

To that aim, we change the definition of a local image in such a way that only the formulas at *precedent* positions get into the context. Psychologically, this is natural, since type declarations, guards, limits and other preliminary assertions are usually written before "significant" formulas.

The *directed local image* of a formula U w.r.t. a formula F and a position $\pi \in \Pi_{\mathbf{F}}(F)$, denoted $\langle\!|U|\!\rangle_\pi^F$, is defined as follows:

$$\langle\!|U|\!\rangle_{0.\pi}^{F\equiv G} = \langle\!|U|\!\rangle_\pi^F \qquad \langle\!|U|\!\rangle_{1.\pi}^{F\equiv G} = \langle\!|U|\!\rangle_\pi^G \qquad \langle\!|U|\!\rangle_{0.\pi}^{\forall x F} = \forall x\, \langle\!|U|\!\rangle_\pi^F$$
$$\langle\!|U|\!\rangle_{0.\pi}^{F\supset G} = \langle\!|U|\!\rangle_\pi^F \qquad \langle\!|U|\!\rangle_{1.\pi}^{F\supset G} = F \supset \langle\!|U|\!\rangle_\pi^G \qquad \langle\!|U|\!\rangle_{0.\pi}^{\exists x F} = \forall x\, \langle\!|U|\!\rangle_\pi^F$$
$$\langle\!|U|\!\rangle_{0.\pi}^{F\wedge G} = \langle\!|U|\!\rangle_\pi^F \qquad \langle\!|U|\!\rangle_{1.\pi}^{F\wedge G} = F \supset \langle\!|U|\!\rangle_\pi^G \qquad \langle\!|U|\!\rangle_{0.\pi}^{\neg F} = \langle\!|U|\!\rangle_\pi^F$$
$$\langle\!|U|\!\rangle_{0.\pi}^{F\vee G} = \langle\!|U|\!\rangle_\pi^F \qquad \langle\!|U|\!\rangle_{1.\pi}^{F\vee G} = F \vee \langle\!|U|\!\rangle_\pi^G \qquad \langle\!|U|\!\rangle_\varepsilon^F = U$$

For a position $\pi \in \Pi_{\mathbf{t}}(F)$, we define $\langle\!|U|\!\rangle^F_\pi$ to be $\langle\!|U|\!\rangle^F_{\hat{\pi}}$.

First, note that all statements proved so far about "indirected" images hold for directed ones, too. In some sense, directed image is just a reduction, with some conditions and alternatives eliminated. This is illustrated by the following trivial lemma.

Lemma 2.2.11. $\vdash \langle\!|U|\!\rangle^F_\pi \supset \langle U\rangle^F_\pi$

Theorem 2.2.12. *For any formula F and positions $\pi, \tau \in \Pi_{\mathbf{F}}(F)$, such that τ is outer to π,*

$$\vdash \langle\!|U \equiv V|\!\rangle^F_\pi \supset \left(\langle\!|W|\!\rangle^{F[U]_\pi}_\tau \equiv \langle\!|W|\!\rangle^{F[V]_\pi}_\tau\right)$$

Proof. Once again, we proceed by induction on the length of π. It is easy to see that, if τ textually precedes π or π is a subposition of τ, then the formulas $\langle\!|W|\!\rangle^{F[U]_\pi}_\tau$ and $\langle\!|W|\!\rangle^{F[V]_\pi}_\tau$ are identical. So we can suppose that π textually precedes τ, that is, there exist ω, μ, and η such that $\pi = \omega.0.\mu$ and $\tau = \omega.1.\eta$. Following the method of Theorem 2.2.7, we can reduce our problem to

$$\vdash \langle\!|U \equiv V|\!\rangle^{G*H}_{0.\mu} \supset \left(\langle\!|W|\!\rangle^{(G*H)[U]_{0.\mu}}_{1.\eta} \equiv \langle\!|W|\!\rangle^{(G*H)[V]_{0.\mu}}_{1.\eta}\right)$$

where $(G * H) = F|_\omega$. The latter is equivalent to

$$\vdash \langle\!|U \equiv V|\!\rangle^{G}_{\mu} \supset \left(\langle\!|W|\!\rangle^{G[U]_\mu * H}_{1.\eta} \equiv \langle\!|W|\!\rangle^{G[V]_\mu * H}_{1.\eta}\right)$$

and then (given that $*$ is not equivalence, which is a trivial case) to

$$\vdash \langle\!|U \equiv V|\!\rangle^{G}_{\mu} \supset \left((G[U]_\mu \dagger \langle\!|W|\!\rangle^H_\eta) \equiv (G[V]_\mu \dagger \langle\!|W|\!\rangle^H_\eta)\right)$$

where $\dagger$ is $\vee$ if $*$ is $\vee$, and $\dagger$ is $\supset$ otherwise. By Theorem 2.2.7 and Lemma 2.2.11, $\langle\!|U \equiv V|\!\rangle^G_\mu$ implies $(G[U]_\mu \equiv G[V]_\mu)$, hence the claim is proved. $\square$

Corollary 2.2.13. *For any formula F and $\pi, \tau \in \Pi_{\mathbf{t}}(F)$, such that τ is outer to π,*

$$\vdash \langle\!|s \approx t|\!\rangle^F_\pi \supset \left(\langle\!|W|\!\rangle^{F[s]_\pi}_\tau \equiv \langle\!|W|\!\rangle^{F[t]_\pi}_\tau\right)$$

Previous work

Reasoning inside a formula is not a new idea. To our knowledge, related concepts were first introduced by L.G. Monk in [6] and were further developed in [1]. P.J. Robinson and J. Staples proposed a full-fledged inference system (so called "window inference") [8] which operated on subexpressions taking the surrounding context into account. This inference system has been generalized and extended by J. Grundy [3].

A common trait of the mentioned approaches is that the local context of an occurrence is represented by a set of formulas which are regarded as local premises for the position in question. Special inference rules are then needed to handle a local context and, what is worse, some "strong" transformations, e.g. replacing $A \vee B$ with $\neg A \supset B$, are required. The notion of local image, as described in this paper, seems to be lighter and less intrusive. In particular, the results of Section 2.2.2 are valid in intuitionistic logic, while the local contexts of [6] cannot be adapted for intuitionistic logic in any obvious way.

Moreover, the definition of a local image can be easily extended to a (uni)-modal language: $\langle U\rangle^{\Box F}_{0.\pi} = \Box\langle U\rangle^F_\pi$ and $\langle U\rangle^{\Diamond F}_{0.\pi} = \Box\langle U\rangle^F_\pi$, and similarly for directed images. Then the statements of Section 2.2.2 can be proved in the modal logic **K**, hence in any normal modal logic.

2.2.3 Formula transformations

In this subsection we will introduce several simple transformations over formulas which are employed in the reasoner's procedures. First of all, let us describe, in a quite general fashion, what do we mean by a "formula transformation" and what kind of properties such a transformation must have.

An atomic formula transformation, in our setting, is a replacement of a subformula or a term in a given position with some other formula or term. Put formally, an atomic transformation is what, being given an occurrence $F[U]_\pi$, returns $F[V]_\pi$. A non-atomic transformation is just a chain of atomic ones. Any particular atomic transformation must satisfy some constraints to be useful for our purposes:

(i) The formula produced by the transformation must be in some clear logical relation with the original one: for example, be equivalent to it or follow from it, or, conversely, entail it (all this, possibly, in view of some premises).

(ii) The transformation must preserve all the local properties at outer (with respect to π) positions in F. Why this and the next condition are important, has been explained in the previous section, when we introduced directed local images.

(iii) Since the new subformula V must have came from somewhere (from premises or from some part of F, or from U itself), the local properties of the "original" must be somehow transferred to V in $F[V]_\pi$. The way in which such a transfer must occur depends on a transformation in question.

And speaking about non-atomic transformations:

(iv) All the atomic transformations in the chain must be consistent with each other (that is, if one give us a logical consequence of the initial formula, the other must produce a consequence, too).

Equivalent transformations

A equivalent transformation is one which produces a logical equivalent of the initial formula. A generic result on this is given in the previous section in Theorems 2.2.7 and 2.2.12, namely: replacement with a locally equivalent subformula produces an equivalent result and preserves directed local properties at outer positions. Thus, any transformation consisting in such replacement automatically fulfils the first and the second condition above.

Below we consider two useful kinds of equivalent transformation and see how well they cope with the condition (iii).

Boolean contraction. The *(atomic) boolean contraction* of a formula H, denoted $\mathsf{C}\,H$, is defined as follows:

$$\begin{array}{llll}
\mathsf{C}(F \equiv \top) = F & \mathsf{C}(F \supset \top) = \top & \mathsf{C}(F \wedge \top) = F & \mathsf{C}(F \vee \top) = \top \\
\mathsf{C}(\top \equiv F) = F & \mathsf{C}(\top \supset F) = F & \mathsf{C}(\top \wedge F) = F & \mathsf{C}(\top \vee F) = \top \\
\mathsf{C}(F \equiv \bot) = \neg F & \mathsf{C}(F \supset \bot) = \neg F & \mathsf{C}(F \wedge \bot) = \bot & \mathsf{C}(F \vee \bot) = F \\
\mathsf{C}(\bot \equiv F) = \neg F & \mathsf{C}(\bot \supset F) = \top & \mathsf{C}(\bot \wedge F) = \bot & \mathsf{C}(\bot \vee F) = F \\
\mathsf{C}(\forall x\, \top) = \top & \mathsf{C}(\exists x\, \top) = \top & \mathsf{C}(\neg \top) = \bot & \mathsf{C}\,H = H \\
\mathsf{C}(\forall x\, \bot) = \bot & \mathsf{C}(\exists x\, \bot) = \bot & \mathsf{C}(\neg \bot) = \top & \text{(for any other } H\text{)}
\end{array}$$

The *(full) boolean contraction* $\bar{\mathsf{C}}\,H$ is obtained by applying C throughout H:

$$\bar{\mathsf{C}}(F * G) = \mathsf{C}((\bar{\mathsf{C}}\,F) * (\bar{\mathsf{C}}\,G)) \qquad \bar{\mathsf{C}}(* F) = \mathsf{C}(* (\bar{\mathsf{C}}\,F)) \qquad \bar{\mathsf{C}}\,A = A$$

Obviously, the full contraction is merely the chain of atomic contractions applied to subformulas.

Lemma 2.2.14. *For any formula H, $\vdash H \equiv \mathsf{C}\,H$ and $\vdash H \equiv \mathsf{C}^*\,H$.*

The proof is trivial. Thus, boolean contraction is indeed an equivalent transformation satisfying the conditions (i) and (ii). The fulfilment of the third condition is demonstrated by the following cumbersome yet simple lemma:

Lemma 2.2.15. *Let H be a formula and π be a position in $\Pi_{\mathbf{F}}(H)$. Let F be a formula such that $\mathsf{C}\,F$ is not a boolean constant. If F is not one of $G \equiv \bot$, $\bot \equiv G$, or $G \supset \bot$, then for any position $\tau \in \Pi(\mathsf{C}\,F)$ there is a position $i.\tau \in \Pi(F)$ such that $F|_{i.\tau} = (\mathsf{C}\,F)|_\tau$ and*

$$\vdash \llparenthesis U \rrparenthesis^{H[F]_\pi}_{\pi.i.\tau} \supset \llparenthesis U \rrparenthesis^{H[\mathsf{C}\,F]_\pi}_{\pi.\tau}$$

Otherwise, if F is one of $G \equiv \bot$, $\bot \equiv G$, or $G \supset \bot$, then for any position $0.\tau \in \Pi(\mathtt{C}\,F)$ there is a position $i.\tau \in \Pi(F)$ such that $F|_{i.\tau} = (\mathtt{C}\,F)|_{0.\tau}$ and

$$\vdash \langle\!\langle U \rangle\!\rangle^{H[F]_\pi}_{\pi.i.\tau} \supset \langle\!\langle U \rangle\!\rangle^{H[\mathtt{C}\,F]_\pi}_{\pi.0.\tau}$$

Proof. Let $F = \top \wedge G$ (other cases can be handled in a similar way). Then $\mathtt{C}\,F = G$ and we set $i = 1$. Then the local image $\langle\!\langle U \rangle\!\rangle^{H[F]_\pi}_{\pi.1.\tau}$ is $\langle\!\langle \top \supset \langle\!\langle U \rangle\!\rangle^{G}_{\tau} \rangle\!\rangle^{H}_{\pi}$ which is equivalent to $\langle\!\langle \langle\!\langle U \rangle\!\rangle^{G}_{\tau} \rangle\!\rangle^{H}_{\pi}$, and therefore to $\langle\!\langle U \rangle\!\rangle^{H[G]_\pi}_{\pi.\tau}$. □

Thus, every occurrence in a contracted formula inherits its local properties from some corresponding occurrence in the initial formula.

