SAPIENZA
UNIVERSITÀ DI ROMA

A simplified framework for first-order languages and its formalization in Mizar

Scuola Dottorale in Scienze Astronomiche, Chimiche, Fisiche, Matematiche e della Terra "Vito Volterra"

Dottorato di Ricerca in Matematica – XXII Ciclo

Candidate

Marco Caminati

ID number 1146957

Thesis Advisor

Prof. Giuseppe Rosolini

A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Mathematics

30/11/2011

Thesis not yet defended

Marco Caminati. *A simplified framework for first-order languages and its formalization in Mizar.*
Ph.D. thesis. Sapienza – University of Rome

WEBSITE: http://www.mat.uniroma1.it/people/caminati
EMAIL: caminati@mat.uniroma1.it

Acknowledgments

Support and guidance from my advisor, Prof. Giuseppe Rosolini, have been invaluable.
I am grateful to Prof. Claudio Bernardi for helpful advice.
I am indebted to Prof. Peter Koepke, who encouraged me with his interest in my research and gave me the opportunity to meet other people working in my area through his gracious hospitality.
I had the luck of making the acquaintance of Flavia Mascioli and Enrico Rogora, among the friendliest and most supportive people I met in my department.
My neighborly fellow graduate students Stefano, Fabio, Linda, Paolo and Andrea supplied good company and interesting discussion.
Finally, I thank rms for being the zealot he is, which I think made this thesis, and the world, better. Through him I wish to thank every individual who ever contributed to free information.

Contents

Introduction

The axioms of set theory in first-order logic, together with a choice of a deductive system, form the foundations on which most mathematicians set their research work. Thus it is quite natural that also logicians study formalizations of first-order logic and of deductive systems in those same foundations. It appears rather surprising that formalizations of deductive systems are still missing.

One possible explanation for the lack of a mathematically-flavored treatment of a foundational block of such kind is that its fundamental role in the mechanization of mathematics makes research efforts focus on it as a computational tool and divert them from rather viewing it as an object of mathematical study in its own sake. The adjective "mathematical" in the last sentence is crucial: indeed, deductive systems are subject to intense study by proof-theorists, but mainly from a computational point of view and with methods typical of computer science. While this is certainly critical for the mechanization, it yields as a consequence that deductive systems are, for instance, usually expressed in languages far from set theory (or any other language a mathematician may be accustomed to).

For example, consider the sequent calculus. Its rules are usually displayed through diagrams like

$$\frac{\Gamma \qquad \psi}{\Gamma \quad \varphi \quad \psi}.$$

Such diagrams serve well the goals of mechanization, because generally they are readily rendered into concrete computer languages adopted by many proof assistants; on the other hand, they are far from being a definition of the rule itself according to set theory. Therefore there is a gap between the mechanization of mathematics and the formalization in (one of the most standard) foundations of mathematics.[1]

Indeed, considering the way standard expositions of sequent calculus or natural deduction define what a derivation or a proof is (often such notions are merely introduced with examples, as in [EFT84] (section IV.1), [CH07] (chapter 2)), it is invariably found that it pivots on some notion describing what an atomic step in a derivation is, and that this latter notion is not rigorous, from a strictly formal point of view, because it is based on the diagrams just discussed, rather than on a set-theoretical description of each single rule (in the quotations below, we emphasize the words referring to entities lacking a rigorous symbolic definition):

> ... the labels at the immediate successors of a node ν are the *premises*

[1] In alternative formal systems there are rigorous definition of deductive systems; see for example [DG10], section 3 and [MVW98], section 2.

> of a rule *application*, the label at ν the *conclusion*.
>
> [TS96], section 1.3.

> By a derivation of Y from X in the system is meant a finite sequence of lines [...] such that for each $i < n$, the line X_{i+1} *is a direct consequence of the preceding line* X_i *by one of the inference rules*.
>
> [Smu95], chapter XVII.

> A formal proof in first-order logic is a finite sequence of statements of the form $X \vdash Y$ each of which *follows from the previous statements by one of the rules we have listed*...
>
> [Hed04], chapter 1.

A symptom of this issue is that virtually every exposition of such matters tends to be rather wordy. It is very usual in other realms of mathematics to turn to symbols and strictly defined concepts even in textbooks (compare the neat definition of group in section 2.1 of [Her96]). This suggests a pragmatic criterion for assessing the affinity of a treatment with standard set-theoretical language of mathematics, basing on the number and complexity of actual implementations of it in a computer-checked proof system adopting set-theoretical foundations. For first-order languages and deductive calculus only one such implementation already existed, and it is written in Mizar [BK05]: we discuss its shortcomings in sections 2.3 and 2.6.6. One major drawback of [BK05] is that it does not aim to be a general framework in which arbitrary rules can be inserted, rather it deals with provability with a fixed set of rules, with the only goal of getting to Gödel's completeness theorem.

The first task accomplished in this thesis is the formulation of first-order logic and sequent calculus in the standard mathematical foundations of set theory. This is done in chapter 1. Given the view, exposed above, that a good formulation should be effectively formalizable, we try to keep definitions set-theoretically simple, that is, invoking low-level entities. This is especially important for sequent calculus, as already discussed. Very few assumptions are made on the actual rules adopted, not even that of monotonicity. This is a departure from the only theory sharing some traits with the present one which the author is aware of, brought out by Tarski in [Tar28; Tar35; Tar30]; on other accounts, that theory is more general than the present one, being agnostic with respect to the type of calculus (Hilbert, natural deduction, sequent calculus, etc...) adopted. The same chapter also tests this formulation against the proofs of cornerstone applications to model theory and proof theory, like satisfiability, Löwenheim-Skolem and completeness theorems. We should stress here that, while it is certainly obvious to every reader of a textbook on first-order logic that a deductive system can be formalized in set theory, frequently it is not so clear if the writer has even considered the problem of how to face that task. Thus the treatment results often in something quite regardless of the mathematization of the deductive system.

Chapter 2 brings the effort a step further, testing all the contents of chapter 1 even more concretely: it passes from the formulation there contained to its mechanically verified formalization, honoring the criterion hinted above. Given our starting goal of supplying a mathematically-oriented, that is, set-theoretical, formalization of the foundations in themselves, it is natural to choose a verifier adopting set theory axioms and first-order logic. This reduces the candidate verifiers to a handful, of which Mizar

is surely the one with the largest library of already verified mathematics: Mizar Mathematical Library (MML). Besides presenting the Mizar verified formalization, chapter 2 aims to supply (notably in sections 2.4, 2.6.5 and 2.6.6) concrete instances and discussions of the thesis that in formalizing a piece of mathematics there is more than just precisely stating it and certifying its correctness: see [Boy+94] and [Gon08] for general analysis of how much more there is.

Chapter 3 discusses related issues in a more concrete context: it gives Mizar examples of design principles stated in chapter 2 and showcases Mizar coding techniques of general applicability. Notably, section 3.1 discusses some general methods for the Mizar system, whose support for custom automation is usually regarded as poor ([Wie07b], section 4), aiming at bypassing, in limited circumstances, this shortage, and thus of possible interest for other Mizar users.

This work can also be viewed as a study of how the process of mechanically verifying some theory influences back the theory itself. Although mechanization of mathematics presents some important differences with respect to writing common software, the main one being that producing executable code is no longer the final goal, it can bring some arguably beneficial factors from the realm of computer programming into the matter being mechanized. First of all, since 'controlling complexity is the essence of computer programming' ([KP81], page 311), one is led to eliminate all that is not strictly needed, and in general to find approaches minimizing the code to write. This has the side effect of accurately evaluating the point at which some notion or construct is really needed, and which results need which notion or construct. Secondly, and relatedly, once one chooses a specific foundational framework, set theory in our case, he is brought to favor the employment of some theoretical toolkit in lieu of another, if the former is more naturally or more simply expressed in the chosen framework than the latter and, consequently, is somehow better supported by the software used. See point (4) of list below.

It is natural to wonder whether the consequences of adopting design principles like the ones stated above are of a merely technical nature, or rather influence the mathematics to an extent possibly interesting in its own sake. Of such consequences, I put forward some I believe are of more than merely technical interest in the particular case of the present work, and refer the reader to the corresponding points of the text, and to related discussion scattered along chapter 2:

1. The introduction of a definition of language with only two special symbols, and no need for constant symbols.

2. The distinction between free and bound occurrences of a variable is not needed to prove the theorems mentioned above. Indeed it is never stated in this work.

3. Monotonicity of single inference rules can often be replaced by monotonicity of a ruleset, which is a weaker condition. Compare definitions 1.6.0.9 and 1.6.1.1.

4. The definition of sequent derivation and of proof can be substituted by those of derivability (1.5.0.4) and of provability (1.5.0.5), respectively. The latter, in turn, can be made without resorting to the notion of tree, which in set-theory is quite a high-level object, and instead basing on the notion of function iteration. This alternative view is shown to be reconcilable with the standard, tree-based one by proposition 1.6.1.4.

Chapter 1

A set-theoretical treatment of first-order logic

This chapter illustrates a way of expressing the building blocks of first-order logic in a standard set-theoretical background. We will define the notions of first-order language, of formulas, of interpretation, of derivation rule, of derivability and provability. We will also define how to evaluate a formula given an interpretation, how to extract subformulas, how to perform substitutions in a formula. Finally, we will deploy this machinery to obtain satisfiability, completeness and Löwenheim-Skolem theorems, after having introduced a suitable set of derivation rules following our definitions. In chapter 2 the task of concretely pouring this formulation into Mizar code will be faced.

1.1 Preliminaries

In this section we fix most of the set-theoretic notations we will be using throughout the chapter. Most of them is certainly conventional; all the same we prefer to make sure that the reader is aware of the meaning of each involved symbol.

1. $|X|$ is the cardinality of the set X.
2. $X \times Y$ is the cartesian product of the sets X and Y:
$$X \times Y = \{(x, y) : x \in X, y \in Y\}.$$
3. $\mathbb{N}$, $\mathbb{Z}$ are the sets of natural numbers (including $0 = \emptyset$) and of the integers, respectively. We also write $\mathbb{Z}^+$ for $\mathbb{N} \setminus \{0\}$.
4. $\operatorname{dom} P$ and $\operatorname{ran} P$ denote the domain and range of a given relation P.
5. We will use the terms function, map and mapping interchangeably.
6. Y^X is the set of the maps from X into Y.
7. Given sets Y and X, 1_X^Y is the *characteristic function* (also known as *indicator function*) of X, defined on Y:
$$1_X^Y := ((Y \setminus X) \times \{0\}) \cup ((Y \cap X) \times \{1\}).$$

Often, X is declaredly a subset of Y and one can write just 1_X.

8. Since $2^X = \left\{1^X_{X'} : X' \subseteq X\right\}$, it is in a one-to-one correspondence with the power set of X; hence we will also abusively write 2^X for the power set of X. 2^X_n is the set of the subsets of X having n elements, and $\mathcal{F}(X) := \bigcup_{n\in\mathbb{N}} 2^X_n \subseteq 2^X$ is the set of the finite subsets of X.

9. $\}\{_X$ is the map:
$$2^X_1 \ni \{x\} \mapsto x \in X;$$
often, we just indicate it with $\}\{$.

10. $\mathcal{I}_X$ is the identity map on the set X: $\mathcal{I}_X := \bigcup_{x\in X} \{x\} \times \{x\}$.

11. Given sets X, Y, Z, and $f \in Z^{X\times Y}$, the unique $F \in \left(Z^Y\right)^X$ such that $(F(x))(y) = f((x,y)) \forall x \in X, y \in Y$ is the currying (known also as schönfinkeling) of f. We denote as ${}_xf \in Z^Y$ its value in $x \in X$:
$${}_xf : Y \ni y \mapsto f((x,y)).$$

Notation 1.1.0.1. Consider a relation P and a set X. We write $P|_X$ for the restriction of P to X:
$$P|_X := (X \times \operatorname{ran} P) \cap P,$$
and $P[X]$ for the set of those elements of $\operatorname{ran} P$ corresponding through P to some element of X:
$$P[X] := \operatorname{ran}(P|_X).$$

Notation 1.1.0.2. $\bullet$ is the infix symbol for the composition of relations: $(Q \bullet P)[X] = P[Q[X]]$.
$\circ$ is the infix symbol for the composition of functions: $g \circ f : x \mapsto g(f(x))$

Remark 1.1.0.3. Mizar provides one single symbol to denote both relation and function compositions, being able to resolve ambiguities thanks to the typing of the arguments it is applied to. This resolution would require an extra effort to the reader, so we chose to adopt distinct symbols in 1.1.0.2.

