

ForTheL Reference Manual

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

2nd November 2004

Contents

Introduction	2
1 Lexical structure	3
2 Units of ForTheL	4
2.1 Syntactic primitives	5
2.2 Notions	7
2.3 Terms	8
2.4 Predicates	10
2.5 Statements	12
2.6 Formula image of statements	13
2.7 Variable description	22
2.8 Definition statements	23
3 Sections of ForTheL	24
3.1 Top-level sections and sentences	24
3.2 Proof sections	26
3.3 Precedence and declaration	28
3.4 Proof normalization	28
3.5 Formula image of sections	32
3.6 Well-formedness and correctness	33
Index	34

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 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 unit 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 or notion unit. Chains of sentences form compound sections: low-level ones, such as proofs and proof cases, and top-level ones, such as axioms, definitions, and propositions. A ForTheL section has no fixed semantics, instead, it possesses a number of characteristics defined partly by its content and partly by its context — the set of sections that *logically precede* that section in the text. In particular, we introduce the notion of a *correct* section.

4. Finally, a ForTheL *text* is a sequence of top-level sections and special constructs called *introducers* which extend the signature of the text with new primitives. A syntactically valid ForTheL text is correct whenever each of its top-level sections is.

1 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:

$$\textit{nonterm} \rightarrow \textit{alt}_1 \mid \textit{alt}_2 \mid \dots \mid \textit{alt}_n$$

and the following conventions are adopted:

$$\begin{array}{llll} \textit{pat}_1 \mid \textit{pat}_2 & \text{choice} & (\textit{pattern}) & \text{grouping} \\ [\textit{pattern}] & \text{optional} & \{\textit{pattern}\} & \text{zero or more repetitions} \end{array}$$

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

$$\begin{aligned} \textit{lexeme} &\rightarrow \textit{word} \mid \textit{symbol} \\ \textit{word} &\rightarrow \textit{alphanum} \{ \textit{alphanum} \} \\ \textit{alphanum} &\rightarrow \textit{alpha} \mid \textit{numeric} \mid _ \\ \textit{alpha} &\rightarrow \textit{small} \mid \textit{capital} \\ \textit{small} &\rightarrow \mathbf{a} \mid \dots \mid \mathbf{z} \\ \textit{capital} &\rightarrow \mathbf{A} \mid \dots \mid \mathbf{Z} \\ \textit{numeric} &\rightarrow \mathbf{0} \mid \dots \mid \mathbf{9} \\ \textit{symbol} &\rightarrow (\mid) \mid [\mid] \mid \{ \mid \} \mid < \mid > \mid ' \mid ' \mid " \mid / \mid \backslash \mid | \mid \% \\ &\quad | \mid ! \mid ? \mid @ \mid \$ \mid \& \mid \% \mid \sim \mid + \mid - \mid * \mid = \mid : \mid ; \mid , \mid \cdot \\ \textit{whitespace} &\rightarrow \textit{whit token} \{ \textit{whit token} \} \\ \textit{whit token} &\rightarrow \textit{space} \mid \textit{newline} \mid \textit{comment} \\ \textit{comment} &\rightarrow \# \{ \text{any character except newline} \} \textit{newline} \\ \textit{space} &\rightarrow \text{a space} \mid \text{a horizontal tab} \\ \textit{newline} &\rightarrow \text{a newline} \end{aligned}$$

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 2.1) 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.

2 Units of ForTheL

In order to give an idea about this level of ForTheL, we begin with a couple of examples of ForTheL statements. For each statement we give its first-order meaning.

ForTheL: every bird is a feathery animal

Meaning: $\forall x (aBird(x) \supset (aAnimal(x) \wedge isFeathery(x)))$

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

Meaning: $\exists S (aSet(S) \wedge \forall x (aSubsetOf(x, S) \supset x = S))$

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

Meaning: $\forall m ((aNumber(m) \wedge isNatural(m) \wedge$
 $\wedge isGreaterThan(m, Zero)) \supset Divides(m, Factorial(m)))$

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

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

ForTheL: no equation in G has a positive solution

Meaning: $\forall x ((aEquation(x) \wedge isIn(x, G)) \supset$
 $\supset \neg \exists y (aSolutionOf(y, x) \wedge isPositive(y)))$

Just as a ForTheL text is a composition of sentences and compound sections, a ForTheL sentence is a composition of *units*. Units are of four general kinds:

- *notion* denotes a general, possibly parametrized, class of objects: 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:

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

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

2.1 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 primitive notions: `element of arg1`
- *definite nouns*, to form functional terms: `zero`, `power set of arg1`
- *adjectives* and *verbs*, to form predicates: `converges`, `equal to arg1`
- *function symbols*, to form symbolic terms: `arg1 + arg2`, `min arg1`
- *predicate symbols*, to form symbolic statements: `arg1 <= arg2`, `arg1 : arg2 -> arg3`

Base primitives are introduced directly in the text with the help of special ForTheL constructions, called *introductors*. 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 arg1` or `complement of arg1 to arg2` produce special primitives to be used in possessive predicate units: `has an element`, `of an infinite cardinality`. The groups of derived primitives will be considered in the subsequent sections.

The syntax of introductors is as follows (one line per base group):

```

introducer → [ ( a | an ) pattern [ @ [ a | an ] nounNotion ] ]
            | [ the pattern [ @ plainTerm ] ]
            | [ variable is pattern [ @ statement ] ]
            | [ variable pattern [ @ statement ] ]
            | [ symbPattern @ plainTerm ]
            | [ symbPattern @ statement ]

```

Each introducer contains a *pattern* which determines the syntax of the new-introduced primitive. A primitive may (a symbolic one, should) be introduced as a *synonym* for some unit of the appropriate kind; in this case, the pattern is followed by an “at” symbol and a *target* unit. The nonterminals *nounNotion*, *plainTerm*, and *statement* used as targets

will be defined and explained in the following sections. All the primitives used in target units should be already present in the vocabulary, so that a self-reference or a mutual synonymy is impossible.

A pattern is a nonempty chain of *tokens* (which define terminals for the new primitive) alternated with variables (which denote argument places):

$$\begin{aligned}
 pattern &\rightarrow \{ tokens \} tokens [variable \{ tokens variable \}] \\
 tokens &\rightarrow token [/ token] \\
 token &\rightarrow small \{ small \} \\
 symbPattern &\rightarrow [variable] symbToken \{ variable symbToken \} [variable] \\
 &\quad | \quad word (variable \{ , variable \}) \\
 &\quad | \quad word [variable] \\
 symbToken &\rightarrow symbol \{ symbol \}
 \end{aligned}$$

In a non-symbolic pattern, one may give several equivalent variants for the same token. This allows us to use nouns and verbs in both singular and plural forms. Tokens and symbolic tokens should not contain spaces. The articles and the forms of the verb “to be” are not accepted as tokens.

In the introductors of verbs, the leading variable should not be named **a** or **A** to avoid collision with introduction of class-nouns. All the variables in a pattern (including the leading variable in verb and adjective introductors) should be different. In a synonym introducer, all the free variables (2.6) of the target should be present in the pattern.

The parser can always determine which kind of primitive is defined in an introducer, either by looking at the pattern or, in the case of symbolic primitives, by recognizing the target unit as a term or a statement.

As soon as a new introducer is read, new alternatives are added to the grammar rules responsible for the corresponding primitives:

Introducer	Primitive
[a subset/subsets of S]	(subset subsets) [names] (of) term
[the power set/sets of S]	(power) (set sets) (of) term
[U is infinite @ U is not finite]	(infinite)
[m divides/divide n]	(divides divide) term
[Pow x @ the power set of x]	(Pow) symbTerm
[x <= y @ x is not greater than y]	symbTerms (<=) symbTerm

Pattern tokens are transformed to grouped choices of terminals and variables are replaced with the nonterminal *term* (*symbTerm*, in symbolic patterns) which “reads” the arguments of a primitive. In symbolic predicate patterns which begin with a variable, this variable is replaced with the nonterminal *symbTerms*. Presence of a target unit does not influence the syntax of the new primitive. In class-nouns, the optional nonterminal *names* (see 2.2) is inserted one token before the first argument place or in the end of the pattern if there are no arguments.

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.

2.2 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:

$$\textit{primaryNotion} \rightarrow \textit{primClassNoun} \mid \textit{notionSymbol}$$

$$\begin{aligned} \textit{primClassNoun} \rightarrow & (\textit{set} \mid \textit{sets}) [\textit{names}] \\ & \mid (\textit{element} \mid \textit{elements}) [\textit{names}] (\textit{of}) \textit{term} \\ & \mid (\textit{function} \mid \textit{functions}) [\textit{names}] (\textit{from}) \textit{term} (\textit{to}) \textit{term} \\ & \mid \dots \end{aligned}$$

$$\textit{notionSymbol} \rightarrow \textit{primNotionSymbol} \mid (\textit{primNotionSymbol})$$

$$\begin{aligned} \textit{primNotionSymbol} \rightarrow & \textit{names} (<<) \textit{symbTerm} \\ & \mid \textit{names} (:) \textit{symbTerm} (->) \textit{symbTerm} \\ & \mid \dots \end{aligned}$$

$$\textit{names} \rightarrow \textit{variable} \{ , \textit{variable} \}$$

Notion symbols are derived primitives. They are produced automatically from descriptive predicate symbols (see 2.7). The first argument place in a notion symbol is taken by the nonterminal *names* and the rest is the same as in the corresponding predicate symbol. The notion symbols above could be produced by the following predicate symbol introductors:

$$\begin{aligned} [x << y @ x \text{ is an element of } y] \\ [f : D \rightarrow R @ f \text{ is a function from } D \text{ to } R] \end{aligned}$$

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-noun primitives, names may be omitted:

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:

$$\begin{aligned} \text{leftAttribute} &\rightarrow \text{primSimpleAdjective} \mid \text{primSimpleAdjectiveM} \\ \text{rightAttribute} &\rightarrow \text{isPredicate} \{ \text{and isPredicate} \} \\ &\mid \text{that doesPredicate} \{ \text{and doesPredicate} \} \\ &\mid \text{such that statement} \end{aligned}$$

Simple adjectives which form left attributes are derived primitives, too. They are produced from those adjectives and m-adjectives (2.4) which have no argument places: every such primitive is just copied to the group of simple (m-)adjectives:

$$\begin{aligned} \text{primSimpleAdjective} &\rightarrow (\text{prime}) \mid (\text{empty}) \mid (\text{natural}) \mid \dots \\ \text{primSimpleAdjectiveM} &\rightarrow (\text{equal}) \mid (\text{parallel}) \mid (\text{disjoint}) \mid \dots \end{aligned}$$

Now we define the syntax of notion:

$$\text{notion} \rightarrow \{ \text{leftAttribute} \} \text{primaryNotion} [\text{rightAttribute}]$$

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.