Obviously, in the proof above, we could immediately take $\pi = \epsilon$ and ignore the formula H, since it produces the same context in the local images at the anthecedent and the succedent of the claimed implication. This is what we will do in subsequent proofs of this kind.

Contraction in view. Let us introduce a more interesting kind of equivalent transformation. To that end, we shall introduce two more varieties of positions. The sets of positions $\Pi_\wedge$ and $\Pi_\vee$ are used to separate the components of outer conjunctions and disjunctions:

$$\begin{aligned}
\Pi_\wedge(F \equiv G) &= \{\epsilon\} & \Pi_\wedge(F \supset G) &= \{\epsilon\}\\
\Pi_\wedge(F \wedge G) &= \{\epsilon\} \cup 0.\Pi_\wedge(F) \cup 1.\Pi_\wedge(G) & \Pi_\wedge(F \vee G) &= \{\epsilon\}\\
\Pi_\wedge(\forall x\, F) &= \{\epsilon\} \cup 0.\{\,\tau \mid \tau \in \Pi_\wedge(F) \wedge x \notin \mathcal{FV}(F|_\tau)\} & \Pi_\wedge(\neg F) &= \{\epsilon\} \cup 0.\Pi_\vee(F)\\
\Pi_\wedge(\exists x\, F) &= \{\epsilon\} \cup 0.\{\,\tau \mid \tau \in \Pi_\wedge(F) \wedge x \notin \mathcal{FV}(F|_\tau)\} & \Pi_\wedge(A) &= \{\epsilon\}
\end{aligned}$$

$$\begin{aligned}
\Pi_\vee(F \supset G) &= \{\epsilon\} \cup 0.\Pi_\wedge(F) \cup 1.\Pi_\vee(G) & \Pi_\vee(F \equiv G) &= \{\epsilon\}\\
\Pi_\vee(F \vee G) &= \{\epsilon\} \cup 0.\Pi_\vee(F) \cup 1.\Pi_\vee(G) & \Pi_\vee(F \wedge G) &= \{\epsilon\}\\
\Pi_\vee(\forall x\, F) &= \{\epsilon\} \cup 0.\{\,\tau \mid \tau \in \Pi_\vee(F) \wedge x \notin \mathcal{FV}(F|_\tau)\} & \Pi_\vee(\neg F) &= \{\epsilon\} \cup 0.\Pi_\wedge(F)\\
\Pi_\vee(\exists x\, F) &= \{\epsilon\} \cup 0.\{\,\tau \mid \tau \in \Pi_\vee(F) \wedge x \notin \mathcal{FV}(F|_\tau)\} & \Pi_\vee(A) &= \{\epsilon\}
\end{aligned}$$

As before, we split these sets into positive, negative, and neutral parts:

$$\begin{array}{lll}
\Pi^+_\wedge(F) = \Pi_\wedge(F) \cap \Pi^+(F) & \Pi^-_\wedge(F) = \Pi_\wedge(F) \cap \Pi^-(F) & \Pi^\circ_\wedge(F) = \Pi_\wedge(F) \cap \Pi^\circ(F)\\
\Pi^+_\vee(F) = \Pi_\vee(F) \cap \Pi^+(F) & \Pi^-_\vee(F) = \Pi_\vee(F) \cap \Pi^-(F) & \Pi^\circ_\vee(F) = \Pi_\vee(F) \cap \Pi^\circ(F)
\end{array}$$

Lemma 2.2.16. *For any formula F,*

$$\begin{array}{ll}
\tau \in \Pi^+_\wedge(F) \;\Rightarrow\; \vdash F \supset F|_\tau & \qquad \tau \in \Pi^-_\wedge(F) \;\Rightarrow\; \vdash F \supset \neg(F|_\tau)\\
\tau \in \Pi^+_\vee(F) \;\Rightarrow\; \vdash F|_\tau \supset F & \qquad \tau \in \Pi^-_\vee(F) \;\Rightarrow\; \vdash \neg(F|_\tau) \supset F
\end{array}$$

Proof. The claim can be easily proved by induction on the length of τ. □

The *(atomic) contraction of F in view of a formula H*, denoted $\mathtt{C}_H\,F$, is defined as follows:

$$\begin{array}{ll}
\mathtt{C}_H\,F = \top & \text{if } F = H|_\pi \text{ for some } \pi \in \Pi^+_\wedge(H)\\
\mathtt{C}_H\,F = \bot & \text{if } F = H|_\pi \text{ for some } \pi \in \Pi^-_\wedge(H)\\
\mathtt{C}_H\,F = F & \text{otherwise}
\end{array}$$

The *(full) contraction* $\overline{\mathtt{C}}_H\,F$ is obtained by applying $\mathtt{C}_H$ throughout F:

$$\begin{array}{rll}
\overline{\mathtt{C}}_H\,F &= \mathtt{C}_H\,F & \text{if } \mathtt{C}_H\,F \text{ is a boolean constant}\\
\overline{\mathtt{C}}_H(F * G) &= (\overline{\mathtt{C}}_H\,F) * (\overline{\mathtt{C}}_H\,G) & \text{otherwise}\\
\overline{\mathtt{C}}_H(*\,F) &= *\,(\overline{\mathtt{C}}_H\,F) &\\
\overline{\mathtt{C}}_H\,A &= A &
\end{array}$$

We do not perform unification or matching when comparing F to $H|_\pi$ but we can rename bound variables if needed. The bound variables of F should not be confused with the free variables from H, i.e. $\overline{\mathsf{C}}_{P(x)}(\exists x\, P(x))$ is $\exists x\, P(x)$ and not $\exists x\, \top$.

In view of H, $\overline{\mathsf{C}}_H\, F$ is obtained from F by a number of locally equivalent replacements (by Lemma 2.2.16). Therefore, the result is equivalent to the initial formula:

Lemma 2.2.17. *For any formulas F, H, we have $H \vdash F \equiv \mathsf{C}_H\, F$ and $H \vdash F \equiv \overline{\mathsf{C}}_H\, F$.*

Thus, contraction in view of a formula is an equivalent tranformation. Moreover, since it does not introduce new formulas (but the boolean constants), we immediately obtain by Theorem 2.2.12 the following lemma:

Lemma 2.2.18. *For every position $\pi \in \Pi(\overline{\mathsf{C}}_H\, F)$, $H \vdash \langle\!| U |\!\rangle_\pi^F \supset \langle\!| U |\!\rangle_\pi^{(\overline{\mathsf{C}}_H\, F)}$.*

Contraction in view of a formula is trivially extended to contraction in view of a set of formulas. It is natural to combine this transformation with boolean contraction so that the latter eliminates the boolean constants introduced by the former.

Remark. Please note that our way to define the atomic contraction $\mathsf{C}_H\, F$ is neither the only possible nor the best one. There are much better ways to observe that a formula F (or its negation) follows from H than comparing F with subformulas in H. What is important, however, is that the properties of C_H and $\overline{\mathsf{C}}_H$ described by Lemmas 2.2.17 and 2.2.18 do not depend upon the method of establishing the entailment. As soon as C_H merely replaces a formula with an equivalent (in view of H) boolean constant, we have a perfectly valid atomic equivalent transformation satisfying the conditions (i)–(iv).

Inequivalent transformations

Inequivalent transformations are tricky. While Theorem 2.2.7 suffices to satisfy the condition (i), the corresponding extension of Theorem 2.2.12 does not hold: replacement with an inequivalent subformula still can falsify local properties at outer positions.

Consider the conjunction $A \wedge B$. By Theorem 2.2.7, we can replace A with $\top$ (since $\top$ follows from A) and obtain a formula $\top \wedge B$ which follows from the initial conjunction. Yet, by this transformation, the occurrence of B loses a local property, namely A. On the other hand, we can safely replace A with a formula which entails A, say $A \wedge C$: the resulting formula will entail $A \wedge B$, and the occurrence of B will inherit all the local properties from the initial formula (and also get some new ones).

Thus, we must restrict the notion of inequivalent transformation. Let H be a formula, E, a formula or a term, $s \in \{+, -\}$, a polarity, and π, a position in $\Pi_{\mathbf{F}}(H)$ if E is a formula, or in $\Pi_{\mathbf{t}}(H)$, otherwise. Then *signed replacement* of $H|_\pi$ with E, denoted $H[E]_\pi^s$ is defined as follows:

$$\begin{array}{rcl}
(\forall x\, F)[E]^+_{0.\tau} = \forall x\, F[E]^+_\tau & (\exists x\, F)[E]^+_{0.\tau} = \exists x\, F[E]^+_\tau & (F \equiv G)[E]^+_\tau = F \equiv G \\
(F \supset G)[E]^+_{0.\tau} = F[E]^-_\tau \supset G & (F \vee G)[E]^+_{0.\tau} = F[E]^+_\tau \vee G & (F \wedge G)[E]^+_{0.\tau} = F[E]^+_\tau \wedge \top \\
(F \supset G)[E]^+_{1.\tau} = F \supset G[E]^+_\tau & (F \vee G)[E]^+_{1.\tau} = F \vee G[E]^+_\tau & (F \wedge G)[E]^+_{1.\tau} = F \wedge G[E]^+_\tau \\
(\neg F)[E]^+_{0.\tau} = \neg F[E]^-_\tau & F[E]^+_\varepsilon = E & A[E]^+_\tau = A[E]_\tau
\end{array}$$

$$\begin{array}{rcl}
(\forall x\, F)[E]^-_{0.\tau} = \forall x\, F[E]^-_\tau & (\exists x\, F)[E]^-_{0.\tau} = \exists x\, F[E]^-_\tau & (F \equiv G)[E]^-_\tau = F \equiv G \\
(F \supset G)[E]^-_{0.\tau} = F[E]^+_\tau \supset \bot & (F \vee G)[E]^-_{0.\tau} = F[E]^-_\tau \vee \bot & (F \wedge G)[E]^-_{0.\tau} = F[E]^-_\tau \wedge G \\
(F \supset G)[E]^-_{1.\tau} = F \supset G[E]^-_\tau & (F \vee G)[E]^-_{1.\tau} = F \vee G[E]^-_\tau & (F \wedge G)[E]^-_{1.\tau} = F \wedge G[E]^-_\tau \\
(\neg F)[E]^-_{0.\tau} = \neg F[E]^+_\tau & F[E]^-_\varepsilon = E & A[E]^-_\tau = A[E]_\tau
\end{array}$$

Signed replacement erases (replacing them with boolean constants) exactly those subformulas which lose their local properties during a corresponding inequivalent transformation.