Notation 1.1.0.4. Given a set $\mathcal{P}$ all elements of which are relations, we define
$$\lfloor \mathcal{P} \rfloor := \bigcup_{P\in\mathcal{P}} \operatorname{ran} P.$$

Notation 1.1.0.5. If P is a relation such that $\operatorname{ran} P \subseteq \operatorname{dom} P$, we can refer to the n-th iteration of P for any given $n \in \mathbb{N}$. We write it as
$$P^{(n)}.$$

Notation 1.1.0.6 ('Functional pasting with right-hand precedence')**.** Given relations Q, P, set
$$Q \lhd P := Q \setminus (\operatorname{dom} P \times (\operatorname{ran} Q)) \cup P.$$

Remark 1.1.0.7. Given two functions f, g:

- $f \lhd g$ is a function;
- if f and g agree on $\operatorname{dom} f \cap (\operatorname{dom} g)$, then $f \lhd g = f \cup g$.

Definition 1.1.0.8 (Simple substitution)**.** Given y, y' and a function f, we define

$$\frac{y'}{y} f := (\mathcal{I}_{\operatorname{ran} f} \lhd \{(y, y')\}) \circ f \in (\operatorname{ran} f \setminus \{y\} \cup \{y'\})^{\operatorname{dom} f}.$$

Definition 1.1.0.9. Given $n \in \mathbb{N}$, a *n-tuple* (or just tuple) is a function having $\{j \in \mathbb{N} : j < n\} = n$ as a domain. By notation (6) introduced on page 1, then, X^n is the set of all n-tuples valued in X. We set $X^+ := \bigcup_{n \in \mathbb{Z}^+} X^n$, and $X^* := X^+ \cup \{\emptyset\}$. We will also refer to an element of X^n or X^* as a (n-)tuple on X.

Definition 1.1.0.10. Given two tuples p, q, we set

$$p * q := \begin{cases} p & q = \emptyset \\ p \cup (q \circ \{(|p|, 0), \dots, (|p| + |q| - 1, |q| - 1)\}) & \text{otherwise,} \end{cases}$$

that is

$$p * q := p \cup \left(q \circ \left((x \mapsto x - |p|)|_{(|p|+|q|)\setminus|p|}\right)\right).$$

Note that

1. $p * q$ is still a tuple: the functions p and $\left((x \mapsto x - |p|)|_{(|p|+|q|)\setminus|p|}\right)$ have as domains respectively $|p|$ and $(|p| + |q|) \setminus |p|$: being the latter mutually disjoint, $p*q$, as a union of the former functions, is still a function; moreover, its domain is precisely the union of $|p|$ and $(|p| + |q|) \setminus |p|$.
2. $\operatorname{ran}(p * q) = (\operatorname{ran} p) \cup \operatorname{ran} q$.

Hence the mapping $(p, q) \mapsto p * q$ is a binary operation on X^*:

Definition 1.1.0.11. Given X, set $*_X := X^* \times X^* \ni (p, q) \mapsto p * q$.

$*$ is associative. That is:

$$(p * q) * r = p * (q * r)$$

for any three tuples p, q, r. This permits to consider $(X^*, *_X, \emptyset)$ as a monoid, also abusively indicated with X^*. Similarly, X^+ will be also used to denote the subsemigroup $\left(X^+, (*_X)|_{(X^+)}\right)$ of X^* on X^+.

Thanks to its associativity, $*_X$ naturally yields a homomorphism $(X^*)^* \to X^*$, which restricts to a homomorphism $(X^+)^+ \to X^+$; both are denoted by $**_X$.

Notation 1.1.0.12. When no ambiguity arises, we reserve to employ the following shorthand notations, writing

1. x instead of $\{(0, x)\} \in X^1 \subseteq X^+$;
2. pq in place of $p * q$;

3. $p * q * r$ for $(p * q) * r = p * (q * r)$.
4. $*$ instead of $*_X$;
5. $**$ instead of $**_X$.

Remark 1.1.0.13. It would be natural to add to the ones in 1.1.0.12 the further shorthand notation identifying the distinct mappings $*$ and $**$ under one symbol. We refrain from doing so: those distinct functions will occasionally appear together, so being able to resolve between them arguably adds clarity when this happens.

1.2 Languages

Definition 1.2.0.14. A *language* is a triple $(\#, \equiv, \downarrow)$, where $\#$ is an integer-valued function and $\equiv$ is an element of its domain, such that

1. $\#(\equiv) = -2$;
2. $\downarrow \notin \operatorname{dom} \#$;
3. $\#^{-1}(\{0\})$ is not finite.

Notation 1.2.0.15.

- $\#$ is called the *arity* of the language, and $\{\downarrow\} \cup \operatorname{dom} \#$ is called the *symbol set* of the language.
- $\equiv$ is called the *equality symbol* of the language, and $\downarrow$ the *logical connective* of the language.
- Given a language S, we also denote by S its symbol set (so that, e.g. S^* is the free monoid on the latter, and $*_S$ the operation of this monoid); when needed, we may use a subscript to refer explicitly to the arity, equality symbol or logical connective of S: $S = (\#_S, \equiv_S, \downarrow_S)$.
- The elements of $\#_S^{-1}(\{0\})$ are called the *literals* of S, those of $\#_S^{-1}(\mathbb{Z} \setminus \{0\})$ its *compounders*.

Definition 1.2.0.16 (The set of terms of depth not exceeding n)**.**
Given a language S, we recursively construct the following countable family of sets of tuples on S:

$$T_{S,0} := \left(\#^{-1}[\{0\}]\right)^1$$
$$T_{S,n+1} := T_{S,n} \cup \bigcup_{o \in \#^{-1}[\mathbb{Z}^+]} * \left[\{\{(0,o)\}\} \times ** \left[(T_{S,n})^{\#(o)}\right]\right].$$

Definition 1.2.0.17 (Terms of a language)**.**

$$T_S := \bigcup_{n \in \mathbb{N}} T_{S,n}.$$

Definition 1.2.0.18 (The set of formulas of depth not exceeding n)**.**
Given a language S, we recursively construct the following countable family of tuples on S:

$$F_{S,0} := \bigcup_{r \in \#^{-1}[\mathbb{Z}^-]} * \left[\{\{(0,r)\}\} \times ** \left[(T_S)^{|\#(r)|} \right] \right]$$

$$F_{S,n+1} := F_{S,n} \cup * [\{\{(0,\downarrow)\}\} \times * [F_{S,n} \times F_{S,n}]] \cup * \left[\left(\#^{-1} [\{0\}] \right)^1 \times F_{S,n} \right].$$

Definition 1.2.0.19 (The formulas, or well-formed tuples, or wffs of a language)**.**

$$F_S := \bigcup_{n \in \mathbb{N}} F_{S,n}.$$

Definition 1.2.0.20 (Depth of a term and of a formula)**.** The *depth* of a term t of S is written $|t|$, and defined as the least $n \in \mathbb{N}$ such that $t \in T_{S,n}$.
The depth of a formula ψ of S is written $|\psi|$, and defined as the least $n \in \mathbb{N}$ such that $\psi \in F_{S,n}$. A formula of depth zero is said to be *atomic*.

Definition 1.2.0.21. Given a language S, we consider the set

$$G(S) := \mathcal{F}(F_S) \times F_S.$$

An element (Γ, φ) of $G(S)$ is called a *sequent* of the language S; Γ is styled the *antecedent* of the sequent, φ its *succedent*.

1.3 Comments and an example

The definition of a first order language presented here, and the subsequent ones, have been devised with an eye to Mizar formalization: as little and as basic as possible objects were pushed into them. In particular, the following points should be emphasized:

- The first design choice is to use polish notation: for example $x > y + z$ becomes $> x + yz$. This is a common choice in software and in formalization for its simplicity; both [RT90] and [Ban90] adopt it as well.

- There is no quantification symbol. This does not mean that we cannot quantify, of course: *existential* quantification is indicated by heading a formula with a literal symbol, and this gives rise to no ambiguity.
  Of course, universal quantification can be rendered via existential and negation constructs, as is customarily done; we shall soon an applied instance of this in the example about group axioms below.

- There is no native distinction between free and bound variables. What's more, there is not even a distinction between variables and constants symbols. There are only symbols of arity zero, which are called literals, and symbols of non zero arity, called compounders. To be more precise, the distinction is left to the semantics, in the sense that a constant becomes a variable exactly when it is caught by quantification inside a formula.

- Arity yields *signed* natural numbers, with the convention that negative arity symbols are relational (predicate) compounders and positive arity symbols are operational compounders. The absolute value of the arity will indicate the actual arity of the compounder. In many treatments, (even inside Mizar's library, see [RT90]) there are no operational symbols, which can always semantically be emulated by relational (predicate) symbols, but this makes the definition of well-formed formulas (wff) and, most importantly, that of free interpretation, trickier.
- There is only one logical connector, that is NOR, here denoted by 'Peirce arrow' ($\downarrow$). This suffices since NOR is universal (functionally complete), as is its dual NAND ($\uparrow$ or 'Sheffer stroke').
- Term substitution, 1.8.0.32, will be defined by leveraging the pre-existing notions of reassignment, of evaluation of an interpretation, and of free interpretation. Additionally, simple substitution, 1.1.0.8, is preferred to it when sufficing, as in definition of W, 1.9.4.6, and of rule $R_{\underset{\exists}{\leftarrow}}$, see 1.4.1.1.

Therefore, in definitions regarding syntax and semantics, we have can take advantage of dealing with only two special symbols: equality and NOR; notably in treating wff formulas and evaluation (see 2.6.3), this will be a life-saving simplification.

To give one among the simplest illustrations, let us rephrase in this language the group axioms, using $\mathbb{N}$ as a symbol set, 1 as $\equiv$, 0 as $\downarrow$, and an arity $f\colon \mathbb{Z}^+ \to \mathbb{Z}$ given by

$$f(n) := \begin{cases} -2 & \text{if n=1} \\ 2 & \text{if n=2} \\ 0 & \text{otherwise} \end{cases}$$

Direct translation might result bewildering, so let us first list axioms in standard human-friendly form (on the left in the table below) and in an intermediate jargon made by combining polish notation with shortcut symbols $\exists, \forall, =, +$ for quantifiers and compounders:

$$\begin{aligned} \forall a, b, c\ a(bc) = (ab)c \qquad & \forall 3\forall 4\forall 5\ = +3 + 45 + +345 \\ \forall a\ ea = a \qquad & \forall 4\ = +344 \\ \forall a \exists b\ ba = e \qquad & \forall 4\exists 5\ = +543. \end{aligned}$$

Finally, we pass to the real coding first by rendering $\forall x\phi$ as $\neg\exists x\phi$, $\neg\phi$ as $\downarrow \phi\phi$, $\exists x\phi$ as $x\phi$, and subsequently by substituting $=$, $+$ respectively with 1, 2, in the end obtaining some nasty strings:

$$\begin{aligned} &0304051232452342551232452342540512324523425512324523425 \\ &040512324523425512324523425405123245234255123245234 25 \\ &0412344412344 \\ &045124534512453, \end{aligned} \tag{1.1}$$

where the first, exceedingly long axiom has been split across two lines.

This shows how the absence of auxiliary boolean connectors and quantifiers makes even trivial formulas go wildly verbose. Note that none of the three axioms uses more than seven literals, so we have been able to unambiguously use decimal representation for $\mathbb{N}$. Also compare the role of the symbol '3' in expressing first and second axioms: in the first case it is quantified and thus used as a variable, while in the second it acts as a constant (the unity of the group) since it is not quantified. Not having distinguished between constants and variables permits reusing a literal symbol in both ways, as long as the corresponding constant does not appear in the formula in which the symbol is used as a variable. Given our goals, we do not care much about readability of the language: all that matters is that any first-order theory is expressible in the language, and that a proof calculus being both sound and *complete* (that is, powerful enough to prove any consequence of a first-order theory) is provided, which we did with completeness theorem 1.11.0.18. Under these constraints, we sought for the design maximizing simplicity and neatness of formalization.

1.4 Formal definition of derivation rule

Definition 1.4.0.22 (Rules and rulesets)**.** A *derivation rule*, or *inference rule* for S is any map $2^{G(S)} \to 2^{G(S)}$. A *ruleset* of S is a set of derivation rules, that is, a subset of $\left(2^{G(S)}\right)^{\left(2^{G(S)}\right)}$.

Notation 1.4.0.23 (Character reservations; abbreviations for writing sequents)**.**

- As a rule, we will use the letter S to indicate a language, and X to indicate a generic set.
- We conventionally agree to reserve (unless otherwise specified) some characters according to the type of S-related objects we will want to denote:
  - s for an element of $\operatorname{dom} \#_S$,
  - v for a literal,
  - w for a tuple on S,
  - t for a term,
  - Γ for a *finite* set of formulas,
  - φ, ψ for a formula,
  - Ψ for a set of formulas,
  - σ for a sequent,
  - Σ for a set of sequents,
  - R for an inference rule, and
  - D for a ruleset.

  Subscripts or superscripts will be added when needed.