2.3 Terms

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

$$\text{term} \rightarrow \text{quantifiedNotion} \mid \text{definiteTerm}$$

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:

$$\begin{aligned} \text{quantifiedNotion} &\rightarrow \text{realQuantifiedNotion} \mid (\text{realQuantifiedNotion}) \\ \text{realQuantifiedNotion} &\rightarrow (\text{every} \mid \text{each} \mid \text{all} \mid \text{any}) \text{notion} \\ &\mid \text{some notion} \\ &\mid \text{no notion} \end{aligned}$$

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 function symbols. We adopt the following conventions about precedence and associativity of function symbols:

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

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

$$definiteTerm \rightarrow realDefiniteTerm \mid (realDefiniteTerm)$$

$$realDefiniteTerm \rightarrow [\mathbf{the}] primDefiniteNoun \mid symbTerm$$

$$symbTerm \rightarrow primInfixFunctionSymbol \mid prefixSymbTerm$$

$$prefixSymbTerm \rightarrow primPrefixFunctionSymbol \mid postfixSymbTerm$$

$$postfixSymbTerm \rightarrow primPostfixFunctionSymbol \mid (symbTerm) \mid variable$$

Here follow typical primitives for definite nouns and function symbols:

$$\begin{aligned} primDefiniteNoun &\rightarrow (\mathbf{zero} \mid \mathbf{zeroes}) \\ &\quad \mid (\mathbf{order} \mid \mathbf{orders}) (\mathbf{of}) term \\ &\quad \mid \dots \end{aligned}$$

$$\begin{aligned} primInfixFunctionSymbol &\rightarrow prefixSymbTerm (*) symbTerm \\ &\quad \mid \dots \end{aligned}$$

$$\begin{aligned} primPrefixFunctionSymbol &\rightarrow (\mathbf{min}) prefixSymbTerm \\ &\quad \mid \dots \end{aligned}$$

$$\begin{aligned} primPostfixFunctionSymbol &\rightarrow (0) \\ &\quad \mid (\mathbf{exp}) (() symbTerm (,) symbTerm ()) \\ &\quad \mid postfixSymbTerm (() symbTerm ()) \\ &\quad \mid \dots \end{aligned}$$

Note the last primitive in the group of postfix functions symbols. 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:

$$plainTerm \rightarrow realPlainTerm \mid (realPlainTerm)$$

$$realPlainTerm \rightarrow [\mathbf{the}] primPlainNoun \mid symbTerm$$

$$\begin{aligned} primPlainNoun &\rightarrow (\mathbf{zero} \mid \mathbf{zeroes}) \\ &\quad \mid (\mathbf{order} \mid \mathbf{orders}) (\mathbf{of}) plainTerm \\ &\quad \mid \dots \end{aligned}$$

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

2.4 Predicates

Functions and notions 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:

$$\begin{aligned}
 \text{doesPredicate} &\rightarrow [\text{does} \mid \text{do}] [\text{not}] \text{primVerb} \\
 &\quad | [\text{does} \mid \text{do}] [\text{not}] [\text{pairwise}] \text{primVerbM} \\
 &\quad | (\text{has} \mid \text{have}) \text{hasPredicate} \\
 &\quad | (\text{is} \mid \text{are} \mid \text{be}) \text{isPredicate} \{ \text{and isPredicate} \} \\
 &\quad | (\text{is} \mid \text{are} \mid \text{be}) \text{is_aPredicate} \{ \text{and is_aPredicate} \} \\
 \\
 \text{isPredicate} &\rightarrow [\text{not}] \text{primAdjective} \\
 &\quad | [\text{not}] [\text{pairwise}] \text{primAdjectiveM} \\
 &\quad | (\text{with} \mid \text{of} \mid \text{having}) \text{hasPredicate} \\
 \\
 \text{is_aPredicate} &\rightarrow [\text{not}] [\text{a} \mid \text{an}] \text{nounNotion} \\
 &\quad | [\text{not}] \text{definiteTerm} \\
 \\
 \text{hasPredicate} &\rightarrow [\text{a} \mid \text{an} \mid \text{the}] \text{possessed} \{ \text{and} [\text{a} \mid \text{an} \mid \text{the}] \text{possessed} \} \\
 &\quad | \text{no possessed}
 \end{aligned}$$

Primitive adjectives and verbs look as follows:

$$\begin{aligned}
 \text{primVerb} &\rightarrow (\text{converges} \mid \text{converge}) \\
 &\quad | (\text{divides} \mid \text{divide}) \text{term} \\
 &\quad | (\text{belongs} \mid \text{belong}) (\text{to}) \text{term} \\
 &\quad | (\text{joins} \mid \text{join}) \text{term} (\text{with}) \text{term} \\
 &\quad | \dots \\
 \\
 \text{primAdjective} &\rightarrow (\text{prime}) \\
 &\quad | (\text{dividing}) \text{term} \\
 &\quad | (\text{equal}) (\text{to}) \text{term} \\
 &\quad | (\text{less}) (\text{than}) \text{term} \\
 &\quad | \dots
 \end{aligned}$$

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 introducer, 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 ((parallel) (to) term) produces the m-adjective ((parallel)) added both to *primAdjectiveM* and to *primSimpleAdjectiveM*; and the verb primitive ((commutes | commute) (with) term (wrt) term) produces the m-verb ((commutes | commute) (wrt) term).

Obviously, predicate units containing m-primitives (*m-predicates*) ought to be used with several subjects: X,Y,Z commute wrt N, parallel lines l,m. Though expressions like these make sense for symmetric (or even transitive-symmetric) predicates only, we do not track such properties on the syntactic level and so we produce new m-primitives for all primitive adjectives and verbs.

It is supposed by default that multisubject predicates are both symmetric and transitive. 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.

$$\begin{aligned}
 \textit{primVerbM} &\rightarrow (\textit{collides} \mid \textit{collide}) \\
 &\quad \mid (\textit{commutes} \mid \textit{commute}) (\textit{wrt}) \textit{term} \\
 &\quad \mid \dots \\
 \textit{primAdjectiveM} &\rightarrow (\textit{equal}) \\
 &\quad \mid (\textit{adjacent}) (\textit{in}) \textit{term} \\
 &\quad \mid \dots
 \end{aligned}$$

With the help of “is a”-predicates we say that an object is either “described” by a notion or equals to a definite term: X is a natural number, X is the order of G. We cannot use symbolic notions in “is a”-predicates or as targets in introduction, so we define a special nonterminal *nounNotion* for the notions built upon class-nouns.

$$\textit{nounNotion} \rightarrow \{ \textit{leftAttribute} \} \textit{primClassNoun} [\textit{rightAttribute}]$$

Remark. Class-noun primitives used in *nounNotion* should have at most one name. Noun notions that have a name and those ones used as targets should not have in attributes any m-predicate applied to the notion. Thus, the units an equal number X, a line N that is parallel, a set S such that S is disjoint are forbidden.

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

$$\begin{aligned}
 \textit{possessed} &\rightarrow \{ \textit{leftAttribute} \} \textit{primPossessedNoun} [\textit{rightAttribute}] \\
 \textit{primPossessedNoun} &\rightarrow (\textit{element} \mid \textit{elements}) [\textit{names}] \\
 &\quad \mid (\textit{solution} \mid \textit{solutions}) [\textit{names}] \\
 &\quad \mid (\textit{order} \mid \textit{orders}) [\textit{names}] \\
 &\quad \mid \dots
 \end{aligned}$$

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 [*names*] if needed.

Thus, the primitive class-noun

$$(\text{ambassador} \mid \text{ambassadors}) [\text{names}] (\text{of}) \text{term} (\text{in}) \text{term}$$

produces the new possessed noun primitive

$$(\text{ambassador} \mid \text{ambassadors}) [\text{names}] (\text{in}) \text{term}$$

while the primitive definite noun `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.