Lemma 2.2.19. *For any formula F, a formula or term E, and a corresponding position π,*

$$\vdash F[E]_\pi \supset F[E]^+_\pi \qquad\qquad \vdash F[E]^-_\pi \supset F[E]_\pi$$

Proof. The formula $F[E]^+_\pi$ can be obtained from $F[E]_\pi$ by a number of subformula replacements with $\top$ in positive positions of F, and with $\bot$ in negative positions of F. For every such replacement, Theorem 2.2.7 is applicable. The second claim is proved in the same way. $\square$

Corollary 2.2.20. *For any formulas F, U, V,*

$$\begin{array}{lcl}
\pi \in \Pi^+_{\mathbf{F}}(F) & \Rightarrow & \vdash \langle\!|U \supset V|\!\rangle^F_\pi \supset (F[U]_\pi \supset F[V]^+_\pi) \\
\pi \in \Pi^-_{\mathbf{F}}(F) & \Rightarrow & \vdash \langle\!|V \supset U|\!\rangle^F_\pi \supset (F[U]_\pi \supset F[V]^+_\pi) \\
\pi \in \Pi^+_{\mathbf{F}}(F) & \Rightarrow & \vdash \langle\!|V \supset U|\!\rangle^F_\pi \supset (F[V]^-_\pi \supset F[U]_\pi) \\
\pi \in \Pi^-_{\mathbf{F}}(F) & \Rightarrow & \vdash \langle\!|U \supset V|\!\rangle^F_\pi \supset (F[V]^-_\pi \supset F[U]_\pi)
\end{array}$$

Now let us show that the condition (ii) is fulfilled by such a "forgetful" transformation.

Theorem 2.2.21. *Let F, U, V, W be formulas, π, a position in $\Pi_{\mathbf{F}}(F)$, and τ, a position in $\Pi(F)$ outer with respect to π. If $(F[V]^+_\pi)|_\tau$ is not a boolean constant, then we have:*

$$\begin{array}{lcl}
\pi \in \Pi^+_{\mathbf{F}}(F) & \Rightarrow & \vdash \langle\!|U \supset V|\!\rangle^F_\pi \supset (\langle\!|W|\!\rangle^{F[U]_\pi}_\tau \supset \langle\!|W|\!\rangle^{F[V]^+_\pi}_\tau) \\
\pi \in \Pi^-_{\mathbf{F}}(F) & \Rightarrow & \vdash \langle\!|V \supset U|\!\rangle^F_\pi \supset (\langle\!|W|\!\rangle^{F[U]_\pi}_\tau \supset \langle\!|W|\!\rangle^{F[V]^+_\pi}_\tau)
\end{array}$$

Similarly, if $(F[V]^-_\pi)|_\tau$ is not a boolean constant,

$$\begin{array}{lcl}
\pi \in \Pi^+_{\mathbf{F}}(F) & \Rightarrow & \vdash \langle\!|V \supset U|\!\rangle^F_\pi \supset (\langle\!|W|\!\rangle^{F[U]_\pi}_\tau \supset \langle\!|W|\!\rangle^{F[V]^-_\pi}_\tau) \\
\pi \in \Pi^-_{\mathbf{F}}(F) & \Rightarrow & \vdash \langle\!|U \supset V|\!\rangle^F_\pi \supset (\langle\!|W|\!\rangle^{F[U]_\pi}_\tau \supset \langle\!|W|\!\rangle^{F[V]^-_\pi}_\tau)
\end{array}$$

Proof. Signed replacement at a position π changes only the subformulas at π and to the right of π. Therefore, if τ is a textual predecessor of π or π is a subposition of τ, then the statement of the theorem is obviously true. Indeed, a directed local image at τ does not depend on subformulas to the right of τ. So, we can suppose that π is a textual predecessor of τ, that is, there exist positions ω, μ, η such that $\pi = \omega.0.\mu$ and $\tau = \omega.1.\eta$. We can also assume that $\omega = \epsilon$, since this part of local image is the same in $\langle\!|W|\!\rangle^{F[U]_\pi}_\tau$ and in $\langle\!|W|\!\rangle^{F[V]^s_\pi}_\tau$. Consider the implication

$$\langle\!|W|\!\rangle^{F[U]_{0.\mu}}_{1.\eta} \supset \langle\!|W|\!\rangle^{F[V]^s_{0.\mu}}_{1.\eta}$$

By definition, this is equal to (given that $F = G * H$)

$$\langle\!|W|\!\rangle^{G[U]_\mu * H}_{1.\eta} \supset \langle\!|W|\!\rangle^{G[V]^s_\mu * H'}_{1.\eta} \tag{†}$$

Now, depending on the connection $*$ and the polarity s, H' can be either H or a boolean constant. However, if H' is a boolean constant then $(F[V]^s_\pi)|_\tau = H'_\eta$ is also a boolean constant. So we can suppose that $H' = H$. We can also suppose that $*$ is not equivalence, since otherwise π would be a neutral position.

Case $F = G \wedge H$. Since H' is H and not a boolean constant, the polarity s must be $-$. Then the implication (†) is equal to

$$(G[U]_\mu \supset \langle\!|W|\!\rangle^H_\eta) \supset (G[V]^-_\mu \supset \langle\!|W|\!\rangle^H_\eta)$$

If π was positive in F, then μ is positive in G. By Corollary 2.2.20, $\langle\!|V \supset U|\!\rangle^F_\pi$ (being equal to $\langle\!|V \supset U|\!\rangle^G_\mu$) implies $(G[V]^-_\mu \supset G[U]_\mu)$ and the claim is proved. The case $\pi \in \Pi^-_{\mathbf{F}}(F)$ is similar.

Case $F = G \supset H$. The polarity s must be $+$ and the implication (†) is equal to

$$(G[U]_\mu \supset \langle\!|W|\!\rangle^H_\eta) \supset (G[V]^-_\mu \supset \langle\!|W|\!\rangle^H_\eta)$$

If π was positive in F, then μ is negative in G. Again, $\langle\!| U \supset V |\!\rangle^F_\pi$ (which is equal to $\langle\!| U \supset V |\!\rangle^G_\mu$) implies $(G[V]^-_\mu \supset G[U]_\mu)$ and the claim is proved. The case $\pi \in \Pi^-_{\mathbf{F}}(F)$ is similar.

Case $F = G \vee H$. The polarity s must be $+$ and the implication (†) is equal to

$$(G[U]_\mu \vee \langle\!| W |\!\rangle^H_\eta) \supset (G[V]^+_\mu \vee \langle\!| W |\!\rangle^H_\eta)$$

If π was positive in F, then μ is positive in G. Again, $\langle\!| U \supset V |\!\rangle^F_\pi$ (which is equal to $\langle\!| U \supset V |\!\rangle^G_\mu$) implies $(G[U]_\mu \supset G[V]^+_\mu)$ and the claim is proved. The case $\pi \in \Pi^-_{\mathbf{F}}(F)$ is similar. □

As a result, the positive replacement allows us to generate succedents, the negative replacement, antecedents, and the both transformations preserve local properties at outer positions (by cutting off ones where they fail to do so).

Local instantiation. One particulary useful varation of inequivalent transformation is instantiation. Let us call an occurrence of a quantifier *eliminable* whenever it is a universal quantifier in a positive position or an existential quantifier in a negative position (we do not define a dual notion for quantifiers in goal formulas since we do not instantiate them). Let us call a variable *instantiable* at a given position in a formula whenever it is bound at that position by an eliminable quantifier.

Consider a formula F, a position $\pi \in \Pi(F)$, and a variable x instantiable at π in F. Let μ be the position of the eliminable quantifier over x. Let t be a term free for substitution into x in the formula $F|_{\mu.0}$. The result of *(atomic) local substitution* of t into x at π in F, denoted $F[[t/x]]^+_\pi$, is the formula $F[(F|_{\mu.0})[t/x]]^+_\mu$. That is, we replace a quantified subformula $F|_\mu$ with the instance $(F|_{\mu.0})[t/x]$ using positive replacement. Note that the result of local substutution stays the same for any π as long as the position μ stays the same.

Lemma 2.2.22. *Let F, π, x, μ, and t be as introduced above. Let η be a position in $\Pi(F|_{\mu.0})$. Let τ be a position in $\Pi(F)$, such that τ is outer with respect to μ and $(F[[t/x]]^+_\pi)|_\tau$ is not a boolean constant. Then the following holds:*

$$\vdash F \supset F[[t/x]]^+_\pi \tag{a}$$

$$\vdash \langle\!| W |\!\rangle^F_\tau \supset \langle\!| W |\!\rangle^{F[[t/x]]^+_\pi}_\tau \tag{b}$$

$$\vdash \langle\!| W |\!\rangle^F_{\mu.0.\eta} \supset \langle\!| W[t/x] |\!\rangle^{F[[t/x]]^+_\pi}_{\mu.\eta} \tag{c}$$

Proof. For the sake of brevity, let F' denote the formula $F[[t/x]]^+_\pi$, G, the subformula $F|_{\mu.0}$, and G', the instantiated subformula $G[t/x]$. Thus, F' is equal to $F[G']^+_\mu$ by definition.

Let us prove the claim (a). If μ is positive, then $F|_\mu$ is of the form $\forall x\, G$. Then $F|_\mu$ implies G', and, by Corollary 2.2.20, F implies F'. If μ is negative, then $F|_\mu$ is of the form $\exists x\, G$. Then $F|_\mu$ follows from G', and, again by Corollary 2.2.20, F implies F'. The claim (b) follows from the same argument and Theorem 2.2.21.

As for (c), note that the antecedent image $\langle\!| W |\!\rangle^F_{\mu.0.\eta}$ is equal to $\langle\!| \forall x\, \langle\!| W |\!\rangle^G_\eta |\!\rangle^F_\mu$. The succedent $\langle\!| W[t/x] |\!\rangle^{F'}_{\mu.\eta}$ is equal to $\langle\!| \langle\!| W[t/x] |\!\rangle^{F'|_\mu}_\eta |\!\rangle^{F'}_\mu$. Since local substitution changes subformulas only at μ and to the right of μ, and since $F'|_\mu$ is G', the last image is equal to $\langle\!| \langle\!| W[t/x] |\!\rangle^{G'}_\eta |\!\rangle^F_\mu$. Hence, it is sufficient to show that $\forall x\, \langle\!| W |\!\rangle^G_\eta$ implies $\langle\!| W[t/x] |\!\rangle^{G'}_\eta$. But the latter formula is equal to $(\langle\!| W |\!\rangle^G_\eta)[t/x]$, and the claim is proved. □

Now let σ be a substitution $[t_1/x_1, t_2/x_2, \ldots, t_n/x_n]$ such that every variable x_1, x_2, …, x_n is instantiable at π in F. We can assume without loss of generality that the variables x_1, x_2, …, x_n do not occur in the terms-substitutes t_1, t_2, …, t_n, and that the corresponding positions of eliminable quantifiers μ_1, μ_2, …, μ_n are in the order of shortening, so that every μ_i is a subposition of μ_{i+1}. The result of *local substitution* $F[\sigma]^+_\pi$ is then defined by the equation:

$$F[\sigma]^+_\pi = (\ldots(F[[t_1/x_1]]^+_{\mu_1.0})[[t_2/x_2]]^+_{\mu_2.0} \cdots)[[t_n/x_n]]^+_{\mu_n.0}$$

The definition would be more clear if we applied all the atomic substitutions $[t_i/x_i]$ at π and not at $\mu_i.0$ (i.e. immediately under the corresponding quantifier). However, we cannot do so, because after the first atomic substitution π may be not a valid position anymore. On the other hand, positions $\mu_i.0$ are not affected by local substitutions at lower positions.

Local substitution is a chain of atomic local substitutions, which, by Lemma 2.2.22, are well-behaving inequivalent atomic transformations. So local substitution is a valid tranformation, too, satisfying the above conditions (i)–(iv). In particular, whenever a subformula of F gets instantiated by local substitution, its local properties are instantiated with it.