- A sequent (Γ, φ) will be often represented as $\Gamma \quad \vdash \quad \varphi$.
- When writing a sequent, the following abbreviations can be adopted:

$$\begin{array}{rrcll} \Gamma_1 & \Gamma_2 & \vdash & \varphi & \text{in lieu of} \quad \Gamma_1 \cup \Gamma_2 \quad \vdash \quad \varphi \\ \Gamma & \psi & \vdash & \varphi & \text{in lieu of} \quad \Gamma \cup \{\psi\} \quad \vdash \quad \varphi. \end{array}$$

- The turnstile symbol ($\vdash$) parting antecedent from succedent can be omitted when adopting the foregoing abbreviations for writing a sequent.

Example 1.4.0.24. Consider $\Gamma_1 := \{\psi_1, \psi_2\}, \Gamma_2 := \{\psi_3\}, \Gamma := \Gamma_1 \cup \Gamma_2$.
Here is a list of some of the notations rendering the sequent (Γ, φ), obtainable by combining shorthand notations introduced in 1.4.0.23:

$$\begin{array}{rrrrcl} \psi_1 & \psi_3 & \psi_2 & \psi_3 & \vdash & \varphi \\ & & \Gamma_1 & \Gamma_2 & \vdash & \varphi \\ & \{\psi_1, \psi_2\} & & \psi_3 & \vdash & \varphi \\ & \{\psi_1, \psi_2, \psi_3\} & & & \vdash & \varphi \\ & & \psi_1 & \psi_2 & \psi_3 & \varphi. \end{array}$$

1.4.1 An example of ruleset

Definition 1.4.1.1. We introduce some particular derivation rules of the language S by specifying the way each acts on a given $\Sigma \subseteq G(S)$:

$$R_0(\Sigma) := \{(\Gamma, \varphi) : \Gamma = \{\varphi\}\}$$

$$R_{\cup}(\Sigma) := \{(\Gamma, \varphi) : \exists (\Gamma', \varphi) \in \Sigma | \Gamma' \subseteq \Gamma\}$$

$$R_{=}(\Sigma) := \{(\Gamma, \varphi) : \exists t | \Gamma = \emptyset \text{ and } \varphi = \equiv tt\}$$

$$R_{\underset{=}{\leftrightarrow}}(\Sigma) := \{(\Gamma, \varphi) : \exists t_1, t_2 | \Gamma = \{\equiv t_1 t_2\} \text{ and } \varphi = \equiv t_2 t_1\}$$

$$R_{\underset{=}{\Rightarrow}}(\Sigma) := \{(\Gamma, \varphi) : \exists t_1, t_2, t_3 | \Gamma = \{\equiv t_1 t_2, \equiv t_2 t_3\} \text{ and } \varphi = \equiv t_1 t_3\}$$

$$R_{+}(\Sigma) := \{(\Gamma, \varphi) : \exists n \in \mathbb{Z}^+, s \in S, \mathbf{t}, \mathbf{t}' \in (T_S)^n | \varphi = \equiv s * * (\mathbf{t}) s * * (\mathbf{t}') \text{ and } \Gamma = \{\equiv \mathbf{t}(j) \mathbf{t}'(j), j \in n\}\}$$

$$R_{\mathcal{R}}(\Sigma) := \{(\Gamma, \varphi) : \exists n \in \mathbb{Z}^+, s \in S, \mathbf{t}, \mathbf{t}' \in (T_S)^n | \varphi = s * * (\mathbf{t}') \text{ and } n = -\#(s) \text{ and } \Gamma = \{\equiv \mathbf{t}(j) \mathbf{t}'(j), j \in n\} \cup \{s * * (\mathbf{t})\}\}$$

$$R_{\downarrow}(\Sigma) := \{(\Gamma, \varphi) : \exists \varphi_1, \varphi_2, \varphi_3, \varphi_4 \in F_S | \Gamma = \{\downarrow \varphi_1 \varphi_2, \downarrow \varphi_3 \varphi_4\} \text{ and } \varphi = \downarrow \varphi_2 \varphi_3\}$$

$$R_{\underset{\exists}{\leftarrow}}(\Sigma) := \left\{(\Gamma, \varphi) : \exists v, v_1, v_2, \psi, \Gamma' | \left(\Gamma' \cup \left\{\tfrac{v_2}{v_1}\psi\right\}, \varphi\right) \in \Sigma \text{ and } \varphi = \downarrow \equiv vv \equiv vv \text{ and } \Gamma = \Gamma' \backslash \left\{\tfrac{v_2}{v_1}\psi\right\} \cup \{v_1 \psi\} \text{ and } v_2 \notin \lfloor \Gamma' \cup \{\psi\} \rfloor\right\}$$

$$R_c(\Sigma) := \{(\Gamma, \varphi) : \exists \psi_1, \psi_2 | (\Gamma \cup \{\psi_1\}, \psi_2), (\Gamma \cup \{\psi_1\}, \downarrow \psi_2 \psi_2) \in \Sigma \text{ and } \varphi = \downarrow \psi_1 \psi_1\}$$

$$R_{\not\downarrow}(\Sigma) := \{(\Gamma, \varphi) : (\Gamma, \downarrow \downarrow \varphi \varphi \downarrow \varphi \varphi) \in \Sigma\}.$$

Notation 1.4.1.2. When wanting to express the particular language S relative to which one of the rules defined in 1.4.1.1 is to be meant, we adjoin its name S to the rule's subscript, as in $R_{=,S}$.

1.5 Formal definitions of derivability and provability

If we want to formalize results about completeness of first-order languages in a first-order language like Mizar or set theory, we first have to rigorously define in it what a proof is. It turns out that it is both sufficient and convenient to establish the notion of provability rather than that of proof.

Definition 1.5.0.3. Given a ruleset D of S, we define the following derivation rule of S:

$$\overline{D} : \Sigma \mapsto \bigcup_{R \in D} R(\Sigma). \tag{1.2}$$

Definition 1.5.0.4. A sequent belonging to $\overline{D}^{(n)}(\Sigma)$ will be said to be *derivable* from Σ through D in n steps.

The set of all sequents derivable from Σ through D will be indicated with $\overline{D}^{(\infty)}(\Sigma)$:

$$\overline{D}^{(\infty)}(\Sigma) := \bigcup_{n \in \mathbb{N}} \overline{D}^{(n)}(\Sigma).$$

Definition 1.5.0.5 (Formal definition of provability)**.** Given S, X and D, we set

$$D(X) := \operatorname{ran}\left(2^X \times F_S \cap \left(\overline{D}^{(\infty)}(\emptyset)\right)\right) \subseteq F_S.$$

As well as $\varphi \in D(X)$, one can also write $X \overset{\vdash}{}_{D} \varphi$, and say that X proves φ in D, or that φ is provable from X in D.

Remark 1.5.0.6. Equivalently, $X \overset{\vdash}{}_{D} \varphi$ if and only if there is a sequent $(\Gamma, \varphi) \in \overline{D}^{(\infty)}(\emptyset)$ such that $\Gamma \subseteq X$.
Alternatively, since $\overline{D}^{(0)}(\emptyset) = \emptyset$, $X \vdash_D \varphi$ if and only if there are $n \in \mathbb{N}$, $\Gamma \in \mathcal{F}(F_S)$ such that $(\Gamma, \varphi) \in \overline{D}^{(n+1)}(\emptyset)$.

Remark 1.5.0.7. In 1.2.0.21 we defined sequents of S as having for an antecedent a finite subset of F_S. Other conventions are to define sequents having either multisets or tuples of formulas as an antecedent. The one adopted here, however, involves lower-level objects than the other two, if one works in a set-theoretical formal framework as we are doing. Moreover, it allows dispensing with introducing exchange and contraction rules.

Definition 1.5.0.8. X is said to be *deductively closed* with respect to D (or just D-closed) if

$$D(X) \subseteq X.$$

1.6 Justification of diagrams

Definition 1.6.0.9. A rule R of S is said to be *monotone* if it is monotone with respect to the partial order $\subseteq$ of $G(S)$; that is, for any $\Sigma_1, \Sigma_2 \in G(S)$ such that $\Sigma_1 \subseteq \Sigma_2$, it is:

$$R(\Sigma_1) \subseteq R(\Sigma_2).$$

Remark 1.6.0.10. Any hypothesis requesting some rule to be monotone will always be made explicit. However, all the concrete examples of rule we will introduce will be monotone. This will be often exploited without explicit mention.

Definition 1.6.0.11. Given a derivation rule R of S and $n \in \mathbb{N}$, we write

$$R \leq n$$

to mean that for any $\Sigma_2 \subseteq G(S), \sigma \in R(\Sigma_2)$, there is $\Sigma_1 \subseteq \Sigma_2$ with $|\Sigma_1| = n$ such that $\sigma \in R(\Sigma_1)$. In this case we say that n is an *upper bound* for R.

If $R \leq 0$ we say R is an *axiom*.

All the rules introduced in 1.4.1.1 are monotone and have 2 as an upper bound (some even admit 1 as an upper bound, with many being just axioms): roughly speaking, this means that each sequent belonging to the image of a given Σ through one of those rules can be derived by applying that rule just to a suitable subset of Σ having cardinality either 0 (for those rules being axioms), 1 or 2.

This allows us to introduce schematic diagrams succinctly illustrating how each of our rules work by a graphical arrangement describing its action on a given generic pair of sequents (or either respectively on a single sequent or on the empty set). This description is done simply by listing above a horizontal line the input sequent(s), if any, and below it the output sequent:

$$R_0\frac{}{\varphi \vdash \varphi} \quad R_\cup\frac{\Gamma \vdash \varphi}{\Gamma' \vdash \varphi} \text{ where } \Gamma \subseteq \Gamma'$$

$$R_=\frac{}{\vdash \equiv tt} \quad R_{\overset{\leftrightarrow}{=}}\frac{}{\equiv t_1t_2 \vdash \equiv t_2t_1} \quad R_{\overset{\Rightarrow}{=}}\frac{}{\equiv t_1t_2 \equiv t_2t_3 \vdash \equiv t_1t_3}$$

$$R_+\frac{}{\equiv t_1t_1' \ \dots \ \equiv t_nt_n' \vdash \equiv st_1 \dots t_nst_1' \dots t_n'} \quad \text{where } n = \#(s) \in \mathbb{Z}^+$$

$$R_{\mathcal{R}}\frac{}{st_1 \dots t_n \equiv t_1t_1' \ \dots \ \equiv t_nt_n' \vdash st_1' \dots t_n'} \quad \text{where } n = -\#(s) \in \mathbb{Z}^+$$

$$R_\downarrow\frac{}{\downarrow \varphi_1\varphi_2 \downarrow \varphi_3\varphi_4 \vdash \downarrow \varphi_2\varphi_3}$$

$$R_{\overset{\leftarrow}{\exists}}\frac{\Gamma \quad \frac{v_2}{v_1}\varphi \vdash \downarrow \equiv vv \equiv vv}{\Gamma \quad v_1\varphi \vdash \downarrow \equiv vv \equiv vv} \quad \text{where } v_2 \text{ does not occur in } \Gamma, \varphi$$

$$R_c\frac{\Gamma \ \varphi \vdash \psi \qquad \Gamma \ \varphi \vdash \downarrow \psi\psi}{\Gamma \vdash \downarrow \varphi\varphi} \quad R_{\not\vdash}\frac{\Gamma \vdash \downarrow\downarrow \varphi\varphi\downarrow \varphi\varphi}{\Gamma \vdash \varphi}$$

We lastly observe that such a suggestive representation of rules is effective because each of the latter works in a syntactically simple manner: hence its action is immediately conveyed by glancing at the variations of the morphological patterns between the sequent schematas above and below the horizontal line.

This is one of the reasons for splitting derivations into several applications of different rules: otherwise we could have helped the trouble of introducing the definitions of a ruleset D and of the derived rule $\overline{D}$ (see 1.5.0.3), and rather state directly 1.5.0.4 and 1.5.0.5 in terms of a single generic, comprehensive rule taking the place of $\overline{D}$.

1.6.1 Justification for the introduction of derivation trees. Formal definitions of derivation and proof

Motivation

Although the notions of derivability and provability of 1.5 will turn out, throughout chapters 1 and 2, to be perfectly sufficient to formalize (see [Cam11e]) all our results, a human is usually more comfortable in carrying out and conveying reasonings involving those notions if he adopts some interface to them more resembling a

calculation. To this end, we will obtain a graphical representation of such calculi in form of oriented trees, which matches the diagrams introduced in 1.6. We start with a rather elementary notational convention. For a generic rule R and sequents σ_1, σ_2, instead of writing $\sigma_2 \in R(\{\sigma_1\})$, we just write

$$\frac{\sigma_1}{\sigma_2} R.$$

Now, the convenience we gain is that such writings can be 'piled up', resulting in a more natural way of expressing a succession of rule applications. When dealing with rules not all of which are bounded by 1, such 'piles' become *trees*.