2.5 Statements

Statements are ForTheL counterparts of logic formulas. First, we consider non-compound, *atomic* statement units. Such a statement may be of four kinds:

$$\begin{aligned} \text{atomicStatement} &\rightarrow \text{simpleStatement} \\ &\mid \text{thereIsStatement} \\ &\mid [\text{we have}] \text{symbStatement} \\ &\mid [\text{we have}] \text{specialStatement} \end{aligned}$$

Simple statements apply predicates to terms:

$$\begin{aligned} \text{simpleStatement} &\rightarrow \text{terms} \text{doesPredicate} \{ \text{and} \text{doesPredicate} \} \\ \text{terms} &\rightarrow \text{term} \{ (, \mid \text{and}) \text{term} \} \end{aligned}$$

So called “there is”-statements affirm or deny nonemptiness of some notion’s class:

$$\begin{aligned} \text{thereIsStatement} &\rightarrow \text{there} (\text{exists} \mid \text{exist}) \text{notions} \\ &\mid \text{there} (\text{exists} \mid \text{exist}) \text{no} \text{notion} \\ \text{notions} &\rightarrow [\text{a} \mid \text{an}] \text{notion} \{ (, \mid \text{and}) [\text{a} \mid \text{an}] \text{notion} \} \end{aligned}$$

Symbolic statements are composed in a traditional first-order syntax from symbolic predicates 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.

$$\begin{aligned}
\textit{symbStatement} &\rightarrow \textit{forall} \textit{notionSymbol} \textit{symbStatement} \\
&| \textit{exists} \textit{notionSymbol} \textit{symbStatement} \\
&| \textit{symbStatement} \textit{<=>} \textit{symbStatement} \\
&| \textit{symbStatement} \textit{=>} \textit{symbStatement} \\
&| \textit{symbStatement} \textit{\setminus/} \textit{symbStatement} \\
&| \textit{symbStatement} \textit{\setminus\} \textit{symbStatement} \\
&| \textit{not} \textit{symbStatement} \\
&| (\textit{statement}) \\
&| \textit{primPredicateSymbol} \\
\\
\textit{primPredicateSymbol} &\rightarrow \textit{symbTerms} (=) \textit{symbTerm} \\
&| \textit{symbTerms} (!=) \textit{symbTerm} \\
&| \textit{symbTerms} (-<-) \textit{symbTerm} \\
&| \textit{symbTerms} (:) \textit{symbTerm} (->) \textit{symbTerm} \\
&| (\textit{Nat}) (() \textit{symbTerm} ()) \\
&| \dots \\
\\
\textit{symbTerms} &\rightarrow \textit{symbTerm} \{ , \textit{symbTerm} \}
\end{aligned}$$

Binary predicate symbols =, != and -<- are preintroduced in ForTheL. The first two are synonyms to the (non-)equality statement, the third one denotes an abstract well-founded relation and is used for induction proofs (3.2).

The special statements **thesis** and **contrary** denote the (negation of) the thesis statement for the current sentence (3.4), and **contradiction** stands for logical *falsum*.

$$\textit{specialStatement} \rightarrow [\textit{the}] \textit{thesis} \mid [\textit{the}] \textit{contrary} \mid [\textit{a}|\textit{an}] \textit{contradiction}$$

Atomic statements are composed with prepositions and conjunctions as follows:

$$\begin{aligned}
\textit{statement} &\rightarrow \textit{headStatement} \mid \textit{chainStatement} \\
\\
\textit{headStatement} &\rightarrow \textit{for} \textit{quantifiedNotion} \{ \textit{and} \textit{quantifiedNotion} \} \textit{statement} \\
&| \textit{if} \textit{statement} \textit{then} \textit{statement} \\
&| \textit{it is wrong that} \textit{statement} \\
\\
\textit{chainStatement} &\rightarrow \textit{andChain} [\textit{and} \textit{headStatement}] \\
&| \textit{orChain} [\textit{or} \textit{headStatement}] \\
&| (\textit{andChain} \mid \textit{orChain}) \textit{iff} \textit{statement} \\
\\
\textit{andChain} &\rightarrow \textit{atomicStatement} \{ \textit{and} \textit{atomicStatement} \} \\
\\
\textit{orChain} &\rightarrow \textit{atomicStatement} \{ \textit{or} \textit{atomicStatement} \}
\end{aligned}$$

2.6 Formula image of statements

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 listed below. Each rule must be applied as many times as possible before the next rule could be applied for the first time.

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

- (a) 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.
- (b) We split “and”-chains of notions in quantified statements. The following rule is applied to occurrences of *headStatement*:

```
for (quantifiedNotion)O
    and (quantifiedNotion { and quantifiedNotion })T (statement)S
⇒ for  $O$  for  $T$   $S$ 
```

- (c) Articles `a`, `an`, `the` and the prefix `we have` are removed. Negation `it is wrong that` is replaced with `not`. 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.

2. **Resolving quantified notions I.** In what follows, we denote with \mathbf{N} any of the nonterminals *primClassNoun*, *primPossessedNoun*, or *primNotionSymbol*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*, *nounNotion*, *notionSymbol*, *quantifiedNotion*, and *possessed* in an obvious manner. As the rule (1a) has been exhaustively applied, the list of names is now nonempty for all occurrences of \mathbf{N} in the unit under consideration.

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 .

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

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

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

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 $\mathbf{n}(O)$:

- (a) $(\text{terms}[(\text{quantifiedNotion})_O])_S (\text{doesPredicate } \{ \text{and doesPredicate} \})_T$
 \Rightarrow for O $S[O \rightarrow \mathbf{n}(O)] T$
- (b) for $(\text{quantifiedNotion}[(\text{quantifiedNotion})_O])_S (\text{statement})_T$
 \Rightarrow for O for $S[O \rightarrow \mathbf{n}(O)] T$

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 (2a)
 for (every member M of some committee C) M declares some opinion O
 such that every member N of C supports O
 \Downarrow (2a)
 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 (2b)
 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

By applying the rules (1a-1c, 2a-2b) to a statement S , we obtain the *casual normal form*² of S , denoted S^\dagger . We use casual normal forms to define the transformations of thesis statement in (3.4).

3. **Unifying attributes.** Now we transform attribute subunits to a common unified form. The following rules apply to occurrences of the nonterminals *notion*, *nounNotion*, and *possessed*.

- (a) $(\{ \text{leftAttribute} \})_L (\mathbf{N})_O (\text{isPredicate } \{ \text{and isPredicate} \})_A$
 \Rightarrow $L O$ such that $\mathbf{n}(O)$ is A
- (b) $(\{ \text{leftAttribute} \})_L (\mathbf{N})_O \text{ that } (\text{doesPredicate } \{ \text{and doesPredicate} \})_V$
 \Rightarrow $L O$ such that $\mathbf{n}(O)$ V
- (c) $(\{ \text{leftAttribute} \})_L (\text{leftAttribute})_A (\mathbf{N})_O [\text{such that } (\text{statement})_S]$
 \Rightarrow $L O$ such that $\mathbf{n}(O)$ is A [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:

²the term *casual* is casual and will be replaced with some meaningful word as soon as I find one.

prime natural number X that divides N
 \Downarrow (3b)
 prime natural number X such that X divides N
 \Downarrow (3c)
 prime number X such that X is natural and X divides N
 \Downarrow (3c)
 number X such that X is prime and X is natural and X divides N

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

- (a) $(terms)_O (doesPredicate)_P$ and $(doesPredicate \{ \text{and } doesPredicate \})_T$
 $\Rightarrow O P$ and $O T$
- (b) $(terms)_O$ is $(isPredicate)_P$ and $(isPredicate \{ \text{and } isPredicate \})_T$
 $\Rightarrow O$ is P and O is T
- (c) $(terms)_O$ is $(is_aPredicate)_P$ and $(is_aPredicate \{ \text{and } is_aPredicate \})_T$
 $\Rightarrow O$ is P and O is T

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

5. **Elimination of “there exists”.** In what follows, \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*:

- (a) there exists $(notionSymbol)_O$ [such that $(statement)_S$] [, $(notions)_T$]
 \Rightarrow for some $\mathbf{n}(O)$ (O [and S] [and there exists T])
- (b) there exists no $(notionSymbol)_O$ [such that $(statement)_S$]
 \Rightarrow for every $\mathbf{n}(O)$ (not (O [and S]))
- (c) there exists $(notion)_O$ [, $(notions)_T$]
 \Rightarrow for some $\mathbf{n}(O)$ ($\mathbf{n}(\bar{O})$ is \bar{O} [and there exists T])
- (d) there exists no $(notion)_O$
 \Rightarrow for every $\mathbf{n}(O)$ ($\mathbf{n}(\bar{O})$ 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 (1c,3c)
 there exists number E, divisors G,H of E such that G,H is prime
 \Downarrow (5c)
 for some E (E is a number and there exists
 divisors G,H of E such that G,H is prime)
 \Downarrow (5c)
 for some E (E is a number and for some G,H
 (G,H is divisors of E such that G,H is prime))

6. Resolving quantified notions II. The following group of rules removes quantified notions from the rest of the occurrences of *term*. Like the rules (2a-2b) 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 $\mathbf{n}(O)$:

- (a) $(terms)_T (doesPredicate[(quantifiedNotion)_O])_S$
 \Rightarrow for O T $S[O \rightarrow \mathbf{n}(O)]$
- (b) for $(quantifiedNotion[(quantifiedNotion)_O])_S (statement)_T$
 \Rightarrow for O for $S[O \rightarrow \mathbf{n}(O)]$ 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 (6a)
 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

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 rules below apply to occurrences of *headStatement* (7a-7f) and *symbStatement* (7g-7h):