2.3 Notion of correctness

2.3.1 Correctness of a formula

In what follows, a ForTheL section $\mathbb{A}$ will be considered as a triple:

$$(\mathrm{T}\, H\, [\Lambda])$$

where T denotes the section kind, H is the formula image of $\mathbb{A}$, (omitted if none), and Λ is the sequence of sections in the body of $\mathbb{A}$ (omitted if none). The kind of $\mathbb{A}$ can be one of the following: `defn` for a definition, `sign` for a signature extension, `axiom` for an axiom, `prop` for a proposition, `case` for a case section, `assume` for an assumption, `select` for a selection, `affirm` for an affirmation concluding a proposition or occurring inside a proof, `posit` for an affirmation concluding an axiom, definition, or signature extension.

A formula F is *logically correct* in view of a sequence of sections Γ, denoted $\Gamma \vdash F$, whenever F can be deduced in the classical first-order predicate calculus from the formula images of sections from Γ. Generally, whenever a section is mentioned where a formula is expected, the formula image of that section is considered.

To define ontological correctness, we need some preparations. Let τ be a position of a term or an atomic formula in F. Let $\mathbb{D}$ be a definition or a signature extension section with the body $\mathbb{B}_1, \ldots, \mathbb{B}_n, \mathbb{A}$. Recall that $\mathbb{B}_i$ are assumptions, and $\mathbb{A}$ is an affirmation of the corresponding kind. Let E be the head term or atomic formula in $\mathbb{A}$ (Section 1.3.6). We say that $\mathbb{D}$ is *applicable* at τ in F in view of Γ, whenever $F|_\tau = E\sigma$ and $\Gamma \vdash \langle\!\langle (|\mathbb{B}_1| \wedge \cdots \wedge |\mathbb{B}_n|)\sigma \rangle\!\rangle^F_\tau$.

A formula F is *ontologically correct* in view of Γ, denoted $\Gamma \blacktriangleright F$, if and only if every occurrence $F|_\tau$ of the form $f(s_1, \ldots, s_n)$, $P(s_1, \ldots, s_n)$, or $s \mathrel{\varepsilon} N(s_1, \ldots, s_n)$ has an applicable definition or signature extension in Γ.

2.3.2 Calculus of Text Correctness

Correctness of a ForTheL text is deduced from logical and ontological correctness of particular formulas in view of appropriate sets of premises according to the Calculus of Text Correctness, or **CTC**.

In **CTC**, we infer sequents of the form $\Gamma \triangleright_G \Delta$. Only those sequents are allowed where $\mathcal{DV}_\Gamma(G) = \varnothing$ (i.e. $\mathcal{FV}(G) \subseteq \mathcal{FV}(\Gamma)$), and neither Γ nor G contain occurrences of $\mathfrak{T}$.

In such a sequent, Δ is a sequence of sections whose correctness is being verified and Γ is a sequence of sections which logically precede Δ. The formula G is a *current thesis*: a formula which we deduce from Γ with the help of auxiliary reasoning in Δ (note the first inference rule of **CTC**).

Note that verification of a sequence Δ consists in counter-applying the rules of **CTC**, reducing the sequent $\Gamma \triangleright_G \Delta$ to $\vdash$- and $\blacktriangleright$-premises which are to be checked directly.

A ForTheL text Γ is said to be *correct* whenever $\triangleright_\top \Gamma$ can be inferred in **CTC**. We say that Γ is *logically correct*, whenever we can infer $\triangleright_\top \Gamma$ assuming the additional axiom $\Gamma \blacktriangleright F$ for arbitrary Γ and F. Similarly, Γ is *ontologically correct* whenever we can infer $\triangleright_\top \Gamma$ assuming the additional axiom $\Gamma \vdash F$ for arbitrary Γ and F.

The inference rules of **CTC** are given in Figure 2.1. Let us explain the new notation and some possible obscurities appearing there.

The premise $(\Gamma \blacktriangleright F)^*$ reflects a special proviso we have to make when checking the ontological correctness of a definition or signature extension: namely, the main term or atomic formula in them need not have an applicable definition or signature extension in the preceding text,

$$\frac{\Gamma \vdash G}{\Gamma \ \triangleright_G} \qquad \frac{(\Gamma \blacktriangleright F)^* \qquad \Gamma,\, (\texttt{posit}\ F) \triangleright_\top \Delta}{\Gamma \ \triangleright_\top \ (\texttt{posit}\ F),\, \Delta}$$

$$\frac{\Gamma \blacktriangleright F \qquad \vec{x} = \mathcal{DV}_\Gamma(F) \qquad \Gamma \vdash \forall \vec{x}\, (F \supset G') \supset G \qquad \Gamma,\, (\texttt{assume}\ F) \triangleright_{G'} \Delta}{\Gamma \ \triangleright_G \ (\texttt{assume}\ \Theta_G(F)),\, \Delta}$$

$$\frac{\Gamma \blacktriangleright F \qquad \mathcal{DV}_\Gamma(F) = \varnothing \qquad \Gamma \triangleright_F \Lambda \qquad \Gamma \vdash (F \wedge G') \supset G \qquad \Gamma,\, (\texttt{affirm}\ F\ [\Lambda]) \triangleright_{G'} \Delta}{\Gamma \ \triangleright_G \ (\texttt{affirm}\ \Theta_G(F)\ [\Lambda]),\, \Delta}$$

$$\frac{\Gamma \blacktriangleright F \qquad \vec{x} = \mathcal{DV}_\Gamma(F) \qquad \Gamma \triangleright_{\exists \vec{x} F} \Lambda \qquad \Gamma \vdash \exists \vec{x}\, (F \wedge G') \supset G \qquad \Gamma, (\texttt{select}\ F\ [\Lambda]) \triangleright_{G'} \Delta}{\Gamma \ \triangleright_G \ (\texttt{select}\ \Theta_G(F)\ [\Lambda]),\, \Delta}$$

$$\frac{\Gamma \blacktriangleright F \qquad \mathcal{DV}_\Gamma(F) = \varnothing \qquad \Gamma, (\texttt{assume}\ F) \triangleright_G \Lambda \qquad \Gamma,\, (\texttt{case}\ (F \supset G)\ [\Lambda]) \triangleright_{G \vee F} \Delta}{\Gamma \ \triangleright_G \ (\texttt{case}\ (F \supset \mathfrak{T})\ [\Lambda]),\, \Delta}$$

$$\frac{\Gamma \triangleright_{\mathrm{IT}_t^{\prec}(G)} \Delta}{\Gamma \ \triangleright_G \ \Delta} \qquad \frac{\mathcal{DV}_{\Gamma,\Lambda}(\mathrm{IH}_t^{\prec}(G)) = \varnothing \qquad \Gamma \triangleright_{\mathrm{IT}_t^{\prec}(G)} \Lambda,\, (\texttt{assume}\ \mathrm{IH}_t^{\prec}(G)),\, \Delta}{\Gamma \ \triangleright_G \ \Lambda, \Delta}$$

$$\frac{\Gamma \triangleright_\top \Lambda \qquad \Gamma, (\texttt{prop}\ |\Lambda|\ [\Lambda]) \triangleright_\top \Delta}{\Gamma \ \triangleright_\top \ (\texttt{prop}\ |\Lambda|\ [\Lambda]),\, \Delta} \qquad \frac{\Gamma \triangleright_\top \Lambda \qquad \Gamma, (\texttt{axiom}\ |\Lambda|\ [\Lambda]) \triangleright_\top \Delta}{\Gamma \ \triangleright_\top \ (\texttt{axiom}\ |\Lambda|\ [\Lambda]),\, \Delta}$$

$$\frac{\Gamma \triangleright_\top \Lambda \qquad \Gamma, (\texttt{defn}\ |\Lambda|\ [\Lambda]) \triangleright_\top \Delta}{\Gamma \ \triangleright_\top \ (\texttt{defn}\ |\Lambda|\ [\Lambda]),\, \Delta} \qquad \frac{\Gamma \triangleright_\top \Lambda \qquad \Gamma, (\texttt{sign}\ |\Lambda|\ [\Lambda]) \triangleright_\top \Delta}{\Gamma \ \triangleright_\top \ (\texttt{sign}\ |\Lambda|\ [\Lambda]),\, \Delta}$$

Figure 2.1: Calculus of Text Correctness **CTC**

exactly because that symbol is introduced in the current section. So, the premise $(\Gamma \blacktriangleright F)^*$ means that F is ontologically correct at every liable position, except for the one of the main term or atomic formula.

The expression $\Theta_G(F)$ denotes the formula F where some occurrences of G are replaced with $\mathfrak{T}$. There may be more than one $\Theta_G(F)$ for given F and G.

The expressions $\mathrm{IT}_t^{\prec}(G)$ and $\mathrm{IH}_t^{\prec}(G)$ stand for the *induction thesis* and *induction hypothesis*, respectively. They are defined as follows. For a given formula G of the form

$$\forall \vec{x}_1\, (H_1 \supset \forall \vec{x}_2\, (H_2 \supset \ldots \forall \vec{x}_n\, (H_n \supset F) \ldots))$$

an arbitrary term t, and a binary relation symbol $\prec$:

$$\begin{aligned}
\mathrm{IH}_t^{\prec}(G) &= \forall \vec{x}'_1\, (H_1\sigma \supset \forall \vec{x}'_2\, (H_2\sigma \supset \ldots \forall \vec{x}'_n\, (H_n\sigma \supset (t\sigma \prec t \supset F\sigma)) \ldots)) \\
\mathrm{IT}_t^{\prec}(G) &= \forall \vec{x}_1\, (H_1 \supset \forall \vec{x}_2\, (H_2 \supset \ldots \forall \vec{x}_n\, (H_n \supset (\mathrm{IH}_t^{\prec}(G) \supset F)) \ldots))
\end{aligned}$$

where σ is the renaming substitution $[\vec{x}'_1/\vec{x}_1, \vec{x}'_2/\vec{x}_2, \ldots, \vec{x}'_n/\vec{x}_n]$ and $\vec{x}'_1, \vec{x}'_2, \ldots, \vec{x}'_n$ are some fresh variables.

The induction thesis $\mathrm{IT}_t^{\prec}(G)$ is equivalent to the original thesis G on condition that $\prec$ denotes a well-founded ordering. Note that the well-foundness of $\prec$ cannot be finitely expressed in a first-order language and the calculus **CTC** takes it on trust. In other words, correctness of a ForTheL text is verified on assumption of the following axiom scheme of general induction: $\mathcal{I}nd = \forall(\mathrm{IT}_{\mathbf{t}}^{\prec}(\mathbf{G}) \supset \mathbf{G})$, where $\mathbf{G}$ and $\mathbf{t}$ are placeholders for a formula and a term, respectively.

Note that $\mathrm{IT}_t^{\prec}(G)$ is equivalent to G if $\prec$ is the always false relation. Therefore the extension of first-order logic with the symbol $\prec$ and the axiom scheme $\mathcal{I}nd$ is conservative.

The first induction handling inference rule of **CTC** says that by proving the induction thesis, we automatically prove the initial one. In counter-application, it means that the verifier has the right to substitute the appropriate induction thesis for the initial thesis, when verifying a proof by induction. The second induction handling rule says additionally that the induction hypothesis need not to be put explicitly in the proof, but can be silently inserted there by the verifier. However, the induction hypothesis can not appear in the proof before all the free variables in it are declared.

Case section handling is another rule where an implicit logical predecessor, namely, the case hypothesis, is added by the verifier (in counter-application). Note that the Θ operation is not applied to a case hypothesis in the conclusion of the rule. That means that the word `thesis` can not appear in a ForTheL case hypothesis sentence, or it will not be verified.

2.3.3 Thesis reduction in CTC

In the rules of **CTC** for assumptions, selections, and affirmations, we see $\vdash$-premises that relate a current thesis G to a new thesis G' (by "new", we mean that G' is used as the thesis for subsequent ForTheL proof sequence). Such a transformation of thesis reflects our perception of a proof development when a complex formula is being demonstrated.