Formal definitions

The aforementioned trees, which will be referred to as *derivations*, can be rigorously defined in terms of derivability (1.5.0.4) and of a basic subset of the usual gear of graph theory. First of all we note that we need the assumption that the rules involved are monotone to proceed. In fact the fitting notion is for rulesets.

Definition 1.6.1.1. A ruleset D is said to be *monotone* if and only if the rule $\overline{D}$ is monotone.

Now the reader may want to consult some reference on graphs (e.g., [Knu97], section 2.3.4.2, 'Oriented trees') for the few standard definitions and results about trees we will need in what follows.

Notation 1.6.1.2. Given an oriented tree $T := (V, E)$, we denote with $|T|$ its depth, with r_T the root of T, that is the only element of $V \setminus \operatorname{ran} E$, and with Γ_T the set $V \setminus \operatorname{dom} E$ (that is, the set of the leaves of T).

Definition 1.6.1.3 (Recursive definition of a derivation tree)**.** Let $T := (V, E)$ be an oriented tree with $n+2$ vertices for some $n \in \mathbb{N}$. Denote as $r_1, \ldots, r_l$ the distinct elements of $E[\{r_T\}]$ (that is, the vertices of T having depth 1), with $T_j, j = 1, \ldots, l$ the unique oriented sub-tree of T having r_j as a root.
Let f be a function with $V \subseteq \operatorname{dom} f$ and $\operatorname{ran} f \subseteq G(S)$. We say that (T, f) is a *D-derivation*, where D is a ruleset of the language S, if

- $|T| = 1$ and $r \in R(f[\Gamma_T])$ for some $R \in D$.
- $|T| = m+2$ for some $m \in \mathbb{N}$, there is $R \in D$ such that $f(r_T) \in R(f[\{r_1, \ldots, r_l\}])$, and, for each $j \in l + 1$:
  - $|T_j| = m + 1$, and
  - (T_j, f) is a D-derivation.

The final step is to state the existence of a D-derivation as sufficient condition for the derivability of its root sequent from the set of its leaves according to the rules of D:

Proposition 1.6.1.4. *If D is a monotone ruleset of S and $(T = (V, E), f)$ is a D-derivation of depth $n + 1 \in \mathbb{Z}$, then $f(r_T) \in \overline{D}^{(n+1)}(f[\Gamma_T])$.*

Proof. By induction on n. For $n = 0$ the thesis is immediate from 1.6.1.3.
Assume $n = m + 1$ for some $m \in \mathbb{N}$. As done in 1.6.1.3, denote with $r_1, \dots, r_l$ the distinct elements of $E[\{r_T\}]$, and with $T_j, j = 1, \dots, l$ the unique oriented subtree of T having r_j as root.
By 1.6.1.3, each (T_j, f) is a D-derivation and has depth $m + 1$; thus, by the inductive hypothesis, $f(r_j) \in \overline{D}^{(m+1)}\left(f\left[\Gamma_{T_j}\right]\right)$. 1.6.1.3 also says that $f(r_T) \in R(f[\{r_1, \dots, r_l\}])$ for some $R \in D$. Hence $f(r_T) \in \overline{D}(f[\{r_1, \dots, r_l\}])$. Since $\overline{D}$ is monotone, we conclude

$$f(r_T) \in \overline{D}\left(\bigcup_j \overline{D}^{(m+1)}\left(f\left[\Gamma_{T_j}\right]\right)\right). \tag{1.3}$$

Now, $\overline{D}^{(m+1)}$ is monotone as well, and $f\left[\Gamma_{T_j}\right] \subseteq f[\Gamma_T]$, yielding

$$\bigcup_j \overline{D}^{(m+1)}\left(f\left[\Gamma_{T_j}\right]\right) \subseteq \overline{D}^{(m+1)}(f[\Gamma_T]).$$

Using this (again along with the fact that $\overline{D}$ is monotone) inside (1.3), we get $f(r_T) \in \overline{D}^{(m+2)}(f[\Gamma_T])$. □

Definition 1.6.1.5. A *D-proof* is a D-derivation (T, f) such that

$$f[\Gamma_T] \subseteq \overline{D}^{(1)}(\emptyset).$$

1.7 Elementary results concerning derivability and provability

Proposition 1.7.0.6. *Given $D_1 \subseteq D_2$ such that at least one among D_1 and D_2 is monotone, for any $\Sigma_1 \subseteq \Sigma_2$ and any $n \in \mathbb{N}$ it holds*

$$\overline{D_1}^{(n)}(\Sigma_1) \subseteq \overline{D_2}^{(n)}(\Sigma_2).$$

Proof. By induction on n. For $n = 0$, we have trivially $\overline{D_1}^{(0)}(\Sigma_1) = \Sigma_1 \subseteq \Sigma_2 = \overline{D_2}^{(0)}(\Sigma_2)$. Now assume $n = m + 1$ for some $m \in \mathbb{N}$.

$$\overline{D_1}^{(n)}(\Sigma_1) = \overline{D_1}\left(\overline{D_1}^{(m)}(\Sigma_1)\right) \subseteq \begin{cases} \overset{!}{\subseteq} \overline{D_1}\left(\overline{D_2}^{(m)}(\Sigma_2)\right) \overset{1.5.0.3}{\subseteq} \overline{D_2}\left(\overline{D_2}^{(m)}(\Sigma_2)\right) \\ \overset{1.5.0.3}{\subseteq} \overline{D_2}\left(\overline{D_1}^{(m)}(\Sigma_1)\right) \overset{!}{\subseteq} \overline{D_2}\left(\overline{D_2}^{(m)}(\Sigma_2)\right) \end{cases} = \overline{D_2}^{(n)}(\Sigma_2).$$

In the reasoning above, upper branch is for the case D_1 monotone, lower branch is for the case D_2 monotone. In both, '!' denotes the passages invoking inductive hypothesis together with (respective) monotonicity hypothesis. □

Proposition 1.7.0.7. *If D is monotone, then*

$$\overline{D}^{(n)}(\emptyset) \subseteq \overline{D}^{(n+1)}(\emptyset)$$

for any $n \in \mathbb{N}$.

Proof. By induction on n:

$$\overline{D}^{(0)}(\emptyset) = \emptyset \subseteq \overline{D}^{(1)}(\emptyset).$$

Assuming $\overline{D}^{(n)}(\emptyset) \subseteq \overline{D}^{(n+1)}(\emptyset)$, one has

$$\overline{D}\left(\overline{D}^{(n)}(\emptyset)\right) \subseteq \overline{D}\left(\overline{D}^{(n+1)}(\emptyset)\right)$$

by monotonicity. □

Definition 1.7.0.8. Ruleset D_2 *emulates* ruleset D_1 *from* Σ (written $D_2 \geq_\Sigma D_1$) if

$$\bigcup_{n\in\mathbb{Z}^+} \overline{D_1}^{(n)}(\Sigma) \subseteq \bigcup_{n\in\mathbb{Z}^+} \overline{D_2}^{(n)}(\Sigma).$$

D_2 emulates D_1 (written $D_2 \geq D_1$) if, for each $\Sigma \subseteq G(S)$:

$$D_2 \geq_\Sigma D_1.$$

Remark 1.7.0.9. Given $\Sigma \subseteq G(S)$, the relation $\geq_\Sigma$ is transitive:

$$D_2 \geq_\Sigma D_1 \quad \text{and} \quad D_3 \geq_\Sigma D_2 \quad \text{imply} \quad D_3 \geq_\Sigma D_1.$$

Corollary 1.7.0.10 (of 1.7.0.6)**.** *If $D_1 \subseteq D_2$ and at least one of D_1 and D_2 is monotone, then*

$$D_2 \geq D_1.$$

Proposition 1.7.0.11. *If $X \vdash_{\overline{D_1}} \varphi$ and $D_2 \geq_\emptyset D_1$, then $X \cup Y \vdash_{\overline{D_2}} \varphi$.*

Corollary 1.7.0.12 (of 1.7.0.10 and 1.7.0.11)**.** *If at least one of D_1, D_2 is monotone, then*

$$D_1 \subseteq D_2 \quad \textit{and} \quad X \vdash_{\overline{D_1}} \varphi \quad \textit{imply} \quad X \vdash_{\overline{D_2}} \varphi.$$

Corollary 1.7.0.13 (of 1.7.0.11)**.** *If X is D_2-closed and $D_2 \geq_\emptyset D_1$, then X is D_1-closed.*

1.8 Semantics

It is not difficult to show that $**|_{F_S{}^1 \cup F_S{}^2}$ is one-to-one, and, by recursion on n (see section 2.5), that $**|_{(T_S)^n}$ is one-to-one; this permits defining the following three functions.

The first one is in $(T_S^*)^{T_S}$:

Definition 1.8.0.14 (Subterms of a term)**.**

$$\odot_0 := t \mapsto \begin{cases} \emptyset & \text{if } t \in T_{S,0} \\ \left(\left(**|_{(T_S)^{\#(t(0))}}\right)^{-1} \circ \left({}_{(t|_1)}*\right)^{-1}\right)(t) & \text{otherwise.} \end{cases}$$

The second function is in $(T_S^*)^{F_{S,0}}$:

Definition 1.8.0.15 (Subterms of an atomic formula)**.**

$$\odot_1 := \psi_0 \mapsto \left(**|_{(T_S)^- \#(\psi_0(0))}\right)^{-1} \left(\left({}_{(\psi_0|_1)}*\right)^{-1}(\psi_0)\right).$$

Finally, the third function is in $\left((F_S)^1 \cup (F_S)^2\right)^{F_S \setminus F_{S,0}}$:

Definition 1.8.0.16.

$$\odot_2 := \psi \mapsto \left(\left(**|_{(F_S{}^1 \cup F_S{}^2)}\right)^{-1} \circ \left({}_{\psi|_1}*\right)^{-1}\right)(\psi).$$

In 1.8.0.14, 1.8.0.15 and 1.8.0.16, we took advantage of the easy fact that ${}_x(*_X)$ is one-to-one for any X and $x \in X^*$.

Since $\odot_0$, $\odot_1$ and $\odot_2$ have mutually disjoint domains, we can refer to the function resulting from their union, denoting it simply as $\odot$:

Definition 1.8.0.17 (Sub-tuples of a term or wff)**.**

$$\odot := \odot_0 \cup \odot_1 \cup \odot_2 \in \left((T_S)^* \cup (F_S)^1 \cup (F_S)^2\right)^{(T_S \cup F_S)}.$$

Notation 1.8.0.18. We will often write $\overrightarrow{w}$ in place of $\odot(w)$. If w is a non-atomic formula, $\overrightarrow{w}$ are the *subformulas* of w, while if it is an atomic formula or a term, $\overrightarrow{w}$ are the *subterms* of w.

Remark 1.8.0.19. If ψ is a non-atomic formula, then the number of its subformulas, $\left|\overrightarrow{\psi}\right|$, is either 1 (if $\psi(0)$ is a literal) or 2 (if $\psi(0) =\downarrow$).

Definition 1.8.0.20 (Interpretation and universe)**.** Given a language S, an *interpretation of* S is a function i for which there is a non empty set U (called the *universe* of the interpretation) such that

$$\forall s \in \operatorname{dom} \#_S,\ i(s) \in \begin{cases} U^{\left(U^{\#(s)}\right)} & \text{if } \#(s) \geq 0 \\ \{0,1\}^{\left(U^{-\#(s)}\right)} & \text{if } \#(s) < 0. \end{cases}$$

Notation 1.8.0.21. The symbol i, with optional subscripts and superscripts, will be reserved for generic interpretations from now on, unless otherwise specified.

Remark 1.8.0.22. Every interpretation has exactly one universe.

Remark 1.8.0.23. According to 1.8.0.20, an interpretation having universe U assigns to each literal a map of the form $\{(\emptyset, u)\}$, where $u \in U$, rather than assigning to it directly the value u.

Example 1.8.0.24 (The free interpretation)**.** Given X and a language S, the *free* interpretation of S given by X is the interpretation of S having T_S as universe and defined thus:

$$\Phi_X := \operatorname{dom} \# \ni s \mapsto \begin{cases} \left({}_{\{(0,s)\}}*\right) \circ \left(**|_{(T_S{}^{\#(s)})}\right) & \#(s) \geq 0 \\ 1_X^{F_S} \circ \left({}_{\{(0,s)\}}*\right) \circ \left(**|_{(T_S{}^{-\#(s)})}\right) & \#(s) < 0. \end{cases}$$

Notation 1.8.0.25 (Reassignment of a literal in an interpretation)**.** Given an interpretation i, an element u' of its universe, and a literal v, we introduce the shorthand notation

$$\frac{u'}{v} i := i \lhd \{(v, \{(\emptyset, u')\})\}$$

designating a new interpretation with the same universe of i, called a *reassignment* of v in i.