- (a) for $[(\]$ every $(notionSymbol)_O$ $[\text{such that } (statement)_T$ $[\]]$ $(statement)_S$
 \Rightarrow for every $\mathbf{n}(O)$ (if O $[\text{and } T]$ then S)
- (b) for $[(\]$ some $(notionSymbol)_O$ $[\text{such that } (statement)_T$ $[\]]$ $(statement)_S$
 \Rightarrow for some $\mathbf{n}(O)$ (O $[\text{and } T]$ and S)
- (c) for $[(\]$ no $(notionSymbol)_O$ $[\text{such that } (statement)_T$ $[\]]$ $(statement)_S$
 \Rightarrow for every $\mathbf{n}(O)$ (if O $[\text{and } T]$ then (not S))
- (d) for $[(\]$ every $(notion)_O$ $[\]]$ $(statement)_S$
 \Rightarrow for every $\mathbf{n}(O)$ (if $\mathbf{n}(O)$ is \bar{O} then S)
- (e) for $[(\]$ some $(notion)_O$ $[\]]$ $(statement)_S$
 \Rightarrow for some $\mathbf{n}(O)$ ($\mathbf{n}(O)$ is \bar{O} and S)
- (f) for $[(\]$ no $(notion)_O$ $[\]]$ $(statement)_S$
 \Rightarrow for every $\mathbf{n}(O)$ (if $\mathbf{n}(O)$ is \bar{O} then (not S))
- (g) forall $(notionSymbol)_O (symbStatement)_S$
 \Rightarrow forall $\mathbf{n}(O)$ ($O \Rightarrow S$)
- (h) exists $(notionSymbol)_O (symbStatement)_S$
 \Rightarrow exists $\mathbf{n}(O)$ ($O \wedge S$)

8. Eliminating “has”-predicates. Given an occurrence O of *primPossessedNoun* and an occurrence N of *term*, we denote with $O[N]$ the original class-noun or definite noun primitive with N put back in the first argument place. For example, if O is *element* and N is *S* then $O[N]$ is *element of S*.

The following rules apply to occurrences of *simpleStatement*:

- (a) $(terms)_N$ is (with | of | having) $(hasPredicate)_S$
 $\Rightarrow N$ has S
- (b) $(term)_N$, $(term \{ , term \})_T$ has $(hasPredicate)_S$
 $\Rightarrow N$ has S and T has S
- (c) $(term)_N$ has $(primPossessedNoun)_O$ [such that $(statement)_S$]
[and $(possessed \{ and possessed \})_T$]
 \Rightarrow for some $\mathbf{n}(O)$ ($\mathbf{n}(O)$ is $\bar{O}[N]$ [and S] [and N has T])
- (d) $(term)_N$ has no $(primPossessedNoun)_O$ [such that $(statement)_S$]
 \Rightarrow for every $\mathbf{n}(O)$ (not ($\mathbf{n}(O)$ is $\bar{O}[N]$ [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
 \Downarrow (1a,1c,3a)
A has wife W and salary S such that S is not enough for W
 \Downarrow (8c)
for some W (W is wife of A and
A has salary S such that S is not enough for W)
 \Downarrow (8c)
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.

9. Handling “is a”-predicates. Consider an occurrence of *nounNotion* of the form $((primClassNoun)_O$ 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 $S[\mathbf{n}(O) \rightarrow N]$ is just equal to S . If $\mathbf{n}(O)$ contains a name v then $S[\mathbf{n}(O) \rightarrow N]$ is the result of substitution of N in place of each occurrence of v in S .

The following rules apply to occurrences of *simpleStatement*:

- (a) $(term)_N$, $(term \{ , term \})_T$ is $([not])_C$ $(nounNotion)_O$
 $\Rightarrow N$ is C O and T is C O
(if O does not contain in the attribute any m-predicate applied to $\mathbf{n}(O)$)
- (b) $(terms)_N$ is $([not])_C$ $(primClassNoun)_O$ [such that $(statement)_S$]
 $\Rightarrow (C (N$ is \bar{O} [and $S[\mathbf{n}(O) \rightarrow N]$]))
- (c) $(term)_N$, $(term \{ , term \})_T$ is $(primClassNoun)_O$
 $\Rightarrow N$ is O and T is O
- (d) $(terms)_N$ 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
 \Downarrow (1a,1c,3c)
A,B is not number X such that X is natural
 \Downarrow (9a)
A is not number X such that X is natural
and B is not number X such that X is natural
 \Downarrow (9b)
(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
 \Downarrow (1a,1c,3c)
A,B is not number X such that X is equal
 \Downarrow (9b)
not (A,B is number and A,B is equal)
 \Downarrow (9c)
not (A is number and B is number and A,B is equal)

Also note that the occurrence N in the rule (9b) 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 (1a). Therefore, $\mathbf{n}(O)$ occurs in S only as a subject of a simple statement (due to the rules (3a-3c)) and hence the expression $S[\mathbf{n}(O) \rightarrow N]$ is well-formed.

10. **Eliminating m-predicates.** Given an occurrence S of an m-primitive and an occurrence N of *term*, we denote with $S[N]$ 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 $S[N]$ is `parallel to X`.

The following rules apply to occurrences of *simpleStatement*:

- (a) $(term)_N$ [does] [not] [pairwise] *primVerbM* \Rightarrow syntax error
- (b) $(term)_N$ is [not] [pairwise] *primAdjectiveM* \Rightarrow syntax error
- (c) $(terms)_N$ [does] not ([pairwise])_P (*primVerbM*)_S
 \Rightarrow not (N P S)
- (d) $(terms)_N$ is not ([pairwise])_P (*primAdjectiveM*)_S
 \Rightarrow not (N is P S)
- (e) $(term)_{N_1}$, ... , $(term)_{N_n}$ [does] pairwise (*primVerbM*)_S
 \Rightarrow (N_1 S[N_2]) and ... (N_i S[N_{i+j}]) ... and (N_{n-1} S[N_n])
- (f) $(term)_{N_1}$, ... , $(term)_{N_n}$ [does] (*primVerbM*)_S
 \Rightarrow (N_1 S[N_2]) and ... (N_i S[N_{i+1}]) ... and (N_{n-1} S[N_n])
- (g) $(term)_{N_1}$, ... , $(term)_{N_n}$ is pairwise (*primAdjectiveM*)_S
 \Rightarrow (N_1 is S[N_2]) and ... (N_i is S[N_{i+j}]) ... and (N_{n-1} is S[N_n])
- (h) $(term)_{N_1}$, ... , $(term)_{N_n}$ is (*primAdjectiveM*)_S
 \Rightarrow (N_1 is S[N_2]) and ... (N_i is S[N_{i+1}]) ... and (N_{n-1} is S[N_n])

11. Handling ordinary and symbolic predicates. Here, the rest of atoms with occurrences of *terms* or *symbTerms* are transformed to conjunctions. The following rules apply to occurrences of *simpleStatement* (11a–11b) and *primPredicateSymbol* (11c):

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

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

$$(c) (primPredicateSymbol[(symbTerm)_N , (symbTerms)_T])_S \\ \Rightarrow S[N] \wedge S[T]$$

12. Resolving synonyms. Now, the resulting unit is essentially a first-order formula with atoms of five possible forms (we consider an n -ary class-noun, adjective, or verb as a kind of a predicate symbol of the arity $(n + 1)$ in a first-order signature):

$$\begin{array}{lll} term [\text{does}] primVerb & term \text{ is } primAdjective & specialStatement \\ term \text{ is } primClassNoun & primPredicateSymbol & \end{array}$$

All the terms in this formula are composed of definite nouns and function symbols and hence are plain. The following rule applies to those primitives in the formula 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 introductors are linearly ordered in the ForTheL text.

For example, the introductors

$$\begin{array}{l} [\text{a relation on } D @ \text{ a relation with the domain equal to } D] \\ [U ** V @ \text{ the intersection of } U \text{ with } V] \end{array}$$

induce the following transformation chain:

$$\begin{array}{c} X \text{ is relation on } A ** B ** C \\ \Downarrow (12) \\ X \text{ is a relation with the domain equal to} \\ \text{the intersection of } A \text{ with the intersection of } B \text{ with } C \\ \Downarrow (1-12) \\ X \text{ is relation and domain of } X \text{ is equal to} \\ \text{intersection of } A \text{ with intersection of } B \text{ with } C \end{array}$$

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.

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 (1a,1c)

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 (2a)

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 (3b)

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 (3c)

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 (4a)

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 (5c)

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 (7d,7e)

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 (9b,9c,9d)

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 (10h)

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 (11a)

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 (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

$$\forall A, B ((\text{aPoint}(A) \wedge \text{aPoint}(B) \wedge A \neq B) \supset$$

$$\supset \exists L (\text{aLine}(L) \wedge \text{isStraight}(L) \wedge \text{LiesOn}(A, L) \wedge \text{LiesOn}(B, L) \wedge$$

$$\wedge \forall M ((\text{aLine}(M) \wedge \text{isStraight}(M) \wedge \text{LiesOn}(A, M) \wedge \text{LiesOn}(B, M)) \supset M = L)))$$

Figure 1: Full Translation Chain

2.7 Variable description

Free variables in a statement S are ones free in its formula image: $\mathcal{FV}(S) = \mathcal{FV}(|S|)$. Similarly, bound variables in S are ones bound in $|S|$: $\mathcal{BV}(S) = \mathcal{BV}(|S|)$. Rules of ForTheL prohibit the statements whose image contains nested quantifiers on the same variable. For example, the statement `every number n divides some number n` is ill-formed.

We say that a statement S *describes* a variable $v \in \mathcal{FV}(S)$ whenever S implies that v belongs to a class denoted by some notion. The following chain of mutually recursive definitions gives 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 arg1`;
- I is a primitive adjective, verb or predicate symbol; I is introduced as a synonym; the pattern in the introducer 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 noun notion with positive arguments;
- P is an “is a”-predicate built upon a positive definite term;
- P is built upon a descriptive adjective or verb primitive with positive 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 descriptive predicate symbol such that 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 definition above is valid, since a syntactic primitive introduced as a synonym can occur neither in the target unit nor before the introducer.