For instance, whenever we want to prove a conjunction $F \wedge G$ and succeed to derive one part of it, say F, and write down the affirmation of F in the text, then the thesis can be reduced just to G. Furthermore, if we prove a universal statement about sets $\forall x\,(x\,\varepsilon\,\mathrm{Set} \supset F)$ then we can begin by declaring x a set in an assumption, thus reducing the thesis to F.

When we see such a connection between the current thesis and a sentence under consideration, we call that sentence *motivated.* Motivated affirmations, selections, assumptions allow to reduce the thesis to a new formula which would be probably simpler to prove. Sometimes, the connection is evident from the syntax: as when the thesis is an implication and the assumption is the antecedent formula. Sometimes, the connection depends on several reasoning steps: for example, if variables S, T are known as sets and the current thesis is $S\,\varepsilon\,\mathrm{Subset}(T)$, then the assumption "`let x be an element of S`" is motivated and reduces the current thesis to $x\,\varepsilon\,\mathrm{Element}(T)$.

There is nothing special in non-motivated affirmations or selections. Whenever we meet such a sentence, we simply do not change the thesis. On the contrary, in a well-written mathematical proof, assumptions should be always motivated, i.e. "suggested" by a current thesis. A non-motivated assumption is an unjustified narrowing of the search space. Also, it may happen that our reasoning capabilities are too weak to discover the justification.

Now, how can we infer $\Gamma \rhd_G (\texttt{assume}\ F), \Delta$, if F is in no visible relation with G? Though the calculus **CTC** admits various solutions, in our system (see Section **??**), the choice of the new thesis is guided by the form of Δ. Whenever the formula images of sentences in Δ contain occurrences of $\mathfrak{T}$, (e.g. when there are case sections), we suppose that the proof of G continues under the non-motivated assumption and leave the thesis unchanged:

$$\frac{\Gamma \blacktriangleright F \qquad \vec{x} = \mathcal{DV}_\Gamma(F) \qquad \Gamma \vdash \forall \vec{x}\,(F \supset G) \supset G \qquad \Gamma,\ (\texttt{assume}\ F) \rhd_G \Delta}{\Gamma\ \rhd_G\ (\texttt{assume}\ \Theta_G(F)),\ \Delta}$$

The premise $\Gamma \vdash \forall \vec{x}\,(F \supset G) \supset G$ is nontrivial: its is equivalent to the disjunction $G \vee (\exists \vec{x}\, F)$. Recall that the free variables of G are all declared in Γ and thus cannot be among $\vec{x}$.

If $\mathfrak{T}$ does not occur in the formula images in Δ, we suppose that the rest of the proof is a sort of independent argument which should be considered by itself. Therefore we take for the new thesis the image of the whole rest of the proof sequence. The inference is as follows:

$$\frac{\Gamma \blacktriangleright F \qquad \vec{x} = \mathcal{DV}_\Gamma(F) \qquad \Gamma \vdash \forall \vec{x}\,(F \supset |\Delta|_\Gamma) \supset G \qquad \Gamma, (\texttt{assume}\ F) \rhd_{|\Delta|_\Gamma} \Delta}{\Gamma\ \rhd_G\ (\texttt{assume}\ \Theta_G(F)),\ \Delta}$$

Here, $|\Delta|_\Gamma$ is the formula image of the rest of the proof, calculated as defined in Section 1.5.5. Note that the new variables in Δ, not known from Γ or F, are all bound in $|\Delta|_\Gamma$.

Any assumption in Δ is an antecedent in the new thesis $|\Delta|_\Gamma$ and therefore will be considered as motivated. Any affirmation or selection in Δ will reduce the thesis, too, so that at the end of Δ the thesis will be simply $\top$. The premise $\Gamma \vdash \forall \vec{x}\,(F \supset |\Delta|_\Gamma) \supset G$ finishes the demonstration by deducing G from the formula image of the proof sequence $(\texttt{assume}\ F)$, Δ.

As one may see, the rules of **CTC** do not require current theses to be ontologically correct. Only at the beginning of a proof, when the initial thesis is set to the statement being proved, its ontological correctness is ensured. Since it is initial theses that we are interested in (see the next section), we do not need to ask more from the calculus. Nevertheless, it is highly desirable to maintain the ontological correctness of theses generated by the verifier, so that we never find ourselves in the middle (still worse, at the end) of a proof with a goal of questionable meaning.

Note that carrying the thesis unchanged by an unmotivated sentence preserves its ontological correctness: $\Gamma \blacktriangleright G$ obviously implies $\Gamma, \mathbb{A} \blacktriangleright G$. Furthermore, taking as the thesis the formula image of the rest of the proof is safe, too, as shown by the following simple lemma:

Lemma 2.3.1. *Let $\Delta = \mathbb{A}_1, \mathbb{A}_2, \ldots, \mathbb{A}_n$ be a sequence of sentences whose formula images do not contain occurrences of $\mathfrak{T}$. Let Γ be a sequence of sections such that $\Gamma \blacktriangleright \mathbb{A}_1$ and $\Gamma, \mathbb{A}_1 \blacktriangleright \mathbb{A}_2$, and so on, down to $\Gamma, \mathbb{A}_1, \mathbb{A}_2, \ldots, \mathbb{A}_{n-1} \blacktriangleright \mathbb{A}_n$. Then $\Gamma \blacktriangleright |\Delta|_\Gamma$.*

Proof. Let D stand for $|\Delta|_\Gamma$. Any local image $\langle\!| W |\!\rangle_\pi^D$ that we must deduce from Γ in order to prove $\Gamma \blacktriangleright D$, is of the form $\forall \vec{x}_1 (|\mathbb{A}_1| \supset \forall \vec{x}_2 (|\mathbb{A}_2| \supset \ldots \forall \vec{x}_{i-1} (|\mathbb{A}_{i-1}| \supset \forall \vec{x}_i \langle\!| W |\!\rangle_\tau^{|\mathbb{A}_i|}) \ldots))$, where $\vec{x}_1 = \mathcal{DV}_\Gamma(\mathbb{A}_1)$ and $\vec{x}_2 = \mathcal{DV}_{\Gamma,\mathbb{A}_1}(\mathbb{A}_2)$, and so on. Since the sets $\vec{x}_1, \ldots, \vec{x}_i$ are pairwise disjoint and also disjoint from $\mathcal{FV}(\Gamma)$, the problem is equivalent to $\Gamma, |\mathbb{A}_1|, |\mathbb{A}_2|, \ldots, |\mathbb{A}_{i-1}| \vdash \langle\!| W |\!\rangle_\tau^{|\mathbb{A}_i|}$. This is given to us by $\Gamma, \mathbb{A}_1, \mathbb{A}_2, \ldots, \mathbb{A}_{i-1} \blacktriangleright \mathbb{A}_i$. □

As will be shown in Section **??**, other rules of thesis reduction implemented in our system, are all based on valid formula transformations, fulfilling the four conditions from Section 2.2.3, and thus maintaining the ontological correctness of the current thesis.

Altogether, the following theorem can be seen as the statement of soundness of **CTC**:

Theorem 2.3.2. *Let Γ and Δ be sequences of ForTheL sections and G, an arbitrary formula. If $\Gamma \rhd_G \Delta$ can be inferred in* **CTC** *then $\mathcal{I}nd, \Gamma \vdash G$.*

Proof. The claim can be proved by induction on the number of steps in the inference of $\Gamma \rhd_G \Delta$. Let us consider the last inference step. If it is made by a rule with $\rhd_\top$ in conclusion, then G is $\top$, and the claim is trivial. Otherwise, we have seven cases to consider. We will denote the cases by the form of the conclusion of the corresponding inference rule.

Case $\Gamma \rhd_G$. The premise of this rule is $\Gamma \vdash G$, hence the claim.

Case $\Gamma \rhd_G (\texttt{assume}\ \Theta_G(F)), \Delta$. By the premises of the rule, we have $\Gamma \vdash \forall \vec{x}\,(F \supset G') \supset G$ and $\Gamma, (\texttt{assume}\ F) \rhd_{G'} \Delta$, where $\vec{x} = \mathcal{DV}_\Gamma(F) = \mathcal{FV}(F) \backslash \mathcal{FV}(\Gamma)$. By the induction hypothesis, the latter implies $\mathcal{I}nd, \Gamma, F \vdash G'$. Also, $\mathcal{FV}(G) \subseteq \mathcal{FV}(\Gamma)$ and $\mathcal{FV}(G') \subseteq \mathcal{FV}(\Gamma) \cup \mathcal{FV}(F)$. Therefore, $\mathcal{FV}(F \supset G') \backslash \mathcal{FV}(\Gamma) = \vec{x}$. Hence $\mathcal{I}nd, \Gamma \vdash \forall \vec{x}\,(F \supset G')$ and we have the claim.

Case $\Gamma \rhd_G (\texttt{affirm}\ \Theta_G(F)\ [\Lambda]), \Delta$. This is subsumed by the next case.

Case $\Gamma \rhd_G (\texttt{select}\ \Theta_G(F)\ [\Lambda]), \Delta$. We have $\Gamma \rhd_{\exists \vec{x} F} \Lambda$ and $\Gamma \vdash \exists \vec{x}\,(F \wedge G') \supset G$, and $\Gamma, (\texttt{select}\ F\ [\Lambda]) \rhd_{G'} \Delta$, where $\vec{x} = \mathcal{FV}(F) \backslash \mathcal{FV}(\Gamma)$. By the induction hypothesis, we have $\mathcal{I}nd, \Gamma \vdash \exists \vec{x}\, F$ and $\mathcal{I}nd, \Gamma, F \vdash G'$. Also, $\mathcal{FV}(G) \subseteq \mathcal{FV}(\Gamma)$ and $\mathcal{FV}(G') \subseteq \mathcal{FV}(\Gamma) \cup \mathcal{FV}(F)$. Hence $\mathcal{FV}(F \wedge G') \backslash \mathcal{FV}(\Gamma) = \vec{x}$ and $\mathcal{I}nd, \Gamma \vdash \forall \vec{x}\,(F \supset G')$. This implies $\mathcal{I}nd, \Gamma \vdash \exists \vec{x}\,(F \wedge G')$ and we have the claim.

Case $\Gamma \rhd_G (\texttt{case}\ (F \supset \mathfrak{T})\ [\Lambda]), \Delta$. From the premises, we obtain $\Gamma, (\texttt{assume}\ F) \rhd_G \Lambda$ and $\Gamma, (\texttt{case}\ (F \supset G)\ [\Lambda]) \rhd_{G \vee F} \Delta$. Also, $\mathcal{DV}_\Gamma(F) = \varnothing$ which means that $\mathcal{FV}(F), \mathcal{FV}(G) \subseteq \mathcal{FV}(\Gamma)$. By the induction hypothesis, we have $\mathcal{I}nd, \Gamma, F \vdash G$ and $\mathcal{I}nd, \Gamma, (F \supset G) \vdash (G \vee F)$. The former gives $\mathcal{I}nd, \Gamma \vdash (F \supset G)$. The latter gives $\mathcal{I}nd, \Gamma, (F \supset G) \vdash G$ and we have the claim.