Definition 1.8.0.26 (Evaluation of terms and atomic formulas)**.** Given an interpretation i of universe U, we define

$$\bar{i}(t_0) := (i(t_0(0)))(\emptyset) \qquad \forall t_0 \in T_{S,0},$$

then recursively:

$$\bar{i}(t) := (i(t(0)))\left(\bar{i} \circ \overrightarrow{t}\right), \qquad t \in T_S;$$

and finally, given $\psi_0 \in F_{S,0}$:

$$\bar{i}(\psi_0) := \begin{cases} (i(\psi_0(0)))\left(\bar{i} \circ \overrightarrow{\psi_0}\right) & \psi_0(0) \neq \equiv \\ 1 & \psi_0(0) = \equiv \text{ and } \bar{i}\left(\overrightarrow{\psi_0}(0)\right) = \bar{i}\left(\overrightarrow{\psi_0}(1)\right) \\ 0 & \text{otherwise.} \end{cases}$$

Definition 1.8.0.27 (Evaluation of non-atomic formulas)**.** Given an interpretation i of universe U, we recursively define

$$\bar{i}(\psi) := \begin{cases} 1 & \text{if } \exists v \in \#^{-1}[\{0\}], u \in U \mid \left(v = \psi(0) \text{ and } \overline{\frac{u}{v} i}\left(\overrightarrow{\psi}(0)\right) = 1\right) \\ 1 & \text{if } \psi(0) = \downarrow \text{ and } \bar{i} \circ \overrightarrow{\psi} = 2 \times \{0\} \\ 0 & \text{otherwise} \end{cases}$$

for every $\psi \in F_S \backslash F_{S,0}$.

Definition 1.8.0.28. Merging 1.8.0.26 with 1.8.0.27, we in the end obtain a function

$$\bar{i} : (T_S \cup F_S) \to (U \cup \{0, 1\}),$$

called the *evaluation* of the interpretation i.

Notation 1.8.0.29 (Model, or satisfaction, relation)**.** Instead of writing $\bar{i}|_{F_S}[X] \subseteq \{1\}$, one often writes $i \models_S X$, or simply $i \models X$, and says that i is a *model* of X, or that i *satisfies* X.

Definition 1.8.0.30. A ruleset D is *sound* if $X \vdash_D \varphi$ and $i \models X$ imply $\bar{i}(\varphi) = 1$.

Remark 1.8.0.31. Any hypothesis requesting some generic ruleset to be sound will always be made explicit. However, all the concrete examples of ruleset we will introduce will be sound.

Definition 1.8.0.32 (Depth-recursive definition of term substitution in a formula)**.** Given v and t, define the map $[v/t] : F_S \to F_S$ as follows:

$$[v/t](\varphi_0) := \left(\varphi_0|_{\{0\}}\right) * \left(** \left(\left(\overline{\frac{t}{v}\Phi_\emptyset}\right) \circ \overrightarrow{\varphi_0}\right)\right)$$

for any atomic formula φ_0; then, given $\varphi \in F_{S,n+1} \backslash F_{S,n}$, recursively on n:

$$[v/t](\varphi) := \begin{cases} \left(\varphi|_{\{0\}}\right) * (** ([v/t] \circ \overrightarrow{\varphi})) & \text{if } \varphi(0) = \downarrow \\ \{(0, v')\} * \left([v/t]\left(\frac{v'}{\varphi(0)}(\overrightarrow{\varphi}(0))\right)\right) & \text{otherwise, where} \\ & v' \notin \{v\} \cup \lfloor \{t, \overrightarrow{\varphi}(0)\} \rfloor . \end{cases}$$

There is a glitch in 1.8.0.32, in that its outcome actually depends on the choice of the literal v' appearing in its definiens. This is immaterial, however, since the different formulas obtained by varying v' are all good candidates to be the substitution result for our purpose: as long as the outcome obeys substitution lemma (see 1.9.4.5), it is acceptable. So we chose not to specify this dependance in 1.8.0.32. To make matters rigorous, one could fix a suitable choice function $\eta : \left(2^{\#^{-1}[\{0\}]}\right) \setminus \left\{\#^{-1}[\{0\}]\right\} \ni X \mapsto x \in \left(\#^{-1}[\{0\}]\right) \backslash X$ and define $[v/t]_\eta$ by substituting v' with $\eta(\{v\} \cup \lfloor \{t, \overrightarrow{\varphi}(0)\} \rfloor)$ inside the definiens of 1.8.0.32, which, however, would probably result a bit too cluttered this way. In Mizar one utterly bypasses such problems generically related to the dependence on some choice function by using the construct `the`, which provides an object of the given type, undefined yet usable as if it was defined. It should be noted, however, that this device as well is merely a convenient way, offered by Mizar, to invoke the axiom of choice: [Try].

Notation 1.8.0.33. We will often write $\psi[v/l]$ instead of $[v/l](\psi)$.

We now introduce a further derivation rule we will need.

Definition 1.8.0.34.

$$R^{\to}_{\exists}(\Sigma) := \{(\Gamma, \varphi) : \exists v, t, \psi | \Gamma = \{\psi[v/t]\} \text{ and } \varphi = v\psi\} .$$

Since $R^{\to}_{\exists} \leq 0$, we can depict $R^{\to}_{\exists}$ via a diagram as those from section 1.6:

Notation 1.8.0.35.

$$R^{\to}_{\exists} \quad \overline{\psi[v/t] \quad \vdash \quad v\psi}$$

1.9 Henkin interpretation

1.9.1 Quotients

Definition 1.9.1.1. Let P, Q be relations, f be a function. We say that f is (P, Q)-compatible if, given $(x, y) \in \operatorname{dom} f \times (\operatorname{dom} f) \cap P$, it is $(f(x), f(y)) \in Q$.

Remark 1.9.1.2. In Mizar code, the keyword `-compatible` being already in use, the attribute `-respecting` is used instead.

Definition 1.9.1.3. Given a non empty relation P, we consider the map

$$\pi_P : \operatorname{dom} P \ni x \mapsto P[\{x\}] \in 2^{\operatorname{ran} P}.$$

Given a set X and a relation P such that $X = \operatorname{dom} P$, we set

$$X/P := \operatorname{ran}(\pi_P).$$

Remark 1.9.1.4. If P is an equivalence relation over X, X/P is the set of the equivalence classes of P (hence a partition of X), and π_P maps each element of the domain of P to the unique equivalence class including it.

Definition 1.9.1.5 (Quotient of a relation)**.** Let O, P, Q be relations, with P and Q non empty. The quotient of O by (P, Q) is defined as:

$$\frac{O}{P \quad Q} := \{(p, q) \in \operatorname{ran}(\pi_P) \times (\operatorname{ran}(\pi_Q)) : p \times q \cap O \neq \emptyset\}.$$

Proposition 1.9.1.6. *Let E, F be non empty equivalence relations. If $f \in (\operatorname{dom} F)^{\operatorname{dom} E}$ is (E, F)-compatible, then*

$$\frac{f}{E \quad F} \in (\operatorname{ran} \pi_F)^{\operatorname{ran} \pi_E}.$$

Proof. Set $g := \frac{f}{E \quad F}$. Since $g \subseteq \operatorname{ran} \pi_E \times \operatorname{ran} \pi_F$ by 1.9.1.5, it is $\operatorname{ran} g \subseteq \operatorname{ran} \pi_F$, hence we are left with two points to prove:

1. g is functional.
2. g is left-total, that is, $\operatorname{ran} \pi_E \subseteq \operatorname{dom} g$.

The two corresponding proofs are given.

1. Consider sets X, Y_1, Y_2 such that $\{(X, Y_1), (X, Y_2)\} \subseteq g$. The goal is to show $Y_1 = Y_2$. By 1.9.1.5, consider x_1, x_2, y_1, y_2 such that $(x_1, y_1) \in X \times Y_1 \cap f$ and $(x_2, y_2) \in X \times Y_2 \cap f$. Since X is an equivalence class of E, this implies $(x_1, x_2) \in E$ which in turn, by 1.9.1.1, gives $(y_1, y_2) \in F$. Hence y_1 and y_2 must belong to the same equivalence class of F, which gives $Y_1 = Y_2$.
2. Let $X \in \operatorname{ran} \pi_E$. X being an equivalence class of the non empty equivalence relation E, there is $x \in X \subseteq \operatorname{dom} E$. Set

$$\begin{aligned} y :=& f(x) \in \operatorname{dom} F \\ Y :=& \pi_F(y) \in \operatorname{ran} F. \end{aligned} \tag{1.4}$$

   Since $(x, y) \in f$ by (1.4), and $y \in Y$, we draw $(X, Y) \in g$ by 1.9.1.5.

□

Result 1.9.1.6 supplies a canonical construction to pass from a function on sets to a function on classes relative to equivalence relations respected by the original function. We want to carry this mechanism over to the case in which the function is $i(s)$ and the equivalence relation is given on U, where i is an interpretation of the language S, s is a symbol of it, and U is the universe of i. Since $i(s)$ is defined on $U^{|\#(s)|}$, we have to specify how to adapt some of the last definitions to tuples. First of all, we formally specify the natural way to pass from a relation over sets to a relation over tuples:

Definition 1.9.1.7 (Tupled relation)**.** Let O be a non empty relation, and n a natural number. We set

$$O^{[n]} := \{(p,q) \in (\operatorname{dom} O)^n \times ((\operatorname{ran} O)^n) : q \subseteq p \bullet O\}.$$

Now, we want to combine the quotient defined in 1.9.1.5 with the construction of 1.9.1.7 to obtain a quotient operating on interpretations. A technical nuisance stands on our way, though: when quotienting by a tupled relation, we are left with a function acting on classes of equivalence of tuples, while an interpretation should act on tuples (of equivalence classes, in this case). So we have to provide an object translating between these two types:

Definition 1.9.1.8. Let P be a relation, n be a natural number. Set

$$\eta_{P,n} := \left(\left(\pi_P^{-1}\right)^{[n]}\right) \bullet \pi_{P^{[n]}}.$$

It can finally be plugged into the following definiens:

Definition 1.9.1.9 (Quotient interpretation)**.** Given an interpretation i and a relation P, set

$$\frac{i}{P} := \operatorname{dom} \# \ni s \mapsto \begin{cases} \eta_{P,|\#(s)|} \bullet \frac{i(s)}{P^{[|\#(s)|]}} P & \#(s) \geq 0 \\ \eta_{P,|\#(s)|} \bullet \frac{i(s)}{P^{[|\#(s)|]}} \{(0,0),(1,1)\} \bullet \} \{ & \#(s) < 0. \end{cases}$$

Now we have to put forward some requests to make the quotient in 1.9.1.9 actually an interpretation:

Definition 1.9.1.10. Given an interpretation i of the language S, having U as universe, we say that i and the relation P are compatible if

$$\forall s \in \operatorname{dom} \# \quad \begin{cases} i(s) \text{ is } \left(P^{[\#(s)]}, P\right)\text{-compatible} & \#(s) \geq 0 \\ i(s) \text{ is } \left(P^{[-\#(s)]}, \{(0,0),(1,1)\}\right)\text{-compatible} & \#(s) < 0 \end{cases}$$

Proposition 1.9.1.11. *Given an interpretation i of the language S having universe U, and an equivalence relation E on U such that i and E are compatible, $\frac{i}{E}$ is an interpretation of S having* $\operatorname{ran}(\pi_E)$ *as universe.*

Proof. Set $I := \frac{i}{E}$. Let $s \in \operatorname{dom} \#_S$; set $n := |\#(s)| \in \mathbb{N}$, $f := i(s)$, $\overline{E} := E^{[n]}$ and $\eta := \eta_{E,n}$. One easily realizes (or may refer to the Mizar article `FOMODEL3.MIZ` to find the proofs) that $\overline{E}$ is an equivalence relation on U^n and that

$$\eta : (\operatorname{ran} \pi_E)^n \to \operatorname{ran} \pi_{\overline{E}}. \tag{1.5}$$

We show that I, s and $\operatorname{ran} \pi_E$ satisfy 1.8.0.20. By cases

$\#(s) \geq 0$ Then $I(s) = \eta_{E,n} \bullet \frac{f}{\overline{E}\ E}$ and $f : U^n \to U$. The goal is to prove that $I(s) : (\operatorname{ran} \pi_E)^n \to \operatorname{ran} \pi_E$. By 1.9.1.10, f is $\left(\overline{E}, E\right)$-compatible, so that $\frac{f}{\overline{E}\ E} : \operatorname{ran} \pi_{\overline{E}} \to \operatorname{ran} \pi_E$ by 1.9.1.6. This yields thesis by (1.5).