For example, after the following introducers

```
[x belongs/belong to y @ x is an element of y]
```

```
[x << y @ 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 predicate symbol `<<` produces a notion symbol since the target statement `x belongs to y` describes the variable `x`.

2.8 Definition statements

There is a special kind of ForTheL statements which is used in definition sections. The syntax of definition statements is described by the following productions:

$$\begin{aligned}
 \text{defStatement} &\rightarrow \text{notionDef} \mid \text{functionDef} \mid \text{predicateDef} \\
 \text{notionDef} &\rightarrow [\text{a} \mid \text{an}] \text{primClassNoun} \text{ is } [\text{a} \mid \text{an}] \text{notion} \\
 &\quad \mid \text{primNotionSymbol} \text{ iff variable is } [\text{a} \mid \text{an}] \text{notion} \\
 \text{functionDef} &\rightarrow \text{functionSym} \text{ is equal to plainTerm} \\
 \text{functionSym} &\rightarrow [\text{the}] \text{primDefiniteNoun} \\
 &\quad \mid \text{primInfixFunctionSymbol} \\
 &\quad \mid \text{primPrefixFunctionSymbol} \\
 &\quad \mid \text{primPostfixFunctionSymbol} \\
 \text{predicateDef} &\rightarrow \text{predicateSym} \text{ iff statement} \\
 \text{predicateSym} &\rightarrow \text{variable is primAdjective} \\
 &\quad \mid \text{variable} \text{ , } \text{variable} \text{ are primAdjectiveM} \\
 &\quad \mid \text{variable primVerb} \\
 &\quad \mid \text{variable} \text{ , } \text{variable} \text{ primVerbM} \\
 &\quad \mid \text{primPredicateSymbol}
 \end{aligned}$$

Each definition statement S has a so-called *main junctor* which may be an equality (as in *functionDef*), an equivalence (as in the second kind of *notionDef* and in *predicateDef*), or a connection *is* (as in the first kind of *notionDef*). The unit to the left of the main junctor is called the *head* of the definition and the unit to the right is called the *body*.

A definition 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 the body occurs also in the head;
- if the primitive in the head unit is introduced as a synonym then the corresponding target is just another primitive with possibly rearranged arguments;
- the primitive in the head unit does not occur in the body (recursive definitions are not supported in the current version of ForTheL); the same is true for the target primitive, if any;
- if S defines a notion then the head notion unit has at most one name;
- if S defines a symbolic notion then the name of the head is the subject in the body.

The formula image of a definition statement is calculated as follows:

$$\begin{aligned} |(primClassNoun)_H \text{ is } (notion)_B| &= \forall v | v \text{ is } \bar{H} \text{ iff } v \text{ is } B | \\ |(primNotionSymbol)_H \text{ iff } ((variable)_v \text{ is } notion)_B| &= \forall v | H \text{ iff } B | \\ |(functionDef)_S| &= |(predicateDef)_S| = |S| \end{aligned}$$

In the first equation, the variable v is $\mathbf{n}(H)$ or a fresh variable if $\mathbf{n}(H)$ is empty. In the second equation, v is $\mathbf{n}(H)$ by the well-formedness condition. Equations in the last line mean that definitions of functions and predicates are treated as ordinary ForTheL statements.

3 Sections of ForTheL

3.1 Top-level sections and sentences

We begin with an example of a ForTheL text (line numbers do not belong to the text).

```

1:    [a set/sets] [an element/elements of x]
2:    [x is in y @ x is an element of y]
3:    [x belongs/belong to y @ x is in y]
4:    [a subset/subsets of x] [x is empty]

5:    Definition DefSubset. Let S be a set.
6:      A subset of S is a set X such that
7:        every element of X belongs to S.

8:    Definition DefEmpty. Let S be a set.
9:      S is empty iff S has no elements.

10:   Axiom ExEmpty. There exists an empty set.

11:   Proposition. Let S be a set.
12:     S is a subset of every set iff S is empty.
13:   Proof.
14:     First let S be a subset of every set.
15:     We can show that S is empty.
16:       Let z belong to S.
17:       Take an empty set E.
18:       z is an element of E.
19:       We have a contradiction.
20:     end.
21:   end.
22:   Now assume that S is empty. Let T be a set.
23:     Every element of S is in T (by DefEmpty).
24:     Hence S is a subset of T (by DefSubset).
25:   end.
26:   qed.

```

A ForTheL text is a sequence of introductors and *top-level sections*. Top-level sections are *axioms*, *definitions* and *propositions*. There are four top-level sections in the text above, namely, a definition of the subset relation (lines 5–7), a definition of the emptiness property (lines 8–9), an axiom (line 10), and a proposition with a proof (lines 11–26).

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 23 and 24).

$$\begin{aligned} \textit{text} &\rightarrow \{ \textit{axiom} \mid \textit{definition} \mid \textit{proposition} \mid \textit{introductor} \} \\ \textit{axiom} &\rightarrow \textit{axmHeader} \{ \textit{assume} \} \textit{axmAfirm} \\ \textit{axmHeader} &\rightarrow \textbf{Axiom} [\textit{label}] . \\ \textit{definition} &\rightarrow \textit{defHeader} \{ \textit{assume} \} \textit{defAfirm} \\ \textit{defHeader} &\rightarrow \textbf{Definition} [\textit{label}] . \\ \textit{proposition} &\rightarrow \textit{prpHeader} \{ \textit{assume} \} \textit{prpAfirm} \\ \textit{prpHeader} &\rightarrow (\textbf{Proposition} \mid \textbf{Theorem} \mid \textbf{Lemma} \mid \textbf{Corollary}) [\textit{label}] . \\ \textit{label} &\rightarrow \textit{word} \end{aligned}$$

Inside top-level sections we meet *sentences*. Some of them are *assumptions* (lines 5,8,11,14,16,22) which may declare variables; some of them are *selections* (line 17) which may declare variables, too; the rest are *affirmations*. Any top-level section consists of zero or more assumptions followed by an affirmation of an appropriate kind.

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

$$\begin{aligned} \textit{assume} &\rightarrow \textit{asmPrefix} \textit{statement} . \\ \textit{asmPrefix} &\rightarrow \textbf{let} \mid [\textbf{let us} \mid \textbf{we can}] (\textbf{assume} \mid \textbf{suppose}) [\textbf{that}] \\ \textit{select} &\rightarrow \textit{selPrefix} \textit{notions} [\textit{reference}] . \\ \textit{selPrefix} &\rightarrow [\textbf{then} \mid \textbf{therefore} \mid \textbf{hence}] [\textbf{let us} \mid \textbf{we can}] (\textbf{take} \mid \textbf{choose}) \\ \textit{axmAfirm} &\rightarrow \textit{statement} . \\ \textit{defAfirm} &\rightarrow \textit{defStatement} . \\ \textit{prpAfirm} &\rightarrow \textit{affPrefix} \textit{statement} [\textit{reference}] . [\textit{prfHeader} \textit{proof}] \\ &\mid \textit{prfPrefix} \textit{statement} [\textit{reference}] . \textit{proof} \\ \textit{affPrefix} &\rightarrow [\textbf{then} \mid \textbf{therefore} \mid \textbf{hence}] \\ \textit{prfHeader} &\rightarrow \textbf{proof} [\textbf{by method}] . \mid \textbf{indeed} \\ \textit{prfPrefix} &\rightarrow [\textbf{let us} \mid \textbf{we can}] (\textbf{prove} \mid \textbf{show}) [\textbf{by method}] [\textbf{that}] \end{aligned}$$

method \rightarrow contradiction | case analysis | induction [on *plainTerm*]
reference \rightarrow (by label { , label })

3.2 Proof sections

In a correct text, every affirmation (except those ones inside axioms and definitions) 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 section* which will be an implicit logical predecessor for the affirmation. In the text above, there are two proof sections: one for the affirmation in the line 12 (lines 13–26) and one for the affirmation in the line 15 (lines 16–20).

Proof sections in ForTheL are composed of assumptions, selections, affirmations (possibly, with proofs of their own), and low-level sections like proof cases or proof blocks. The main proof in the text above is composed of two proof blocks (lines 14–21 and 22–25).

Proof blocks are used just to structure the proof: they determine the scope of assumptions and variable declarations. Proof blocks may be put anywhere inside a proof section. Case analysis is supposed to conclude a proof section after some preliminary reasoning. Case sections are listed one by one; every case section is headed with a statement that expresses the case hypothesis.

proof \rightarrow [{ *prfBody* } *prfLast*] *qed*
qed \rightarrow end. | qed. | obvious. | trivial.
prfBody \rightarrow assume | select | *prpAffirm* | block
prfLast \rightarrow *prpAffirm* | block | case { case }
block \rightarrow *blkHeader proof*
blkHeader \rightarrow Block [*label*] . | now | first | second | ...
case \rightarrow Case *statement* [*reference*] . *proof*

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 section. Conversely, the mention “by induction” is necessary as it instructs the verifier to formulate the *induction hypothesis* and to insert it into the proof section in the appropriate place.

Formalization of induction reasoning in ForTheL is based on the following *general induction principle*:

$$\forall \vec{x} (\text{IH}(\vec{x}) \supset F[\vec{x}]) \supset \forall \vec{x} F[\vec{x}]$$

where $\text{IH}(\vec{x}) = \forall \vec{y} (t[\vec{y}] \prec t[\vec{x}] \supset F[\vec{y}])$, t is a certain first-order term, and \prec is a well-founded ordering. This principle can be verbalized as follows: *when proving that a property P holds for some arbitrary but fixed \vec{x} , one can assume that this property holds for all \vec{y} that are less than \vec{x} with respect to some well-founded ordering.* The *induction term* t is used to “gather” a tuple of values in one value.

So, the proofs by induction are read and verified in assumption of the corresponding induction hypothesis. This hypothesis is formulated automatically and becomes an additional logical predecessor for the induction argument.

A proof by induction must have the thesis (3.4) of the form **for every** $(notion)_{O_1}$... **for every** $(notion)_{O_n}$ *statement*. If the induction term is not mentioned explicitly, it is assumed by default to be the first variable in $\mathbf{n}(O_1)$. All the variables in the induction term should belong to $\mathbf{n}(O_1) \cup \dots \cup \mathbf{n}(O_n)$ or be declared before the proof. The induction hypothesis is obtained from the thesis statement as follows:

$$\begin{aligned} & \text{for every } (notion)_{O_1} \dots \text{for every } (notion)_{O_n} (statement)_S \\ \Rightarrow & \text{for every } O_1\sigma \dots \text{for every } O_n\sigma \text{ if } t\sigma \text{ -<- } t \text{ then } S\sigma \end{aligned}$$

where t is the induction term and σ is a substitution that renames the variables in t to some fresh variables. The induction hypothesis is implicitly inserted into the proof during the normalization process (3.4).

In first-order logic we cannot give a finite axiomatization of well-foundedness. Therefore ForTheL provides a preintroduced binary predicate symbol `-<-` which is used as an abstract well-founded relation in induction arguments. It is left to the author to write down the axioms defining the concrete properties of the relation `-<-`. For example, to simulate the natural induction in ForTheL, the following axioms are sufficient (on condition that traditional Peano axioms are given, too):

Axiom ZeroOrSucc. For every natural number N $N = 0$
or there exists a natural number M such that $N = \text{succ } M$.

Axiom IndOrdering. For every natural number N $N \text{ -<- succ } N$.

Indeed, consider a simple lemma which we want to prove by induction on n :

Lemma. For all natural numbers m, n $(\text{succ } m) + n = \text{succ } (m + n)$.

The induction hypothesis (IH) for this lemma is the following statement:

for all natural numbers x, y if $y \text{ -<- } n$ then $(\text{succ } x) + y = \text{succ } (x + y)$

Now, let us write the proof section:

```
Proof by induction on n.
Let m,n be natural numbers.
## the scope of IH starts here
Case n = 0. Obvious.                                ## proved by Peano axioms
Case n != 0.
  Take a natural number x such that succ x = n.    ## by ZeroOrSucc
  (succ m) + n = succ ((succ m) + x).             ## by Peano axioms
  (succ m) + x = succ (m + x).                     ## by IH and IndOrdering
  succ (m + x) = m + n.                             ## by Peano axioms
  Hence (succ m) + n = succ (m + n).              ## from the previous
end.
qed.
```

The axiom `ZeroOrSucc` is needed to split the induction argument to the base and the step. The axiom `IndOrdering` is needed to prove that $x \text{ -<- } n$ so that the induction hypothesis could be applied. Note that the ordering properties of `-<-` are not needed for the proof.

3.3 Precedence and declaration

We set the boundaries of sections in a text \mathbb{T} 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, an affirmation supplied with a proof ends where the proof section ends, not with the sentence-ending dot.

We say that a section A *covers* a section B (B is *inside* A) if B lies within the boundaries of A and B is not A . Thus, an affirmation supplied with a proof covers this proof section and all the sections inside the proof. We call A a *subsection* of B if B is the smallest section covering A . For example, any proof block or proof case has exactly one member, namely, the proof section attached to that block or case. We say that A *textually precedes* B if B starts after the end of A in the text.

A section A *logically precedes* a section B if A textually precedes B and no section covering A textually precedes B . We denote the set of logical predecessors of a section A in a text \mathbb{T} with $\mathbf{LP}_{\mathbb{T}}(A)$.

In the example text from (3.1), the affirmation in the line 15 has the following logical predecessors: the both definitions, the axiom **ExEmpty**, and the assumptions in the lines 11 and 14. The affirmation in the line 12 is not a predecessor of the affirmation in question.

Assumptions and selections in ForTheL may be used to declare variables. We determine the set of declared variables according to the syntactic form of an assumption or a selection. A variable v is declared by a selection if there is a notion O in the selected notion chain such that $v \in \mathbf{n}(O)$. Denoting the set of declared variables with $\mathbf{Decl}_{\mathbb{T}}(\cdot)$, we have:

$$\begin{aligned} \mathbf{Decl}_{\mathbb{T}}(\mathit{selPrefix} (\mathit{notions})_N [\mathit{reference}] .) &= \mathbf{Decl}_{\mathbb{T}}(N) \\ \mathbf{Decl}_{\mathbb{T}}([\mathbf{a} \mid \mathbf{an}] (\mathit{notion})_O [(, \mid \mathbf{and}) (\mathit{notions})_N]) &= \mathbf{n}(O) [\cup \mathbf{Decl}_{\mathbb{T}}(N)] \end{aligned}$$

An assumption declares exactly the variables described by the assumed statement:

$$\mathbf{Decl}_{\mathbb{T}}(\mathit{asmPrefix} (\mathit{statement})_S .) = \{v \mid S \text{ describes } v\}$$

For those sections A which are not assumptions or selections, $\mathbf{Decl}_{\mathbb{T}}(A) = \emptyset$.

Variables declared in a sentence A can occur free in A and in the logical successors of A . It is allowed for an assumption to declare variables which were previously declared; subsequent declarations do not cancel but just specify the precedent ones. Conversely, variables declared by a selection should not be declared by any logical predecessor of that selection. We denote the set of variables declared by the logical predecessors of a section A with $\overline{\mathbf{Decl}}_{\mathbb{T}}(A)$. Such variables will be called *predeclared* in A .

3.4 Proof normalization

We provide a set of transformation rules that normalize proofs in a ForTheL text: eliminate special statements **thesis** and **contrary**, complete proof sections, translate proof cases into combination of proofs and blocks, insert induction hypotheses at the appropriate places. This transformation applied to a ForTheL text \mathbb{T} results in a *normalized* ForTheL text \mathbb{T}° .

Handling the thesis. Every proposition A in \mathbb{T} is replaced with the proposition $\langle A \rangle$, defined with the following equations.

$$\langle (prpHeader)_H (\{ assume \})_Q (prpAffirm)_A \rangle = H Q \langle A \rangle_{\text{contradiction}}$$

We pass by the header and the leading assumptions and normalize the affirmation of the proposition. Sentences and low-level sections are normalized with respect to the *thesis statement* (passed in the subscript). Roughly speaking, the thesis for a given section is the statement which was being proved when the section started. The initial thesis is **contradiction** (meaning that so far we do not prove any statement).

$$\begin{aligned} \langle affPrefix (statement)_S ([reference])_R . [(prfHeader)_H (proof)_P] \rangle_T &= \\ &= S[T] R . [H \langle P \rangle_{S[T]^\dagger}^\top] \\ \langle (prfPrefix)_H (statement)_S ([reference])_R . (proof)_P \rangle_T &= \\ &= H S[T] R . \langle P \rangle_{S[T]^\dagger}^\top \end{aligned}$$

Here and below, $S[T]$ is the unit S where the occurrences of **thesis** and **contrary** are replaced with the statement T and with its negation, respectively. The casual normal form of the statement being proved becomes the thesis for the supplied proof section (passed in the subscript). Also, the supplied proof is supposed to be *motivated* by default (\top passed in the superscript).

$$\begin{aligned} \langle asmPrefix (statement)_S . \rangle_T &= \text{assume } S[T] . \\ \langle selPrefix (notions)_S ([reference])_R . \rangle_T &= \text{take } S[T] R . \\ \langle (blkHeader)_H (proof)_P \rangle_T &= H \langle P \rangle_T^\perp \\ \langle \text{Case } (statement)_S ([reference])_R . (proof)_P \rangle_T &= \\ &= \langle (\text{if } S[\text{contradiction}] \text{ then thesis}) R . \\ &\quad \text{proof. assume } S[\text{contradiction}] . P \rangle_T \end{aligned}$$

The thesis statement of a proof block is passed inside the block. Also, the proof blocks are unmotivated by default (\perp in the superscript). A case section is translated to an affirmation of the thesis statement in the corresponding case; this affirmation is supplied with a proof. The special statements **thesis** and **contrary** should not occur in the case hypothesis.

$$\begin{aligned} \langle (assume)_A (proof)_P \rangle_T^\top &= \langle (\text{Block. } A P) \text{ end.} \rangle_T^\top && (\text{if } T \setminus A = T) \\ \langle (assume)_A (proof)_P \rangle_T^s &= \langle A \rangle_T \langle P \rangle_{T \setminus A}^s && (\text{otherwise}) \end{aligned}$$

In a motivated proof, every assumption is in a certain way related to the thesis statement and provides a *reduced thesis* for the subsequent proof. Above, we denote this reduction with $T \setminus A$. If we encounter an assumption that does not reduce the thesis then we consider the rest of the proof as unmotivated and encapsulate it into a block section. Conversely, if the encountered assumption is relevant ($T \setminus A \neq T$) or we are already inside a nonmotivated proof section, we just proceed with the new thesis.

Given a thesis T and an assumption A with the statement S , we define the new thesis $T \setminus A$ as follows. Note that if T is in the casual normal form then $T \setminus A$ also is.

- if T is of the form (if $S[T]$ then R) then $T \setminus A = R$;
- if $S[T]$ is of the form (not T) then $T \setminus A = \text{contradiction}$. In particular, this is the case when A is of the form **assume the contrary.**;