Cases $\Gamma \rhd_G \Delta$ and $\Gamma \rhd_G \Delta, \Lambda$ (induction handling rules). By the premise of the rule and the induction hypothesis, we have $\mathcal{I}nd, \Gamma \vdash \mathrm{IT}_t^{\prec}(G)$. By definition of $\mathcal{I}nd$, we have the claim. □

2.4 Verification example

Let us consider an example of a well-formed ForTheL text. We are going to check its correctness according to the correctness calculus from Figure 2.1.

```
[number/numbers]
```

```
Signature Nat.                                          # 0
  A natural number is a notion.                         # 0.0
Signature Zer.                                          # 1
  0 is a natural number.                                # 1.0
Signature Suc.                                          # 2
  Let i be a natural number.                            # 2.0
  succ i is a natural number.                           # 2.1
Signature Add.                                          # 3
  Let i,j be natural numbers.                           # 3.0
  i + j is a natural number.                            # 3.1
Signature Ord.                                          # 4
  Let i,j be natural numbers.                           # 4.0
  i -<- j is an atom.                                   # 4.1

Axiom ZerSuc.                                           # 5
  For any natural number i if i != 0 then               # 5.0
  there exists a natural number j such that succ j = i.
Axiom AddZer.                                           # 6
  For any natural number i (i + 0 = i).                 # 6.0
Axiom AddSuc.                                           # 7
  For all natural numbers i,j (i + succ j = succ (i+j)). # 7.0
Axiom OrdSuc.                                           # 8
  For any natural number i (i -<- succ i).              # 8.0

Lemma ZerAdd.                                           # 9
  For any natural number i (0 + i = i).                 # 9.0
Proof by induction.
  Let i be a natural number.                            # 9.0.0

  Case i = 0.                                           # 9.0.1
  Obvious.

  Case i != 0.                                          # 9.0.2
    Take a natural number j such that succ j = i.       # 9.0.2.0
    We have j -<- i.                                    # 9.0.2.1
    Hence 0 + j = j.                                    # 9.0.2.2
    Then we have the thesis.                            # 9.0.2.3
  end.
qed.
```

First of all, note the numbers in the comments. Each number denotes a “position” of a particular ForTheL section (we put the term in quotes, since we have not extended the notion of position on texts). For example, 9.0 is the position of the main affirmation in the lemma `ZerAdd`, 9.0.0 is the position of the starting assumption in the proof, 9.0.1 and 9.0.2 point at the case sections. The relation of logical precedence on that positions (as defined at the beginning of Section 2.2.1) corresponds to the one on ForTheL sections (as defined in Section 1.5.4).

Let us reconsider this text substituting the formula images instead of ForTheL sentences:

```
sign Nat
   posit  $\forall x\,(x \mathrel{\varepsilon} \mathrm{NatNum} \supset \top)$
sign Zer
   posit  $\forall x\,(x \approx 0 \supset x \mathrel{\varepsilon} \mathrm{NatNum})$
sign Suc
   assume  $i \mathrel{\varepsilon} \mathrm{NatNum}$
   posit  $\forall x\,(x \approx \mathrm{succ}\ i \supset x \mathrel{\varepsilon} \mathrm{NatNum})$
sign Add
```

```
    assume  $i \mathrel{\varepsilon} \mathrm{NatNum} \wedge j \mathrel{\varepsilon} \mathrm{NatNum}$
    posit  $\forall x\,(x \approx i + j \supset x \mathrel{\varepsilon} \mathrm{NatNum})$
  sign Ord
    assume  $i \mathrel{\varepsilon} \mathrm{NatNum} \wedge j \mathrel{\varepsilon} \mathrm{NatNum}$
    posit  $i \prec j \supset \top$

  axiom ZerSuc
    posit  $\forall i\,(i \mathrel{\varepsilon} \mathrm{NatNum} \supset i \neq 0 \supset \exists j\,(j \mathrel{\varepsilon} \mathrm{NatNum} \wedge \mathrm{succ}\ j \approx i))$
  axiom AddZer
    posit  $\forall i\,(i \mathrel{\varepsilon} \mathrm{NatNum} \supset i + 0 \approx i)$
  axiom AddSuc
    posit  $\forall i\,(i \mathrel{\varepsilon} \mathrm{NatNum} \supset \forall j\,(j \mathrel{\varepsilon} \mathrm{NatNum} \supset i + (\mathrm{succ}\ j) \approx \mathrm{succ}\ (i + j)))$

  axiom OrdSuc
    posit  $\forall i\,(i \mathrel{\varepsilon} \mathrm{NatNum} \supset i \prec \mathrm{succ}\ i)$

  prop ZerAdd
    affirm  $\forall i\,(i \mathrel{\varepsilon} \mathrm{NatNum} \supset 0 + i \approx i)$
      assume  $i \mathrel{\varepsilon} \mathrm{NatNum}$
      case  $i = 0 \supset \mathfrak{T}$
      case  $i \neq 0 \supset \mathfrak{T}$
        select  $j \mathrel{\varepsilon} \mathrm{NatNum} \wedge (\mathrm{succ}\ j) \approx i$
        affirm  $j \prec i$
        affirm  $0 + j \approx j$
        affirm  $\mathfrak{T}$
```

Now, we are going to list the inference steps which prove correctness of our text. In order to fit into the page width we will write position numbers in parentheses in place of the corresponding sections. We proceed in a bottom-top manner, from the desired conclusion to the axioms. Top level goes first.

$$\frac{\rhd_\top (0),\dots,(9)}{\rhd_\top (0.0) \qquad \dfrac{(0) \rhd_\top (1),\dots,(9)}{(0) \rhd_\top (1.0) \qquad \dfrac{(0,1) \rhd_\top (2),\dots,(9)}{(0),(1) \rhd_\top (2.0),(2.1) \qquad (0),(1),(2) \rhd_\top (3),\dots,(9)}}}$$

$$\frac{(0),(1),(2) \rhd_\top (3),\dots,(9)}{(0),(1),(2) \rhd_\top (3.0),(3.1) \qquad \dfrac{(0),\dots,(3) \rhd_\top (4),\dots,(9)}{(0),\dots,(3) \rhd_\top (4.0),(4.1) \qquad (0),\dots,(4) \rhd_\top (5),\dots,(9)}}$$

$$\frac{(0),\dots,(4) \rhd_\top (5),\dots,(9)}{(0),\dots,(4) \rhd_\top (5.0) \qquad \dfrac{(0),\dots,(5) \rhd_\top (6),\dots,(9)}{(0),\dots,(5) \rhd_\top (6.0) \qquad (0),\dots,(6) \rhd_\top (7),(8),(9)}}$$

$$\frac{(0),\dots,(6) \rhd_\top (7),(8),(9)}{(0),\dots,(6) \rhd_\top (7.0) \qquad \dfrac{(0),\dots,(7) \rhd_\top (8),(9)}{(0),\dots,(7) \rhd_\top (8.0) \qquad \dfrac{(0),\dots,(8) \rhd_\top (9)}{(0),\dots,(8) \rhd_\top (9.0) \qquad \dfrac{(0),\dots,(9) \rhd_\top}{(0),\dots,(9) \vdash \top}}}}$$

Now, we inspect the bodies of the top-level sections. Note that the thesis $\top$ never needs to be reduced in the rules for assumptions, selections and affirmations, i.e. the new thesis (denoted

G' in Figure 2.1) will be also equal to $\top$.

$$\frac{\rhd_\top (0.0)}{\blacktriangleright \forall x\, (x \,\varepsilon\, \mathrm{NatNum} \supset \top) \qquad \dfrac{(0.0) \rhd_\top}{(0.0) \vdash \top}} \qquad\qquad \frac{(0) \rhd_\top (1.0)}{(0) \blacktriangleright 0 \,\varepsilon\, \mathrm{NatNum} \qquad \dfrac{(0), (1.0) \rhd_\top}{(0), (1.0) \vdash \top}}$$

$$\frac{(0), (1) \rhd_\top (2.0), (2.1)}{(0), (1) \blacktriangleright i \,\varepsilon\, \mathrm{NatNum} \qquad (0), (1), \vdash \forall i\, (i \,\varepsilon\, \mathrm{NatNum} \supset \top) \supset \top \qquad (0), (1), (2.0) \rhd_\top (2.1)}$$

$$\frac{(0), (1), (2.0) \rhd_\top (2.1)}{(0), (1), (2.0) \blacktriangleright (\mathrm{succ}\ i) \,\varepsilon\, \mathrm{NatNum} \qquad \dfrac{(0), (1), (2.0), (2.1) \rhd_\top}{(0), (1), (2.0), (2.1) \vdash \top}}$$

$$\frac{(0), (1), (2) \rhd_\top (3.0), (3.1)}{(0), (1), (2) \blacktriangleright i, j \,\varepsilon\, \mathrm{NatNum} \quad (0), (1), (2) \vdash \forall i\, (|(3.0)| \supset \top) \supset \top \quad (0), (1), (2), (3.0) \rhd_\top (3.1)}$$

$$\frac{(0), (1), (2), (3.0) \rhd_\top (3.1)}{(0), (1), (2), (3.0) \blacktriangleright (i + j) \,\varepsilon\, \mathrm{NatNum} \qquad \dfrac{(0), (1), (2), (3.0), (3.1) \rhd_\top}{(0), (1), (2), (3.0), (3.1) \vdash \top}}$$

$$\frac{(0), \ldots, (3) \rhd_\top (4.0), (4.1)}{(0), \ldots, (3) \blacktriangleright i, j \,\varepsilon\, \mathrm{NatNum} \quad (0), \ldots, (3) \vdash \forall i\, (|(4.0)| \supset \top) \supset \top \quad (0), \ldots, (3), (4.0) \rhd_\top (4.1)}$$

$$\frac{(0), \ldots, (3), (4.0) \rhd_\top (4.1)}{(0), \ldots, (3), (4.0) \blacktriangleright i \prec j \supset \top \qquad \dfrac{(0), \ldots, (3), (4.0), (4.1) \rhd_\top}{(0), \ldots, (3), (4.0), (4.1) \vdash \top}}$$

$$\frac{(0), \ldots, (i-1) \rhd_\top ((i).0)}{(0), \ldots, (i-1) \blacktriangleright |((i).0)| \qquad \dfrac{(0), \ldots, (i-1), ((i).0) \rhd_\top}{(0), \ldots, (i-1), ((i).0) \vdash \top}} \qquad (\text{for all } i \in \{5, 6, 7, 8\})$$

$$\frac{\Gamma \rhd_\top (9.0)}{\Gamma \blacktriangleright |(9.0)| \qquad \dfrac{\Gamma \rhd_{|(9.0)|} (9.0.0), (9.0.1), (9.0.2)}{\Gamma \rhd_G (9.0.0), \mathbb{A}, (9.0.1), (9.0.2)} \qquad \Gamma \vdash (|(9.0)| \wedge \top) \supset \top \qquad \dfrac{\Gamma, (9.0) \rhd_\top}{\Gamma, (9.0) \vdash \top}}$$

where

$$\begin{aligned} \Gamma &= (0), \ldots, (8) \\ \mathbb{A} &= (\texttt{assume}\ H) \\ |(9.0)| &= \forall i\, (i \,\varepsilon\, \mathrm{NatNum} \supset 0 + i \approx i) \\ G = \mathrm{IT}_i^{\prec}(|(9.0)|) &= \forall i\, (i \,\varepsilon\, \mathrm{NatNum} \supset (H \supset 0 + i \approx i)) \\ H = \mathrm{IH}_i^{\prec}(|(9.0)|) &= \forall i'\, (i' \,\varepsilon\, \mathrm{NatNum} \supset (i' \prec i \supset 0 + i' \approx i')) \end{aligned}$$

Pay attention to the last fragment of inference, where we apply the induction rule. Instead of proving the statement of the affirmation (9.0) as is, we descend into the proof with a weakened current thesis G having the additional induction hypothesis H.