$\#(s) < 0$ Then $I(s) = \eta \bullet \frac{f}{\overline{E}\ \mathcal{I}_2}$ and $f : U^n \to 2$. The goal is to prove that $I(s) : (\operatorname{ran}\pi_E)^n \to 2$. By 1.9.1.10, f is $\left(\overline{E}, \mathcal{I}_2\right)$-compatible, so that $\frac{f}{\overline{E}\ \mathcal{I}_2} : \operatorname{ran}\pi_{\overline{E}} \to \operatorname{ran}\pi_{\mathcal{I}_2}$ by 1.9.1.6. This yields thesis by (1.5), being $\}\{ \quad : \operatorname{ran}\pi_{\mathcal{I}_2} = \{\{0\},\{1\}\} \to 2$.

□

Result 1.9.1.11 ends this section. Wanting to apply it to the free interpretation, in the next section we introduce a relation on terms, and investigate the conditions to make it an equivalence relation, as required by 1.9.1.11. In the subsequent section, we finally face the issue of compatibility.

1.9.2 The equability relation on terms and the Henkin interpretation

Definition 1.9.2.1. Given a ruleset D and a set X, we define

$$\overset{D}{\underset{X}{\sim}} := \left(*|_{T_S \times T_S}\right)^{-1} \left[\left(_{\{(0,\equiv)\}}*\right)^{-1} [D(X)]\right].$$

Remark 1.9.2.2. Since

$$\overset{D}{\underset{X}{\sim}} = \left\{(t_1, t_2) \in T_S \times T_S : X \left|\frac{}{D}\right. \equiv t_1 t_2\right\}, \tag{1.6}$$

$\overset{D}{\underset{X}{\sim}}$ is a relation on T_S.

Definition 1.9.2.3 (The Henkin 'interpretation')**.** $\mathcal{H}_{D,X} := \frac{\Phi_X}{\overset{D}{\underset{X}{\sim}}}$.

Proposition 1.9.2.4. *If $D \geq_\emptyset \{R_=\}$, then $\operatorname{dom}\overset{D}{\underset{X}{\sim}} = T_S$ and $\overset{D}{\underset{X}{\sim}}$ is reflexive.*

Proof. Set $D_0 := \{R_=\}$, $P := \overset{D}{\underset{X}{\sim}}$. Let t be a term. We have to show that $(t,t) \in P$. Now

$$(\emptyset, \equiv tt) \in R_=(\emptyset) \subseteq \overline{D_0}(\emptyset) \subseteq \overline{D_0}^{(\infty)}(\emptyset) \subseteq \overline{D}^{(\infty)}(\emptyset),$$

which shows that $X \left|\frac{}{D}\right. \equiv tt$ by 1.5.0.6, and hence thesis by virtue of (1.6). □

Proposition 1.9.2.5. *If $D \geq_\emptyset \left\{R_{\underset{=}{\leftrightarrow}}\right\}$ and X is D-closed, then $\overset{D}{\underset{X}{\sim}}$ is symmetric.*

Proof. Set $D_0 := \{R_=\}$. Assume $X \left|\frac{}{D}\right. \equiv t_1 t_2$. We have to show $X \left|\frac{}{D}\right. \equiv t_2 t_1$.

$$(\{\equiv t_1 t_2\}, \equiv t_2 t_1) \in R_{\underset{=}{\leftrightarrow}}(\emptyset) = \overline{D_0}(\emptyset) \subseteq \overline{D_0}^{(\infty)}(\emptyset) \subseteq \overline{D}^{(\infty)}(\emptyset),$$

and closure yields $\equiv t_1 t_2 \in X$. Hence $X \left|\frac{}{D}\right. \equiv t_2 t_1$ by 1.5.0.6. □

Proposition 1.9.2.6. *If $D \geq_\emptyset \left\{R_{\underset{=}{\Rightarrow}}\right\}$ and X is D-closed, then $\overset{D}{\underset{X}{\sim}}$ is transitive.*

Proof. Set $D_0 := \left\{R_{\Rightarrow}^{=}\right\}$. Assume $X \vdash_{\overline{D}} \equiv t_1 t_2$ and $X \vdash_{\overline{D}} \equiv t_2 t_3$. We have to show $X \vdash_{\overline{D}} \equiv t_1 t_3$.

$$(\{\equiv t_1 t_2, \equiv t_2 t_3\}, \equiv t_1 t_3) \in R_{\Rightarrow}^{=}(\emptyset) = \overline{D_0}(\emptyset) \subseteq \overline{D_0}^{(\infty)}(\emptyset) \subseteq \overline{D}^{(\infty)}(\emptyset),$$

and closure yields $\{\equiv t_1 t_2, \equiv t_2 t_3\} \subseteq X$. Hence $X \vdash_{\overline{D}} \equiv t_1 t_3$ by 1.5.0.6. □

Lemma 1.9.2.7. *If $D \geq_\emptyset \{R_=\}$, $D \geq_\emptyset \left\{R_{\Leftrightarrow}^{=}\right\}$, $D \geq_\emptyset \left\{R_{\Rightarrow}^{=}\right\}$ and X is D-closed, then $\overset{D}{\underset{X}{\sim}}$ is an equivalence relation on T_S.*

Proof. Immediate from 1.9.2.4, 1.9.2.5, 1.9.2.6. □

1.9.3 Compatibility

Lemma 1.9.3.1. *If $D \geq_\emptyset \{R_=\}$, X is D-closed, $D \geq_\emptyset \{R_+\}$, X is $\{R_\mathcal{R}\}$-closed, X is $\left\{R_{\Leftrightarrow}^{=}\right\}$-closed, then Φ_X and $\overset{D}{\underset{X}{\sim}}$ are compatible.*

Proof. Take $s \in \operatorname{dom} \#$. Set $P := \overset{D}{\underset{X}{\sim}}$ and $f := \Phi_X(s)$. By cases.

1) $\#(s) = 0$

By 1.9.1.10, we have to show that f is $\left(P^{[0]}, P\right)$-compatible. Since $P^{[0]} = \{(\emptyset, \emptyset)\}$, it suffices to show that $(f(\emptyset), f(\emptyset)) \in P$. $f(\emptyset)$ is in the universe T_S of Φ_X (see 1.8.0.23); hence, since $\operatorname{dom} P = T_S$ and P is reflexive by 1.9.2.4 and the hypothesis $D \geq_\emptyset \{R_=\}$, we have thesis.

2) $\#(s) > 0$

By 1.9.1.10, we have to show that f is $\left(P^{[n]}, P\right)$-compatible, where we set $n := \#(s) \in \mathbb{Z}^+$. As from 1.9.1.1, let $\mathbf{t}, \mathbf{t}' \in (T_S)^n$, and assume $(\mathbf{t}, \mathbf{t}') \in P^{[n]}$. The goal is to prove $(f(\mathbf{t}), f(\mathbf{t}')) \in P$. Set $\Gamma := \{\equiv \mathbf{t}(j)'(j) : j \in n\} \in 2_n^{F_S}$ and $\varphi := \equiv s{*}{*}(\mathbf{t})\, s{*}{*}(\mathbf{t}') \equiv f(\mathbf{t})\, f(\mathbf{t}')$. From

$$(\Gamma, \varphi) \in R_+(\emptyset) = \overline{\{R_+\}}(\emptyset) \subseteq \overline{\{R_+\}}^{(\infty)}(\emptyset) \subseteq \overline{D}^{(\infty)}(\emptyset),$$

which takes advantage of the hypothesis $D \geq_\emptyset \{R_+\}$, and

$$\begin{aligned} (\mathbf{t}, \mathbf{t}') \in P^{[n]} \Leftrightarrow \quad & \forall j \in n\, (tt(j), tt'(j)) \in P \Leftrightarrow \\ \forall j \in n X \vdash_{\overline{D}} \equiv tt(j)\, tt'(j) \Rightarrow \quad & \Gamma \subseteq X, \end{aligned}$$

where last deduction employed D-closure, we draw $X \vdash_{\overline{D}} \varphi$ thanks to 1.5.0.6.

3) $\#(s) < 0$

By 1.9.1.10, we have to show that f is $\left(P^{[n]}, \mathcal{I}_2\right)$-compatible, where we set $n := -\#(s) \in \mathbb{Z}^+$ and $\mathcal{I}_2 := \{(0,0), (1,1)\}$. As from 1.9.1.1, let $\mathbf{t}, \mathbf{t}' \in (T_S)^n$, and assume $(\mathbf{t}, \mathbf{t}') \in P^{[n]}$. The goal is to prove $(f(\mathbf{t}), f(\mathbf{t}')) \in \mathcal{I}_2$. Set $\Gamma := \{\equiv \mathbf{t}(j)'(j) : j \in n\} \in 2_n^{F_S}$, and preliminarily deduce

$$\begin{aligned} (\mathbf{t}, \mathbf{t}') \in P^{[n]} \Leftrightarrow \quad & \forall j \in n\, (tt(j), tt'(j)) \in P \Leftrightarrow \\ \forall j \in n X \vdash_{\overline{D}} \equiv tt(j)\, tt'(j) \Rightarrow \quad & \Gamma \subseteq X \end{aligned} \tag{1.7}$$

thanks to D-closure. Now proceed by subcases.

a) $f(\mathbf{t}) = 1$ The thesis reduces to showing $f(\mathbf{t}') = 1$, which, by 1.8.0.24, means $\varphi' := s * * (\mathbf{t}') \in X$. Let $\varphi := s * * (\mathbf{t}) \in X$. The subcase assumption gives $\varphi \in X$ by 1.8.0.24, hence $\Gamma \cup \{\varphi\} \subseteq X$ by (1.7). Also,

$$(\Gamma \cup \{\varphi\}, \varphi') \in R_{\mathcal{R}}(\emptyset) = \overline{R_{\mathcal{R}}}(\emptyset) \subseteq \overline{\{R_{\mathcal{R}}\}}^{(\infty)}(\emptyset).$$

Thus $X \vdash_{\overline{\{R_{\mathcal{R}}\}}} \varphi'$. By $\{R_{\mathcal{R}}\}$-closure, we are finished.

b) $f(\mathbf{t}) = 0$ Thesis reduces to showing $f(\mathbf{t}') = 0$, which, by 1.8.0.24, means $\varphi' := s * * (\mathbf{t}') \notin X$. By contradiction, assume

$$\varphi' \in X. \tag{1.8}$$

Set $\Gamma' := \{\equiv \mathbf{t}'(j)\, \mathbf{t}(j) : j \in n\}$. Given $j \in n$, it is easily seen that $X \vdash_{\overline{\left\{R_{\overset{\leftrightarrow}{=}}\right\}}} \equiv \mathbf{t}'(j)\, \mathbf{t}(j)$, since $\{\equiv \mathbf{t}(j)\, \mathbf{t}'(j)\} \subseteq X$ by (1.7), and

$$(\{\equiv \mathbf{t}(j)\, \mathbf{t}'(j)\}, \equiv \mathbf{t}'(j)\, \mathbf{t}(j)) \in R_{\overset{\leftrightarrow}{=}}(\emptyset) = \overline{\left\{R_{\overset{\leftrightarrow}{=}}\right\}}(\emptyset) \subseteq \overline{\left\{R_{\overset{\leftrightarrow}{=}}\right\}}^{(\infty)}(\emptyset).$$

By $\left\{R_{\overset{\leftrightarrow}{=}}\right\}$-closure, we conclude that $\Gamma' \subseteq X$, and hence that $\Gamma' \cup \{\varphi'\} \subseteq X$ by (1.8). Moreover,

$$(\Gamma' \cup \{\varphi'\}, \varphi) \in R_{\mathcal{R}}(\emptyset) = \overline{R_{\mathcal{R}}}(\emptyset) \subseteq \overline{\{R_{\mathcal{R}}\}}^{(\infty)}(\emptyset),$$

yielding $X \vdash_{\overline{\{R_{\mathcal{R}}\}}} \varphi$, and hence $\varphi \in X$ by $\{R_{\mathcal{R}}\}$-closure, contradicting $f(\mathbf{t}) = 0$.

□

Corollary 1.9.3.2. *If* $D \geq_{\emptyset} \left\{R_{=}, R_{\overset{\leftrightarrow}{=}}, R_{\overset{\Rightarrow}{=}}, R_{+}, R_{\mathcal{R}}\right\}$ *and* $D(X) \subseteq X$*, then* $\overset{D}{\underset{X}{\sim}}$ *and* Φ_X *are compatible.*

Corollary 1.9.3.3 (of 1.9.3.2 and 1.9.1.11)**.** *If* $D \geq_{\emptyset} \left\{R_{=}, R_{\overset{\leftrightarrow}{=}}, R_{\overset{\Rightarrow}{=}}, R_{+}, R_{\mathcal{R}}\right\}$ *and* $D(X) \subseteq X$*, then* $\mathcal{H}_{D,X}$ *is an interpretation having* $T_S/\overset{D}{\underset{X}{\sim}}$ *as universe.*

1.9.4 The Henkin model

Here, the conditions making $\mathcal{H}_{D,X}$ a model of X are studied. We first work out two preparatory results.