- if $T = (\text{for every } (notion)_{O_1} \dots \text{for every } (notion)_{O_n} (statement)_R)$, and the arguments in the primitive notions in O_1, \dots, O_n are plain terms (this is the case when T is in the casual normal form), and $S[T]$ is of the form ($\mathbf{n}(O_i)$ is \bar{O}_i), and $\mathbf{n}(O_i)$ is disjoint with $\overline{\mathbf{Decl}}_{\mathbb{T}}(A)$ — then $T \setminus A$ is the statement

for every $O_1 \dots$ for every O_{i-1} for every $O_{i+1} \dots$ for every O_n R

- in any other case, $T \setminus A = T$.

Selections, affirmations, and proof blocks do not influence the thesis:

$$\begin{aligned} \langle (select | prpAffirm | block)_A (proof)_P \rangle_T^s &= \langle A \rangle_T \langle P \rangle_T^s \\ \langle (case)_{C_1} \dots (case)_{C_n} qed \rangle_T^s &= (S_i \text{ denotes the case hypothesis in the header of } C_i) \\ &= T. \text{ proof. } \langle C_1 \rangle_T \dots \langle C_n \rangle_T \langle S_1 \text{ or } \dots \text{ or } S_n. \rangle_{\text{contradiction}} \text{ end. end.} \end{aligned}$$

The sequence of case sections finishing the proof section is transformed to the affirmation of the thesis statement supplied with a formal proof by case analysis. Recall that the special statements **thesis** and **contrary** should not occur in the case hypotheses.

$$\langle qed \rangle_T^\top = T. \text{ end.} \qquad \langle qed \rangle_T^\perp = \text{end.}$$

When we reach the end of a motivated proof, it is sufficient to prove the current thesis to prove the initial goal. Therefore, we insert the thesis statement as the last affirmation in the proof section. When we reach the end of a block (unmotivated proof), there is no intended goal to prove, so we just leave.

Shifting references. In the text \mathbb{T}' resulted from the previous transformation, every proof section supplied for an affirmation ends with an affirmation, too (it may be not the case for proof sections supplied for blocks). Moreover, this concluding affirmation does not have a reference.

For every affirmation A which is supplied with a reference and a proof section P , we move the reference from A to the affirmation B that concludes P . If B has a proof, too, we repeat the transformation.

Setting induction hypotheses. In the text \mathbb{T}'' resulted from the previous transformation, we consider every affirmation A which is supplied with a proof by induction on a plain term t (given explicitly or not). As we said above (3.2), A should affirm the statement whose casual normal form is of the form

for every $(notion)_{O_1} \dots$ for every $(notion)_{O_n} (statement)_S$

and the variables of t should belong to $(\mathbf{n}(O_1) \cup \dots \cup \mathbf{n}(O_n)) \cup \overline{\mathbf{Decl}}_{\mathbb{T}''}(A)$.

Let I be the corresponding induction hypothesis. Now, if all the variables of t declared in the proof are declared by assumptions, and if there are member subsections in the

proof such that all the variables of t are predeclared in them, then we take the first such a member subsection and insert the sentence **assume that** I . immediately before it.

The transformation is finished and we obtain a normalized text \mathbb{T}° . Let us consider as an example the lemma from (3.2). The normalized proof for it will look as follows:

Lemma. For all natural numbers m, n $(\text{succ } m) + n = \text{succ } (m + n)$.

Proof by induction on n .

Let m, n be natural numbers.

Assume that for all natural numbers x, y

if $y \leftarrow n$ then $(\text{succ } x) + y = \text{succ } (x + y)$.

$(\text{succ } m) + n = \text{succ } (m + n)$.

proof.

if $n = 0$ then $(\text{succ } m) + n = \text{succ } (m + n)$.

proof.

assume $n = 0$.

$(\text{succ } m) + n = \text{succ } (m + n)$.

end.

if $n \neq 0$ then $(\text{succ } m) + n = \text{succ } (m + n)$.

proof.

assume $n \neq 0$.

Take a natural number x such that $\text{succ } x = n$.

$(\text{succ } m) + n = \text{succ } ((\text{succ } m) + x)$.

$(\text{succ } m) + x = \text{succ } (m + x)$.

$\text{succ } (m + x) = m + n$.

Hence $(\text{succ } m) + n = \text{succ } (m + n)$.

$(\text{succ } m) + n = \text{succ } (m + n)$.

end.

$n = 0$ or $n \neq 0$.

end.

end.

If we omit the base case in the proof of the initial lemma:

Lemma. For all natural numbers m, n $(\text{succ } m) + n = \text{succ } (m + n)$.

Proof by induction on n .

Let m, n be natural numbers.

Assume that $n \neq 0$.

Take a natural number x such that $\text{succ } x = n$.

$(\text{succ } m) + n = \text{succ } ((\text{succ } m) + x)$.

$(\text{succ } m) + x = \text{succ } (m + x)$.

$\text{succ } (m + x) = m + n$.

Hence $(\text{succ } m) + n = \text{succ } (m + n)$.

qed.

the resulting normalized text will look as follows:

Lemma. For all natural numbers m, n $(\text{succ } m) + n = \text{succ } (m + n)$.

Proof by induction on n .

Let m, n be natural numbers.
 Assume that for all natural numbers x, y
 if $y \leftarrow n$ then $(\text{succ } x) + y = \text{succ } (x + y)$.
 Block.
 Assume that $n \neq 0$.
 Take a natural number x such that $\text{succ } x = n$.
 $(\text{succ } m) + n = \text{succ } ((\text{succ } m) + x)$.
 $(\text{succ } m) + x = \text{succ } (m + x)$.
 $\text{succ } (m + x) = m + n$.
 Hence $(\text{succ } m) + n = \text{succ } (m + n)$.
 end.
 $(\text{succ } m) + n = \text{succ } (m + n)$.
 end.

3.5 Formula image of sections

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

1. Assumptions: $|(\text{asmPrefix } (\text{statement})_S \text{ .})_P| = |S|$
2. Selections: $|(\text{selPrefix } (\text{notions})_N \text{ .})| = F'$

The formula F' is obtained as follows. Let F be the formula image of the statement `there exist N` . For every notion O in the chain N such that $\mathbf{n}(O)$ is nonempty, we remove the corresponding existential quantifier on $\mathbf{n}(O)$ from F , thus making $\mathbf{n}(O)$ free. The resulting formula is F' . Consider the following examples:

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

3. Affirmations:

$|(\text{statement})_S \text{ .}| = |(\text{defStatement})_S \text{ .}| = |S|$
 $|(\text{affPrefix } (\text{statement})_S \text{ [reference] . [prfHeader proof]})| = |S|$
 $|(\text{prfPrefix } (\text{statement})_S \text{ [reference] . proof})| = |S|$

4. Proofs:

$|(\text{assume})_A (\text{proof})_P| = \forall \vec{v}_A (|A| \supset |P|)$ $|(\text{prfAffirm})_A (\text{proof})_P| = |A| \wedge |P|$
 $|(\text{select})_A (\text{proof})_P| = \exists \vec{v}_A (|A| \wedge |P|)$ $|(\text{block})_A (\text{proof})_P| = |A| \wedge |P|$
 Here, $\vec{v}_A = \mathbf{Decl}_{\mathbb{T}^\circ}(A) \setminus \overline{\mathbf{Decl}_{\mathbb{T}^\circ}(A)}$ $|qed| = \text{verum}$

5. Proof blocks: $|(\text{blkHeader } (\text{proof})_P)| = |P|$

6. Top-level sections:

$$\begin{aligned}
& | \text{axmHeader } (\text{assume})_{A_1} \dots (\text{assume})_{A_n} (\text{axmAfirm})_B | = \\
& = | \text{defHeader } (\text{assume})_{A_1} \dots (\text{assume})_{A_n} (\text{defAfirm})_B | = \\
& = | \text{prpHeader } (\text{assume})_{A_1} \dots (\text{assume})_{A_n} (\text{prpAfirm})_B | = \\
& = \forall \vec{v}_{A_1} (|A_1| \supset \dots \supset \forall \vec{v}_{A_n} (|A_n| \supset |B|) \dots)
\end{aligned}$$

Here, $\vec{v}_{A_i} = \mathbf{Decl}_{\mathbb{T}^\circ}(A_i) \setminus \overline{\mathbf{Decl}}_{\mathbb{T}^\circ}(A_i)$.

Note that a proof section supplied for an affirmation 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 given proof section supports that indented meaning.

As before, we define the set of free (bound) variables of a section A as the set of free (bound) variables of the formula image of A : $\mathcal{FV}(A) = \mathcal{FV}(|A|)$ and $\mathcal{BV}(A) = \mathcal{BV}(|A|)$.

3.6 Well-formedness and correctness

First, we introduce the corresponding notions for normalized texts. A section A in a normalized ForTheL text \mathbb{T}° is *well-formed* if the following conditions hold:

- $\mathcal{FV}(A) \subseteq \mathbf{Decl}_{\mathbb{T}^\circ}(A) \cup \overline{\mathbf{Decl}}_{\mathbb{T}^\circ}(A)$
- $\mathcal{BV}(A) \cap (\mathbf{Decl}_{\mathbb{T}^\circ}(A) \cup \overline{\mathbf{Decl}}_{\mathbb{T}^\circ}(A)) = \emptyset$
- if A is a selection then $\mathbf{Decl}_{\mathbb{T}^\circ}(A) \cap \overline{\mathbf{Decl}}_{\mathbb{T}^\circ}(A) = \emptyset$
- if A is a definition affirmation then $\mathcal{FV}(A) = \overline{\mathbf{Decl}}_{\mathbb{T}^\circ}(A)$
- if A is a definition affirmation then the defined primitive does not occur in $\mathbf{LP}_{\mathbb{T}^\circ}(A)$; the same is true for the target of the defined primitive, if any;
- if A is a proof by induction then A has the thesis of the form **for every** $(\text{notion})_{O_1}$ **and** \dots **and every** $(\text{notion})_{O_n}$ *statement*; if the induction term t is explicitly given then $\mathcal{FV}(t) \subseteq \mathbf{n}(O_1) \cup \dots \cup \mathbf{n}(O_n)$.

Note that purely composed sections such as top-level sections, proof blocks, and non-induction proofs are well-formed whenever all their members are.

A section A in a normalized ForTheL text \mathbb{T}° is *correct* if the following holds:

- A is well-formed and each member of A is correct;
- if A is a selection with the notion chain N then $\mathbf{LP}_{\mathbb{T}^\circ}(A) \models \text{there exists } N$.
- if A is an affirmation inside a proposition section and A is not supplied with a proof then $\mathbf{LP}_{\mathbb{T}^\circ}(A) \models A$

Here, $A_1, \dots, A_n \models A$ stands for $|A_1|, \dots, |A_n| \models |A|$. Note that we do not check affirmations supplied with proof sections: their correctness ($\mathbf{LP}_{\mathbb{T}^\circ}(A) \models A$) is guaranteed by the correctness of the proof — it is the important property of normalization.

A normalized ForTheL text is *correct* if every top-level section in it is correct.

A ForTheL text \mathbb{T} is *correct* if the normalized text \mathbb{T}° is correct.

Index

- $\mathcal{BV}(S)$ (bound variables), 22, 33
- $\text{Decl}_{\mathbb{T}}(A)$ (declaration), 28
- $\overline{\text{Decl}}_{\mathbb{T}}(A)$ (predeclaration), 28
- $\mathcal{FV}(S)$ (free variables), 22, 33
- $\text{LP}_{\mathbb{T}}(A)$ (logical predecessors), 28
- $\mathbf{n}(O)$ (name list), 14
- \mathbf{N} (notion nonterminals), 14
- \bar{O} (unnaming), 16
- $|S|$ (formula image), 14, 32
- S^{\dagger} (casual normal form), 15
- \mathbb{T}° (normalized text), 28
- $T \setminus A$ (thesis reduction), 29

- affirmation sentence, 25
- assumption sentence, 25
- attribute unit, 8, 15
- axiom section, 25

- block section, 26

- case section, 26, 29
- casual normal form, 15, 29
- correctness
 - of section, 33
 - of text, 33
- declaration, 28
 - precedent, 28
- definition section, 25

- formula image
 - of definition statement, 24
 - of section, 32
 - of statement, 14

- induction proof, 26, 30
- introducer, 3, 5

- label, 25
- lexeme, 2, 3
- logical precedence, 28

- m-predicate unit, 11, 18, 19

- nonterminal, 3
 - affPrefix*, 25
 - alpha*, 3
 - alphanum*, 3
 - andChain*, 13
 - asmPrefix*, 25
 - assume*, 25
 - atomicStatement*, 12
 - axiom*, 25
 - axmAffirm*, 25
 - axmHeader*, 25
 - block*, 26
 - capital*, 3
 - case*, 26
 - chainStatement*, 13
 - comment*, 3
 - defAffirm*, 25
 - defHeader*, 25
 - definiteTerm*, 9
 - definition*, 25
 - defStatement*, 23
 - doesPredicate*, 10
 - functionDef*, 23
 - functionSym*, 23
 - hasPredicate*, 10
 - headStatement*, 13
 - introducer*, 5
 - is_aPredicate*, 10
 - isPredicate*, 10
 - label*, 25
 - leftAttribute*, 8
 - lexeme*, 3
 - method*, 25
 - names*, 7
 - newline*, 3
 - notion*, 8
 - notionDef*, 23
 - notions*, 12
 - notionSymbol*, 7
 - nounNotion*, 11
 - numeric*, 3
 - orChain*, 13
 - pattern*, 6
 - plainTerm*, 9
 - possessed*, 11
 - postfixSymbTerm*, 9
 - predicateDef*, 23

- predicateSym*, 23
- prefixSymbTerm*, 9
- prfBody*, 26
- prfHeader*, 25
- prfLast*, 26
- prfPrefix*, 25
- primAdjective*, 10
- primAdjectiveM*, 11
- primaryNotion*, 7
- primClassNoun*, 7
- primDefiniteNoun*, 9
- primInfixFunctionSymbol*, 9
- primNotionSymbol*, 7
- primPlainNoun*, 9
- primPossessedNoun*, 11
- primPostfixFunctionSymbol*, 9
- primPredicateSymbol*, 12
- primPrefixFunctionSymbol*, 9
- primSimpleAdjective*, 8
- primSimpleAdjectiveM*, 8
- primVerb*, 10
- primVerbM*, 11
- proof*, 26
- proposition*, 25
- prpAffirm*, 25
- prpHeader*, 25
- quantifiedNotion*, 8
- realDefiniteTerm*, 9
- realPlainTerm*, 9
- realQuantifiedNotion*, 8
- reference*, 25
- rightAttribute*, 8
- select*, 25
- selPrefix*, 25
- simpleStatement*, 12
- small*, 3
- space*, 3
- specialStatement*, 13
- statement*, 13
- symbol*, 3
- symbPattern*, 6
- symbStatement*, 12
- symbTerm*, 9
- symbTerms*, 12
- symbToken*, 6
- term*, 8
- terms*, 12
- text*, 25
- thereIsStatement*, 12
- token*, 6
- tokens*, 6
- variable*, 3
- whitespace*, 3
- whitetoken*, 3
- word*, 3
- notion unit, 4, 7
 - primary, 7
 - quantified, 8, 14, 17
- occurrence, 13
 - native, 14
 - proper, 14
- pattern, 5
- predicate unit, 4, 10
 - “has”-, 10, 17
 - “is a”-, 10, 18
 - descriptive, 22
 - multisubject, 11, 18, 19
 - primary, 10
- proof methods, 26
- proof section, 26
 - motivated, 29
- proposition section, 25
- reference, 25, 30
- section, 2, 24
 - axiom, 25
 - boundaries, 28
 - definition, 25
 - low-level, 26
 - member of, 28
 - proof, 26
 - proof block, 26
 - proof case, 26, 29
 - proposition, 25
 - sentence, 2, 25
 - top-level, 25
- selection sentence, 25
- sentence, 2, 25
 - affirmation, 25
 - assumption, 25
 - selection, 25
- statement unit, 4, 12
 - “there is”-, 12, 16

- atomic, 12
- conditional, 13
- definition, 23
- descriptive, 22
- quantified, 13, 17
- simple, 12
- special, 13
- symbolic, 12
- thesis, 13, 15, 29
- synonym, 5, 20
- syntactic primitive, 2, 5
 - adjective, 5, 10
 - base, 5
 - class-noun, 5, 7
 - definite noun, 5, 9
 - descriptive, 22
 - function symbol, 5, 9
 - precedence, 8
 - introduction, 5
 - m-adjective, 10
 - m-verb, 10
 - multisubject, 10, 19
 - adjective, 10
 - simple adjective, 8
 - verb, 10
 - notion symbol, 7
 - possessed noun, 11, 17
 - predicate symbol, 5, 12
 - preintroduced, 10, 12, 27
 - simple adjective, 8
 - simple m-adjective, 8
 - verb, 5, 10
- target unit, 5, 20
- term unit, 4, 8
 - definite, 8
 - plain, 9
 - positive, 22
- terminal, 3
 - \leftarrow , 27
 - \Rightarrow , \Leftrightarrow , 12
 - $=$, \neq , 12
 - $[]$, $\textcircled{}$, 5
 - \setminus , \wedge , 12
 - $\#$, 3
 - all, any, 8
 - and, 8, 10, 12, 13
 - assume, 25
 - Axiom, 25
 - be, is, are, 10, 23
 - Block, 26
 - by, 25
 - Case, 26
 - case analysis, 25
 - choose, 25
 - contradiction, 13, 25
 - contrary, 13, 29
 - Corollary, 25
 - Definition, 25
 - do, does, 10
 - each, every, 8
 - end, 26
 - equal to, 10, 23
 - exists, 12
 - for, 13
 - forall, 12
 - have, has, 10
 - having, 10
 - iff, 13, 23
 - if then, 13
 - indeed, 25
 - induction, 25
 - it is wrong that, 13
 - Lemma, 25
 - let, 25
 - no, 8, 12
 - not, 10, 12
 - now, first, second, 26
 - obvious, 26
 - of, 10
 - or, 13
 - pairwise, 10
 - proof, 25
 - Proposition, 25
 - prove, 25
 - qed, 26
 - show, 25
 - some, 8
 - such that, 8
 - suppose, 25
 - take, 25
 - that, 8
 - Theorem, 25
 - there exists, 12
 - thesis, 13, 29

- trivial, 26
- with, 10
- text, 3, 25
 - normalized, 28
- textual precedence, 28
- thesis reduction, 29
- thesis statement, 13, 15, 29
- token, 6

- unit, 2, 4
 - attribute, 8, 15
 - m-predicate, 11
 - notion, 4, 7
 - predicate, 4, 10
 - statement, 4, 12
 - target, 5, 20
 - term, 4, 8

- variable, 3
 - bound, 22, 33
 - declared, 28
 - free, 22, 33
- vocabulary, 5

- well-formedness
 - of definition statement, 23
 - of section, 33
 - of statement, 22