We begin by inserting that induction hypothesis H into the proof. Note that the variable i which is free in H is declared in (9.0.0) and therefore known at the position of added hypothesis. Also note how the two assumptions reduce the current thesis from G to G' and then to G''.

$$\frac{\Gamma \rhd_G (9.0.0), \mathbb{A}, (9.0.1), (9.0.2)}{\Gamma \blacktriangleright i \,\varepsilon\, \mathrm{NatNum} \qquad \Gamma \vdash \forall i\, (i \,\varepsilon\, \mathrm{NatNum} \supset G') \supset G \qquad \Gamma, (9.0.0) \rhd_{G'} \mathbb{A}, (9.0.1), (9.0.2)}$$

$$\dfrac{\Gamma, (9.0.0) \rhd_{G'} \mathbb{A}, (9.0.1), (9.0.2)}{\Gamma, (9.0.0) \blacktriangleright H \qquad \Gamma, (9.0.0) \vdash (H \supset G'') \supset G' \qquad \Gamma, (9.0.0), \mathbb{A} \rhd_{G''} (9.0.1), (9.0.2)}$$

where

$$G' = (H \supset 0 + i \approx i) \qquad\qquad G'' = (0 + i \approx i)$$

The first case section is very short. Note how $\mathfrak{T}$ in the formula image of (9.0.1) is replaced with the actual thesis in $(9.0.1)'$:

$$\dfrac{\Gamma, (9.0.0), \mathbb{A} \rhd_{G''} (9.0.1), (9.0.2)}{\Gamma, (9.0.0), \mathbb{A} \blacktriangleright i \approx 0 \qquad \dfrac{\dfrac{\Gamma, (9.0.0), \mathbb{A}, \mathbb{C}_1 \rhd_{G''}}{\Gamma, (9.0.0), \mathbb{A}, \mathbb{C}_1 \vdash G''}}{\Gamma, i \,\varepsilon\, \mathrm{NatNum}, i \approx 0 \;\vdash\; 0 + i \approx i} \qquad \Gamma, (9.0.0), \mathbb{A}, (9.0.1)' \rhd_{G'''} (9.0.2)}$$

where

$$\mathbb{C}_1 = (\texttt{assume}\ i \approx 0) \qquad (9.0.1)' = (\texttt{case}\ (i \approx 0 \supset G'')) \qquad G''' = G'' \vee i \approx 0$$

The second case section is longer but no more complex:

$$\dfrac{\Delta_0 \rhd_{G'''} (\tau)}{\Delta_0 \blacktriangleright i \not\approx 0 \qquad \Delta_0, \mathbb{C}_2 \rhd_{G'''} (\tau.0), (\tau.1), (\tau.2), (\tau.3) \qquad \dfrac{\dfrac{\Delta_0, (\tau)' \;\rhd_{G''' \vee i \not\approx 0}}{\Delta_0, (\tau)' \;\vdash G''' \vee i \not\approx 0}}{\vdash G'' \vee i \approx 0 \vee i \not\approx 0}}$$

$$\dfrac{\Delta_1 \rhd_{G'''} (\tau.0), (\tau.1), (\tau.2), (\tau.3)}{\Delta_1 \blacktriangleright F_1 \qquad \dfrac{\Delta_1 \rhd_{\exists j F_1}}{\Delta_1 \vdash \exists j F_1} \qquad \Delta_1 \vdash \exists j\, (F_1 \wedge G''') \supset G''' \qquad \Delta_1, (\tau.0) \rhd_{G'''} (\tau.1), (\tau.2), (\tau.3)}$$

$$\dfrac{\Delta_2 \rhd_{G'''} (\tau.1), (\tau.2), (\tau.3)}{\Delta_2 \blacktriangleright j \prec i \qquad \dfrac{\Delta_2 \rhd_{j \prec i}}{\Delta_2 \vdash j \prec i} \qquad \Delta_2 \vdash (j \prec i \wedge G''') \supset G''' \qquad \Delta_2, (\tau.1) \rhd_{G'''} (\tau.2), (\tau.3)}$$

$$\dfrac{\Delta_3 \rhd_{G'''} (\tau.2), (\tau.3)}{\Delta_3 \blacktriangleright 0 + j \approx j \qquad \dfrac{\Delta_3 \rhd_{0+j \approx j}}{\Delta_3 \vdash 0 + j \approx j} \qquad \Delta_3 \vdash (0 + j \approx j \wedge G''') \supset G''' \qquad \Delta_3, (\tau.2) \rhd_{G'''} (\tau.3)}$$

$$\dfrac{\Delta_4 \rhd_{G'''} (\tau.3)}{\Delta_4 \blacktriangleright G''' \qquad \dfrac{\dfrac{\Delta_4 \rhd_{G'''}}{\Delta_4 \vdash G'''}}{\Delta_4 \vdash 0 + i \approx i} \qquad \Delta_4 \vdash (G''' \wedge \top) \supset G''' \qquad \dfrac{\Delta_4, (\texttt{affirm}\ G'''\ [\ ]) \rhd_{\top}}{\Delta_4, (\texttt{affirm}\ G'''\ [\ ]) \vdash \top}}$$

where

$$\begin{aligned} \tau &= 9.0.2 & \Delta_0 &= \Gamma, (9.0.0), \mathbb{A}, (9.0.1)' \\ \mathbb{C}_2 &= (\texttt{assume}\ (i \not\approx 0)) & \Delta_1 &= \Delta_0, \mathbb{C}_2 \\ \Lambda &= (\tau.0), (\tau.1), (\tau.2), (\tau.3) & \Delta_2 &= \Delta_1, (\tau.0) \\ (\tau)' &= (\texttt{case}\ (i \not\approx 0 \supset G''')\ [\Lambda]) & \Delta_3 &= \Delta_2, (\tau.1) \\ F_1 &= j \,\varepsilon\, \mathrm{NatNum} \wedge \mathrm{succ}\ j \approx i & \Delta_4 &= \Delta_3, (\tau.2) \end{aligned}$$

A few comments should be made here. First, note the right-hand branch in the first inference fragment, where the goal $G'' \vee i \approx 0 \vee i \not\approx 0$ is proved. According to the rules of our calculus, each additional case section weakens the current thesis by putting it into a disjunction with the case's hypothesis. At the end of case analysis we have to prove the formula $G \vee H_1 \vee \cdots \vee H_n$, where G is the original thesis and $H_1, \ldots, H_n$ are explored cases. Yet, it is a good style to make case analyses exhaustive so that just the disjunction $H_1 \vee \cdots \vee H_n$ would hold at the end.

Second, a selection sentence is valid whenever we can prove the existence of named objects, i.e. the nonemptyness of the classes corresponding to the listed notions. While in the ForTheL text in question the selection (9.0.2.0) does not change the current thesis, it may happen when the current thesis is a statement of existence.

Third, pay attention that the affirmation (9.0.2.2) is a direct consequence of the induction hypothesis H and the previous affirmation (9.0.2.1). If the assumption $\mathbb{A}$ (whose formula image is H) was not inserted in the proof, the affirmation (9.0.2.2) could not be proved. However, one can write a proof where no sentence requires the induction hypothesis in order to be verified. For example, the whole proof of the affirmation (9.0) could be simply omitted. Then the system would try to prove just the induction thesis G, which is not difficult and does not require any induction reasoning capabilities.

Fourth, let us consider the last inference fragment. The formula image of the affirmation (9.0.2.3) is just the atomic formula $\mathfrak{T}$, which stands for the current thesis, G'''. Once having this affirmation proved, we have no pending obligations so that the new thesis is simply $\top$. Recall that G''' is the formula $0 + i \approx i \vee i \approx 0$, that is, we must either prove the initial thesis (G'') or reduce the task to the previous case.

Now, assuming the validity of all first-order leaves ($\vdash$-sequents) in our derivation, we have demonstrated the *logical correctness* of the ForTheL-text under consideration. In order to prove a text fully correct, we must also check the ontological correctness of its sentences ($\blacktriangleright$-sequents). In our example it is mostly obvious task. Let us consider the sequent $\Delta_1 \blacktriangleright F_1$, which is equivalent to

$$(0), \ldots, (8), (9.0.0), \mathbb{A}, (9.0.1)', \mathbb{C}_2 \blacktriangleright j \,\varepsilon\, \mathrm{NatNum} \wedge \mathrm{succ}\; j \approx i$$

We have two occurrences of signature symbols in F_1: the notion symbol NatNum of zero arity and the unary function symbol succ (recall that the equality symbol is considered logical in our language). For each of those there is a unique signature extension section to inspect: (0) for NatNum and (2) for succ. The first occurrence is obviously ontologically correct since there are no guards in (0) to instantiate and prove at the position of $j \,\varepsilon\, \mathrm{NatNum}$. In order to verify the ontological correctness of succ j we must prove:

$$(0), \ldots, (8), (9.0.0), \mathbb{A}, (9.0.1)', \mathbb{C}_2 \vdash \langle\!\langle j \,\varepsilon\, \mathrm{NatNum} \rangle\!\rangle^{F_1}_{1.0}$$

By definition of directed local images, $\langle\!\langle j \,\varepsilon\, \mathrm{NatNum} \rangle\!\rangle^{F_1}_{1.0} = j \,\varepsilon\, \mathrm{NatNum} \supset j \,\varepsilon\, \mathrm{NatNum}$. Thus, the ontological correctness of F_1 is proved.

Index

Bibliography

[1] Corella, F.: *What holds in a context?* Journal of Automated Reasoning, 10(2):79–93, 1993.

[2] Fuchs, N.E., U. Schwertel, and R. Schwitter: *Attempto Controlled English — not just another logic specification language.* In Flener, P. (ed.): *Logic-Based Program Synthesis and Transformation: 8th International Workshop, LOPSTR'98*, vol. 1559 of *Lecture Notes in Computer Science*, pp. 1–20. Springer, 1999.

[3] Grundy, J.: *Transformational hierarchical reasoning.* The Computer Journal, 39(4):291–302, 1996.

[4] Kamareddine, F. and R.P. Nederpelt: *A refinement of de Bruijn's formal language of mathematics.* Journal of Logic, Language and Information, 13(3):287–340, 2004.

[5] Kleene, S.C.: *Introduction to Metamathematics.* Van Nostrand, 1952.

[6] Monk, L.G.: *Inference rules using local contexts.* Journal of Automated Reasoning, 4(4):445–462, 1988.

[7] Nederpelt, R.P., J.H. Geuvers, and R.C. de Vrijer (eds.): *Selected Papers on Automath*, vol. 133 of *Studies in Logic and the Foundations of Mathematics.* North-Holland, 1994.

[8] Robinson, P.J. and J. Staples: *Formalising the hierarchical structure of practical mathematical reasoning.* Journal of Logic and Computation, 3(1):47–61, 1993.

[9] Sowa, J.F.: *Common Logic Controlled English (unpublished draft).* Disponible sur http://www.jfsowa.com/clce/specs.htm, 2004.

[10] Trybulec, A. and H. Blair: *Computer assisted reasoning with Mizar.* In Joshi, A.K. (ed.): *Proc. 9th International Joint Conference on Artificial Intelligence, IJCAI 1985*, pp. 26–28. Morgan-Kaufmann, 1985.

[11] Weidenbach, C., R. Schmidt, T. Hillenbrand, R. Rusev, and D. Topic: *SPASS version 3.0.* In Pfenning, F. (ed.): *Automated Deduction: 21st International Conference, CADE-21*, vol. 4603 of *Lecture Notes in Computer Science*, pp. 514–520. Springer, 2007.

[12] Wenzel, M.: *Isabelle/Isar — a generic framework for human-readable proof documents.* In Matuszewski, R. and A. Zalewska (eds.): *From Insight to Proof — Festschrift in Honour of Andrzej Trybulec*, vol. 10(23) of *Studies in Logic, Grammar, and Rhetoric*, pp. 277–298. University of Białystok, 2007.

[13] Wiedijk, F. (ed.): *The Seventeen Provers of the World*, vol. 3600 of *Lecture Notes in Computer Science.* Springer, 2006.

A Tarski-Knaster fixed point theorem

In this appendix we give a verified ForTheL formalization of the Tarski-Knaster theorem about fixed points of a monotone function on a complete lattice.