Lemma 1.9.4.1. *Let* i *be an interpretation of* S*, and* P *an equivalence relation over its universe* U *such that* i *and* P *are compatible. Then*

$$\overline{\left(\frac{i}{P}\right)}\Big|_{T_S} = \pi_P \circ \bar{i}|_{T_S} \qquad \text{and}$$

$$\overline{\left(\frac{i}{P}\right)}(\varphi_0) = \bar{i}(\varphi_0) \qquad \text{if } \varphi_0(0) \in \#^{-1}[\mathbb{Z}^-] \setminus \{\equiv\}.$$

Proof. Set $I := \frac{i}{P}$. Let us show that

$$\overline{I}\Big|_{T_{S,n}} = \pi_P \circ \overline{i}|_{T_{S,n}} \tag{1.9}$$

for every $n \in \mathbb{N}$ by complete induction on n. For the case $n = 0$, consider $t_0 \in T_{S,0}$; the goal equation is $\overline{I}(t_0) = \pi_P(\overline{i}(t_0))$. Set $v := t_0(0)$, $f := i(v)$ and reason as follows:

$$\begin{aligned}\overline{I}(t_0) &\overset{1.8.0.26}{=} (I(v))(0) \overset{1.9.1.9}{=} \left(\frac{f}{P^{[0]}\ P} \circ \eta_{P,0}\right)(0) = \left(\frac{f}{\mathcal{I}_1\ P} \circ \{(0,\{0\})\}\right)(0) \\ &= \frac{f}{\mathcal{I}_1\ P}(\{(0,\{0\})\}(0)) = \frac{f}{\mathcal{I}_1\ P}(\{0\}) = \pi_P(f(0)).\end{aligned}$$

Now assume (1.9) holds for every $n \leq m$. Let us prove that it holds for $n = m+1$. Considered arbitrary $t \in T_{S,m+1}$, it suffices to show $\overline{I}(t) = \pi_P(\overline{i}(t))$. Set $s := t(0)$, $k := \#(s)$, $f := i(s)$. We can assume $k > 0$; then

$$\begin{aligned}\overline{I}(t) &\overset{1.8.0.26}{=} I(s)\left(\overline{I} \circ \overrightarrow{t}\right) \overset{!}{=} I(s)\left(\pi_P \circ \overline{i} \circ \overrightarrow{t}\right) \overset{1.9.1.9}{=} \left(\frac{f}{P^{[k]}\ P} \circ \eta_{P,k}\right)\left(\pi_P \circ \overline{i} \circ \overrightarrow{t}\right) \\ &= \left(\frac{f}{P^{[k]}\ P} \circ \eta_{P,k}\right)\left((\pi_P)^{[k]}\left(\overline{i} \circ \overrightarrow{t}\right)\right) = \left(\frac{f}{P^{[k]}\ P} \circ \eta_{P,k} \circ (\pi_P)^{[k]}\right)\left(\overline{i} \circ \overrightarrow{t}\right) \\ &= \left(\frac{f}{P^{[k]}\ P} \circ \pi_{P^{[k]}}\right)\left(\overline{i} \circ \overrightarrow{t}\right) \overset{!!}{=} (\pi_P \circ f)\left(\overline{i} \circ \overrightarrow{t}\right) = \pi_P\left(f\left(\overline{i} \circ \overrightarrow{t}\right)\right) \overset{1.9.1.9}{=} \pi_P(\overline{i}(t)).\end{aligned}$$

! denotes the step employing inductive hypothesis. !! denotes the spot where compatibility has been used. This secures the first thesis.

Finally, set $r := \varphi_0(0)$, $l := -\#(r) \in \mathbb{Z}^+$ and $g := i(r)$:

$$\begin{aligned}\overline{I}(\varphi_0) &= (I(r))\left(\overline{I} \circ \overrightarrow{\varphi_0}\right) \overset{!}{=} (I(r))\left(\pi_P \circ \overline{i} \circ \overrightarrow{\varphi_0}\right) = (I(r))\left((\pi_P)^{[l]}\left(\overline{i} \circ \overrightarrow{\varphi_0}\right)\right) \\ &\overset{1.9.1.9}{=} \left(\}\{ \circ \frac{g}{P^{[l]}\ \mathcal{I}_2} \circ \eta_{P,l} \circ (\pi_P)^{[l]}\right)\left(\overline{i} \circ \overrightarrow{\varphi_0}\right) = \left((\}\{_2) \circ \frac{g}{P^{[l]}\ \mathcal{I}_2} \circ \pi_{P^{[l]}}\right)\left(\overline{i} \circ \overrightarrow{\varphi_0}\right) \\ &\overset{!!}{=} ((\}\{_2) \circ \pi_{\mathcal{I}_2} \circ g)\left(\overline{i} \circ \overrightarrow{\varphi_0}\right) = g\left(\overline{i} \circ \overrightarrow{\varphi_0}\right).\end{aligned}$$

Last equality is due to $\}\{_2 = \pi_{\mathcal{I}_2}^{-1}$. In the passage marked by '!', the freshly proved first thesis were employed. '!!' denotes the step employing compatibility. □

Lemma 1.9.4.2. $\overline{\Phi_X}\Big|_{T_S} = \mathcal{I}_{T_S}$.

Proof. Let us show

$$\overline{\Phi_X}\Big|_{T_{S,n}} = \mathcal{I}_{T_{S,n}} \qquad \forall n \in \mathbb{N} \tag{1.10}$$

by complete induction on n. For the case $n = 0$, consider $t_0 \in T_{S,0}$, and set $v := t_0(0)$.

$$\begin{aligned}\overline{\Phi_X}(t_0) &\overset{1.8.0.26}{=} \Phi_X(v)(0) \overset{1.8.0.24}{=} \left(_{\{(0,v)\}}*\right) \circ \left(**|_{(T_S{}^0)}\right)(0) \\ &= \left(_{\{(0,v)\}}*\right)\left(\left(**|_{\{0\}}\right)(0)\right) = \{(0,v)\} * \emptyset = t_0.\end{aligned}$$

Now, assume (1.10) is verified for every $n \leq m+1$, and consider $t \in T_{S,m+1}$. Set $s := t(0)$, $k := \#(t) \in \mathbb{N}$. We can assume $k > 0$, and have to show that $\overline{\Phi_X}(t) = t$:

$$\overline{\Phi_X}(t) \overset{1.8.0.26}{=} (\Phi_X(s))\left(\overline{\Phi_X} \circ \overrightarrow{t}\right) \overset{!}{=} (\Phi_X(s))\left(\overrightarrow{t}\right)$$
$$\overset{1.8.0.24}{=} \left(_{\{(0,s)\}}*\right)\left(\left(**|_{(T_S{}^k)}\right)\left(\overrightarrow{t}\right)\right) = \{(0,s)\} * \left(**\left(\overrightarrow{t}\right)\right) = t.$$

'!' denotes the induction step. □

Now we see that, when restricting to atomic formulas, one actually needs to impose very little additional requests for $\mathcal{H}_{D,X}$ to be a model, besides those from 1.9.3.3 making it an interpretation:

Theorem 1.9.4.3. *If* $D \geq_\emptyset \left\{R_0, R_=, R_{\overset{\leftrightarrow}{=}}, R_{\overset{\Rightarrow}{=}}, R_+, R_{\mathcal{R}}\right\}$ *and* $D(X) \subseteq X$, *then*

$$\overline{\mathcal{H}_{D,X}}\Big|_{F_{S,0}} = 1_X^{F_{S,0}}.$$

Proof. We set $i := \Phi_X$, $P := \overset{D}{\underset{X}{\sim}}$, $I := \mathcal{H}_{D,X} = \frac{i}{P}$. Let $\varphi_0 \in F_{S,0}$, and set $r := \varphi_0(0)$, $n := -\#(r) \in \mathbb{Z}^+$. By cases.
Case $r \neq \equiv$**:**

$$\overline{I}(\varphi_0) \overset{1.9.3.2,\ 1.9.4.1}{=} \overline{i}(\varphi_0) \overset{1.8.0.26}{=} (i(r))(\overline{i} \circ \overrightarrow{\varphi_0}) \overset{1.9.4.2}{=} (i(r))(\overrightarrow{\varphi_0}) \overset{1.8.0.24}{=}$$
$$1_X^{F_{S,0}} \circ \left(_{\{(0,r)\}}*\right) \circ \left(**|_{T_S^n}\right)(\overrightarrow{\varphi_0}) = 1_X^{F_{S,0}} \circ \left(\left(_{\{(0,r)\}}*\right) \circ **\right)(\overrightarrow{\varphi_0}) =$$
$$1_X^{F_{S,0}}\left(\left(_{\{(0,r)\}}*\right)(**(\overrightarrow{\varphi_0}))\right) = 1_X^{F_{S,0}}(*(\{(0,r)\}, **(\overrightarrow{\varphi_0}))) = 1_X^{F_{S,0}}(\varphi_0).$$

Case $r = \equiv$**:**
Set $t_1 := \overrightarrow{\varphi_0}(0)$, $t_2 := \overrightarrow{\varphi_0}(1)$.

$$\overline{I}(\varphi_0) = 1 \overset{1.8.0.26}{\Leftrightarrow} \overline{I}(t_1) = \overline{I}(t_2) \Leftrightarrow \pi_P(\overline{i}(t_1)) = \pi_P(\overline{i}(t_2)) \overset{1.9.4.2}{\Leftrightarrow}$$
$$\pi_P(t_1) = \pi_P(t_2) \overset{1.9.2.1}{\Leftrightarrow} X \left|\overline{\vdash_D} \equiv t_1 t_2 \Leftrightarrow \equiv t_1 t_2 = \varphi_0 \in X.\right.$$

Last equivalence is due to D-closure ($\Rightarrow$) and to $D \geq_\emptyset \{R_0\}$ ($\Leftarrow$). □

The ultimate goal of this section is the extension of 1.9.4.3 to the whole F_S. To this end, we will need to employ a couple of auxiliary results significant in their own right, as relating the syntactical constructions of simple substitution and term substitution (defined in 1.1.0.8 and 1.8.0.32) to the semantical one of reassignment (defined in 1.8.0.25):

Lemma 1.9.4.4.

$$\overline{\frac{u}{v_1} i}(\psi) = \overline{\frac{u}{v_2} i}\left(\frac{v_2}{v_1}\psi\right),$$

where u *is an element of the universe of the interpretation* i *and* $v_2 \notin \operatorname{ran}\psi$.

Proof. Denote with S the language we are working in, with A the symbol set of S, and with U the universe of i. Set $i_1 := \frac{u}{v_1} i$, $i_2 := \frac{u}{v_2} i$, $f_1 := \overline{i_1}$, $f_2 := \overline{i_2}$, $g := \mathcal{I}_A \lhd \{(v_1, v_2)\}$, $B := A \setminus \{v_2\}$. We start with showing that

$$f_1(t') = f_2(g \circ t') \qquad \forall t' \in T_{S,n} \cap B^* \tag{1.11}$$

by complete induction on n. The case $n = 0$ is trivial, and anyway is treated in MML article `FOMODEL3`, at the label `Lm44`. Now suppose (1.11) holds for every $n \leq m$, and consider $t \in T$ such that $|t| \leq m + 1$. Set $s := t(0)$. We can assume

$$\#(s) > 0 \tag{1.12}$$

and complete the proof of (1.11) as from the following iterative equation

$$f_2(g \circ t) \overset{1.8.0.26}{=} (i_2(s)) \left(f_2 \circ \overrightarrow{g \circ t}\right) = (i_1(s)) \left(f_2 \circ \overrightarrow{g \circ t}\right) = (i_1(s)) \left(f_1 \circ \overrightarrow{t}\right),$$

whose last step rests on inductive hypothesis applied to (1.11). The immediately preceding step is due to the fact that (1.12) implies $s \notin \{v_1, v_2\}$. Similarly, one can show that

$$f_1(\psi_0) = f_2(g \circ \psi_0) \qquad \forall \psi_0 \in F_{S,0} \cap B^*. \tag{1.13}$$

To avoid repetitions, we refer the interested reader to `FOMODEL3:Lm45` for the proof of (1.13). At last, we show

$$f_1(\psi') = f_2(g \circ \psi') \qquad \forall \psi' \in B^* \cap F_{S,n} \tag{1.14}$$

by complete induction on n. The case $n = 0$ is given by (1.13). Let us then assume (1.14) for every $n \leq m$, and consider $\psi \in B^* \cap F_{S,m+1}$. We can assume as well $|\psi| > 0$ and set $s := \psi(0) \in \#^{-1}[\{0\}] \setminus \{v_2\} \cup \{\downarrow\}$. By cases.