The text below illustrates certain extensions of ForTheL that are not described in the chapter 1: the groups of abbreviated tokens (`[set/-s]` is equivalent to `[set/sets]`), the predeclared variables (that are ascribed to certain notions by default), the chains of symbolic relations (e.g. `x <= y <= z`), and the set comprehensions.

```
[set/-s] [subset/-s] [element/-s] [belong/-s]

Signature SetSort. A set is a notion.
Signature ElmSort. An element is a notion.

Let S,T denote sets.
Let x,y,z,u,v,w denote elements.

Signature EOfElem. An element of S is an element.

Let x << S denote (x is an element of S).
Let x belongs to S denote (x is an element of S).

Definition DefEmpty.  S is empty iff S has no elements.

Definition DefSub.
  A subset of S is a set T such that every (x << T) belongs to S.

Let S [= T denote S is a subset of T.

Signature LessRel.  x <= y is an atom.

Axiom ARefl. x <= x.
Axiom ASymm. x <= y <= x => x = y.
Axiom Trans. x <= y <= z => x <= z.

[bound/-s] [supremum/-s] [infimum/-s] [lattice/-s]

Definition DefLB.   Let S be a subset of T.
  A lower bound of S in T is an element u of T
    such that for every (x << S) u <= x.

Definition DefUB.   Let S be a subset of T.
  An upper bound of S in T is an element u of T
    such that for every (x << S) x <= u.

Definition DefInf.  Let S be a subset of T.
  An infimum of S in T is an element u of T
```

```
    such that u is a lower bound of S in T and
      for every lower bound v of S in T we have v <= u.

Definition DefSup.  Let S be a subset of T.
  A supremum of S in T is an element u of T
    such that u is a upper bound of S in T and
      for every upper bound v of S in T we have u <= v.

Lemma SupUn.  Let S be a subset of T.
  Let u,v be supremums of S in T. Then u = v.

Lemma InfUn.  Let S be a subset of T.
  Let u,v be infimums of S in T. Then u = v.

Definition DefCLat.   A complete lattice is a set S such that
  every subset of S has an infimum in S and a supremum in S.

[function/-s] [point/-s]

Signature ConMap.   A function is a notion.

Let f stand for a function.

Signature DomSort.  Dom f is a set.
Signature RanSort.  Ran f is a set.

Definition DefDom.  f is on S iff Dom f = Ran f = S.

Signature ImgSort.  Let x belong to Dom f.
  f(x) is an element of Ran f.

Definition DefFix.  A fixed point of f is
  an element x of Dom f such that f(x) = x.

Definition DefMonot.  f is monotone iff
  for all (x,y << Dom f)  x <= y  =>  f(x) <= f(y).

Theorem Tarski.
  Let U be a complete lattice and f be a monotone function on U.
  Let S be the set of fixed points of f.
  S is a complete lattice.
Proof.
  Let T be a subset of S.
  Let us show that T has a supremum in S.
    Take P = { x << U | f(x) <= x and x is an upper bound of T in U }.
    Take an infimum p of P in U.
    f(p) is a lower bound of P in U and an upper bound of T in U.
    Hence p is a fixed point of f and a supremum of T in S.
  end.
  Let us show that T has an infimum in S.
    Take Q = { x << U | x <= f(x) and x is a lower bound of T in U }.
    Take a supremum q of Q in U.
    f(q) is an upper bound of Q in U and a lower bound of T in U.
    Hence q is a fixed point of f and an infimum of T in S.
  end.
qed.
```

B Newman's lemma

In this appendix we give a verified ForTheL formalization of the Newman's lemma about confluence of term rewriting systems. The syntax extensions in the text below are the same as in the previous appendix.

```
[element/-s] [system/-s] [reduct/-s]

Signature ElmSort.  An element is a notion.
Signature RelSort.  A rewriting system is a notion.

Let a,b,c,d,u,v,w,x,y,z denote elements.
Let R,S,T denote rewriting systems.

Signature Reduct.   A reduct of x in R is an element.

Let x -R> y stand for y is a reduct of x in R.

Signature WFOrd.    x -<- y is a relation.

Signature TCbr.     x -R+> y is a relation.

Definition TCDef.   x -R+> y <=> x -R> y \/ exists z : x -R> z -R+> y.

Axiom TCTrans.      x -R+> y -R+> z => x -R+> z.

Definition TCRDef.  x -R*> y <=> x = y \/ x -R+> y.

Lemma TCRTrans.     x -R*> y -R*> z => x -R*> z.

Definition CRDef.   R is confluent iff
  for all a,b,c  such that a -R*> b,c
  there exists d such that b,c -R*> d.

Definition WCRDef.  R is locally confluent iff
  for all a,b,c  such that a -R> b,c
  there exists d such that b,c -R*> d.

Definition Termin.  R is terminating iff for all a,b
  a -R+> b => b -<- a.

Definition NFRDef.  A normal form of x in R is an element y
                such that x -R*> y and y has no reducts in R.

Lemma TermNF.   Let R be a terminating rewriting system.
                Every element x has a normal form in R.
Proof by induction. Obvious.
```

```
Lemma Newman.
  Any locally confluent terminating rewriting system is confluent.
Proof.
  Let R be locally confluent and terminating.
  Let us demonstrate by induction that for all a,b,c
  such that a -R*> b,c there exists d such that b,c -R*> d.

    Assume that a -R+> b,c.

    Take u such that a -R> u -R*> b.
    Take v such that a -R> v -R*> c.
    Take w such that u,v -R*> w.
    Take a normal form d of w in R.

    b -R*> d. Indeed take x such that b,d -R*> x.
    c -R*> d. Indeed take y such that c,d -R*> y.
  end.
qed.
```

C Chinese remainder theorem

Below we give a verified ForTheL formalization of the Chinese remainder theorem in a commutative unitary ring and of the Bezout's theorem in a euclidean ring. The syntax extensions in the text are the same as in the appendix A.

```
[element/-s]

Signature ElmSort. An element is a notion.

Let a,b,c,x,y,z,u,v,w denote elements.

Signature SortsC.  0 is an element.
Signature SortsC.  1 is an element.
Signature SortsU.  -x is an element.
Signature SortsB.  x + y is an element.
Signature SortsB.  x * y is an element.

Let x is nonzero stand for x != 0.
Let x - y stand for x + -y.

Axiom AddComm.   x + y = y + x.
Axiom AddAsso.  (x + y) + z = x + (y + z).
Axiom AddZero.   x + 0 = x = 0 + x.
Axiom AddInvr.   x + -x = 0 = -x + x.

Axiom MulComm.   x * y = y * x.
Axiom MulAsso.  (x * y) * z = x * (y * z).
Axiom MulUnit.   x * 1 = x = 1 * x.

Axiom AMDistr.  x * (y + z) = (x * y) + (x * z) and
                (y + z) * x = (y * x) + (z * x).

Lemma MulMnOne. -1 * x = -x = x * -1.
Lemma MulZero.  x * 0 = 0 = 0 * x.

Axiom Cancel.   If x * y = 0 then x = 0 or y = 0.

Axiom UnNeZr.   1 != 0.

[set/-s] [belong/-s]

Signature SetSort.  A set is a notion.

Let X,Y,Z,U,V,W denote sets.

Signature EOfElem.  An element of W is an element.
```

```
Let x << W denote (x is an element of W).
Let x belongs to W denote (x is an element of W).

Axiom SetEq.  If every element of X belongs to Y
    and every element of Y belongs to X then X = Y.

Definition DefSSum. X + Y = { x + y | x << X and y << Y }.

Definition DefSInt. X ** Y = { h | h << X and h << Y }.

[ideal/-s]

Definition DefIdeal.
  An ideal is a set X such that for every x << X
    forall y << X (x + y) << X and
    forall z (z * x) << X.

Let I,J denote ideals.

Lemma IdeSum.   I + J is an ideal.
Proof.
  Let x, y belong to I + J and z be an element.
  Take k << I and l << J such that x = k + l.
  Take m << I and n << J such that y = m + n.
  k + m belongs to I and l + n belongs to J.
  z * k belongs to I and z * l belongs to J.
  We have x + y = (k + m) + (l + n) (by AddComm, AddAsso).
  We have z * x = (z * k) + (z * l) (by AMDistr).
  Hence (x + y), (z * x) belong to I + J.
qed.

Lemma IdeInt.   I ** J is an ideal.

Definition DefMod.  x = y (mod I)  iff  x - y << I.

Theorem ChineseRemainder.
  Suppose that every element belongs to I + J.
  Let x, y be elements.
  There exists an element w such that
    w = x (mod I) and w = y (mod J).
Proof.
  Take a << I and b << J such that a + b = 1.
  Take w = (y * a) + (x * b).

  Let us show that w - x belongs to I.
    w - x = (y * a) + ((x * b) - x) (by AddAsso).
    x * (b - 1) belongs to I.
  end.

  Let us show that w - y belongs to J.
    w - y = (x * b) + ((y * a) - y) (by AddAsso,AddComm).
    y * (a - 1) belongs to J.
  end.
qed.
```

```
[number/-s]

Signature NatSort.  A natural number is a notion.

Signature EucSort.  Let x be a nonzero element.
  |x| is a natural number.

Signature NatLess.  Let i,j be natural numbers.
  i -<- j is a relation.

Axiom Division.
  Let x, y be elements and y != 0.
  There exist elements q,r such that
    x = (q * y) + r and (r != 0 => |r| -<- |y|).

[divisor/-s] [divide/-s]

Definition DefDiv.
  x divides y  iff  for some z (x * z = y).

Let x | y stand for x divides y.
Let x is divided by y stand for y | x.

Definition DefDvs.
  A divisor of x is an element that divides x.

Definition DefGCD.
  A gcd of x and y is a common divisor c of x and y
    such that any common divisor of x and y divides c.

Definition DefRel.
  x, y are relatively prime iff 1 is a gcd of x and y.

Definition DefPrIdeal.
  <c> is { c * x | x is an element }.

Lemma PrIdeal.  <c> is an ideal.
Proof.
  Let x,y belong to <c> and z be an element.
  Take an element u such that c * u = x.
  Take an element v such that c * v = y.
  We have x + y = c * (u + v) (by AMDistr).
  We have z * x = c * (u * z) (by MulComm,MulAsso).
  Hence (x + y), (z * x) belong to <c>.
qed.

Theorem Bezout.  Let a, b be elements.
  Assume that a is nonzero or b is nonzero.
  Let c be a gcd of a and b. Then c belongs to <a> + <b>.
Proof.
  Take an ideal I equal to <a> + <b>.
  We have 0,a << <a> and 0,b << <b>.
  Hence there exists a nonzero element of <a> + <b>.

  Take a nonzero u << I
    such that for no nonzero v << I (|v| -<- |u|).
  Indeed we can show by induction on |w| that
    for every nonzero w << I there exists nonzero u << I
      such that for no nonzero v << I (|v| -<- |u|).
    Obvious.

  u is a common divisor of a and b.
  proof.
    Assume the contrary.
    For some elements x,y  u = (a * x) + (b * y).
    proof.
      Take k << <a> and l << <b> such that u = k + l.
      Hence the thesis.
    end.

    Case u does not divide a.
      Take elements q,r such that a = (q * u) + r
        and (r = 0 \/ |r| -<- |u|).
      r is nonzero.
      - (q * u) belongs to I.
      a belongs to I.
      r = - (q * u) + a.
      Hence r belongs to I.
    end.

    Case u does not divide b.
      Take elements q,r such that b = (q * u) + r
        and (r = 0 \/ |r| -<- |u|).
      r is nonzero.
      - (q * u) belongs to I.
      b belongs to I.
      r = - (q * u) + b.
      Hence r belongs to I.
    end.
  end.

  Hence u divides c.
  Hence the thesis.
qed.
```