Case 1): $s = \downarrow$
Then set $\psi_1 := \overrightarrow{\psi}(0)$, $\psi_2 := \overrightarrow{\psi}(1)$ and $N := 1^{2\times 2}_{\{(0,0)\}}$. We employ (1.14) via induction on the unmarked step of the following chain:

$$\begin{aligned} f_2(g \circ \psi) &\overset{1.8.0.27}{=} N((f_2(g \circ \psi_1), f_2(g \circ \psi_2))) \\ &= N((f_1(\psi_1), f_1(\psi_2))) \overset{1.8.0.27}{=} f_1(\psi). \end{aligned}$$

Case 2): $s \in \#^{-1}[\{0\}] \setminus \{v_2\}$
Then consider $\varphi \in B^* \cap F_{S,m}$ such that $\psi = s\varphi$. By subcases.

Subcase $s = v_1$**:**
Then $g \circ \psi = v_2 * (g \circ \varphi)$. Assume $f_2(g \circ \psi) = 1$. Then, by 1.8.0.27, consider $u' \in U$ such that

$$1 = \overline{\frac{u'}{v_2}\frac{u}{v_2} i}(g \circ \varphi) = \overline{\frac{u'}{v_2} i}(g \circ \varphi) \overset{(1.14)}{=} \overline{\frac{u'}{v_1} i}(\varphi) = \overline{\frac{u'}{v_1}\frac{u}{v_1} i}(\varphi).$$

Hence, again by 1.8.0.27, $\overline{\frac{u}{v_1} i}(v_1\varphi) = 1$. Analogously one shows $\overline{\frac{u}{v_1} i}(v_1\varphi) = 1 \Longrightarrow \overline{\frac{u}{v_2} i}(g \circ \psi) = 1$.

Subcase $s \neq v_1$:

Assume $f_2(g \circ \psi) = 1$. Then, by 1.8.0.27, consider $u' \in U$ such that

$$1 = \overline{\frac{u'}{s} i_2}(g \circ \varphi) = \overline{\frac{u'}{s}\frac{u}{v_2} i}(g \circ \varphi) = \overline{\frac{u}{v_2}\frac{u'}{s} i}(g \circ \varphi)$$
$$= \overline{\frac{u}{v_1}\frac{u'}{s} i}(\varphi) = \overline{\frac{u'}{s}\frac{u}{v_1} i}(\varphi).$$

Hence $\overline{\frac{u}{v_1} i} = 1$ by 1.8.0.27. In a similar way, one shows $1 = f_1(\psi) \implies f_2(g \circ \psi) = 1$.

□

Lemma 1.9.4.5 (Substitution lemma)**.** *Given v, t, φ:*

1. $|\varphi[v/t]| = |\varphi|$;
2. $\bar{i}(\varphi[v/t]) = \overline{\frac{\bar{i}(t)}{v} i}(\varphi)$, *for any interpretation i.*

Proof. See appendix A. □

Definition 1.9.4.6 (Witness)**.** Given a language S, consider the following relation on F_S:

$$W_S := \left\{ \left(\{(0, v_1)\} * \varphi, \frac{v_2}{v_1}\varphi \right) : v_1, v_2 \in \#^{-1}[\{0\}], \varphi \in F_S | \quad v_2 \notin \operatorname{ran} \varphi \right\}.$$

Often the context will allow to drop the subscript and write just W.

If $\varphi \in W_S[\{\psi\}]$, we say that φ is a *witness* for ψ.

A set X will be said to be *S-witnessed* (simply *witnessed* when the context is safe) if

$$X \cap \operatorname{dom} W_S \subseteq W_S^{-1}[X].$$

Definition 1.9.4.7. X is a minimal cover of the language S (or an S-mincover, or even just a mincover) if

$$\forall \varphi \in F_S (\varphi \in X \text{ if and only if } \downarrow \varphi\varphi \notin X).$$

Theorem 1.9.4.8 (Henkin's theorem)**.** *Suppose*

- $D \geq_\emptyset \left\{ R_0, R_=, R_{\underset{=}{\leftrightarrow}}, R_{\underset{=}{\Rightarrow}}, R_+, R_{\mathcal{R}}, R_\downarrow, R_{\underset{\exists}{\rightarrow}} \right\}$,
- *X is a mincover,*
- *$D(X) \subseteq X$, and*
- *X is witnessed.*

Then

$$\overline{\mathcal{H}_{D,X}}\Big|_F = 1_X^F.$$

Proof. Set $i := \Phi_X$, $P := \overset{D}{\underset{X}{\sim}}$, $I := \mathcal{H}_{D,X} = \frac{i}{P}$. We will prove

$$\overline{I}\Big|_{F_{S,m}} = 1_X^{F_{S,m}} \tag{1.15}$$

by complete induction on m. For $m = 0$, thesis is given by 1.9.4.3. Assume the inductive hypothesis: (1.15) holds for all $m \leq n$. Let $\psi \in F_{S,n+1}$. We have to show that

$$\overline{I}(\psi) = 1 \Leftrightarrow \psi \in X.$$

We can suppose $\psi \notin F_{S,0}$, and proceed by cases.

Case $\psi(0) \neq \downarrow$**:** Then consider v_1, φ such that $\psi = v_1\varphi$.

$$\begin{aligned}\overline{I}(\psi) = 1 \overset{1.8.0.27}{\Leftrightarrow} \exists t \in T|\quad 1 = \overline{\frac{\pi_P(t)}{v_1} I(\varphi)} &\overset{1.9.4.2}{=} \overline{\frac{\pi_P(\overline{i}(t))}{v_1} I(\varphi)} \\ &\overset{1.9.4.1,\ 1.9.3.2}{=} \overline{\frac{\overline{I}(t)}{v_1} I(\varphi)} \overset{1.9.4.5}{=} \overline{I}(\varphi[v_1/t]).\end{aligned} \tag{1.16}$$

$\Leftarrow$

Assume $\psi \in X$. Then consider $v_2 \in \#^{-1}[\{0\}] \setminus \operatorname{ran}\varphi$ such that $\frac{v_2}{v_1}\varphi \in X$ by 1.9.4.6. Since $\left|\frac{v_2}{v_1}\varphi\right| = |\varphi| < |\psi|$, we can trigger induction:

$$1 = \overline{I}\left(\frac{v_2}{v_1}\varphi\right) = \overline{\frac{(I(v_2))(\emptyset)}{v_2} I\left(\frac{v_2}{v_1}\varphi\right)} \overset{1.9.4.4}{=} \overline{\frac{(I(v_2))(\emptyset)}{v_1} I(\varphi)}.$$

Thesis follows from 1.8.0.27.

$\Rightarrow$

Assume $\overline{I}(\psi) = 1$ and, by (1.16), consider $\overline{t}|\quad \varphi[v_1/\overline{t}] \in X$.

$$(\{\varphi[v_1/\overline{t}]\}, v_1\varphi) \in R_{\overset{\rightarrow}{\exists}}(\emptyset) \subseteq \overline{\left\{R_{\overset{\rightarrow}{\exists}}\right\}}^{(\infty)}(\emptyset) \subseteq \overline{D}^{(\infty)}(\emptyset).$$

By D-closure, we draw $\psi \in X$.

Case $\psi(0) = \downarrow$**:** Set $\varphi_1 := \overrightarrow{\psi}(0)$, $\varphi_2 := \overrightarrow{\psi}(1)$.

$$\begin{aligned}\overline{I}(\psi) = 1 \overset{1.8.0.27}{\Leftrightarrow} \overline{I}(\varphi_1) = 0 = \overline{I}(\varphi_2) \Leftrightarrow \\ \{\varphi_1, \varphi_2\} \cap X = \emptyset \Leftrightarrow \{\downarrow \varphi_1\varphi_1, \downarrow \varphi_2\varphi_2\} \subseteq X,\end{aligned}$$

where last equivalence is due to mincover hypothesis, and previous one to inductive hypothesis. Hence we have reduced our task to showing that

$$\psi \in X \Leftrightarrow \{\downarrow \varphi_1\varphi_1, \downarrow \varphi_2\varphi_2\} \subseteq X.$$

$\Leftarrow$

Assume $\{\downarrow \varphi_1\varphi_1, \downarrow \varphi_2\varphi_2\} \subseteq X$. Since

$$(\{\downarrow \varphi_1\varphi_1, \downarrow \varphi_2\varphi_2\}, \psi) \in R_{\downarrow}(\emptyset) \subseteq \overline{\{R_{\downarrow}\}}^{(\infty)}(\emptyset) \subseteq \overline{D}^{(\infty)}(\emptyset),$$

thesis follows immediately from 1.5.0.6 and D-closure hypothesis.

$\Rightarrow$

Assume $\psi \in X$. Set $\psi' :=\downarrow \varphi_2\varphi_1$. Now

$$\{\psi\} \left|\!\!\underset{\{R_\downarrow\}}{\vdash} \psi' \Longrightarrow \{\psi\} \left|\!\!\underset{D}{\vdash} \psi',\right.\right.$$

where the implication is given by $D \geq_\emptyset \{R_\downarrow\}$. By D-closure, we conclude $\{\psi, \psi'\} \subseteq X$. This, together with

$$\begin{aligned}\left(\{\psi, \psi'\}, \downarrow \varphi_1\varphi_1\right), \left(\{\psi, \psi'\}, \downarrow \varphi_2\varphi_2\right) &\in R_\downarrow(\emptyset) \\ &\subseteq \overline{\{R_\downarrow\}}^{(\infty)}(\emptyset) \subseteq \overline{D}^{(\infty)}(\emptyset),\end{aligned}$$

ends the proof by virtue of D-closure.

□

Remark 1.9.4.9. It is readily checked that in proof of 1.9.4.8, the following slightly weaker flavor of $R_\downarrow$ would suffice:

$$\begin{aligned}G(S) \supseteq \Sigma \mapsto \{(\Gamma, \varphi) : \exists \varphi_1, \varphi_2, \varphi_3, \varphi_4 \in F_S | \Gamma = \{\downarrow \varphi_1\varphi_2, \downarrow \varphi_3\varphi_4\} \\ \text{and } \varphi =\downarrow \varphi_2\varphi_3 \text{ and } |\{\varphi_1, \varphi_2, \varphi_3, \varphi_4\}| \leq 2\}.\end{aligned}$$

Since all the forthcoming results requiring $R_\downarrow$ do so precisely to invoke 1.9.4.8, the same goes for them. We adopt $R_\downarrow$ mainly because it is more straightly put into a diagram than its variant above.

1.10 Enlarging sets of formulas

In this section we study how to enlarge a given set X of formulas to make it

- closed with respect to a given ruleset D and
- witnessed,

so that the enlargement can be applied 1.9.4.8: in particular, this automatically supplies a model for X, which is our ultimate goal. We shall investigate the conditions X and D must obey to perform this operation. Also, we will restrict to countable languages to more easily develop constructive methods to build the two distinct enlargements corresponding to the points of the above checklist. The rub is how to combine sequentially the two enlargements avoiding the second cancelling the effect of the first. The property of the witness subjoining construction expressed by 1.10.2.4 and deployed in 1.11.0.6 will be the key.

1.10.1 Preliminaries

Definition 1.10.1.1. Consider the following element of $(F_S)^{F_S}$:

$$\neg_S : \varphi \mapsto \downarrow \varphi\varphi.$$

Notation 1.10.1.2. Again, we can drop the subscript in $\neg_S$ when it is safe to do so. Also, we will usually write $\neg\varphi$ instead of $\neg(\varphi)$:

$$\neg\varphi =\downarrow \varphi\varphi,$$

and $\neg^{(n)}\varphi$ instead of $\neg^{(n)}(\varphi)$.

Definition 1.10.1.3 (Forms of consistency)**.** X is said to be S-consistent (or syntactically consistent when the context is clear) if

$$X \cap \neg_S^{-1}[X] = \emptyset.$$

X is S-inconsistent (syntactically inconsistent) if it is not S-consistent. It is said to be an S-cover (or just a cover) if

$$X \cup \neg_S^{-1}[X] \supseteq F_S.$$

It is termed D-consistent (or just consistent when no ambiguity can arise) if $D(X)$ is S-consistent, otherwise we say it is D-inconsistent (inconsistent): we write $\mathrm{Con}_D(X)$ and $\mathrm{Inc}_D(X)$, respectively.

Remark 1.10.1.4. X is a mincover if and only if X is a syntactically consistent cover.

Definition 1.10.1.5. A ruleset D is said to be *weakly assumptive* if any D-consistent cover is a D-closed mincover.

Definition 1.10.1.6. A ruleset D is said to be *strongly assumptive* if for any D-consistent cover X it holds $D(X) = X \cap F_S$.

Remark 1.10.1.7. Any strongly assumptive ruleset is weakly assumptive.

Proposition 1.10.1.8. *$\{R_0\}$ is strongly assumptive.*

Proof. Let X be a $\{R_0\}$-consistent cover. We can assume $X \subseteq F_S$. Of course, being $(\{\psi\}, \psi) \in R_0(\emptyset)$ for any $\psi \in F_S$, one has, in particular, that $X \subseteq (\{R_0\})(X)$. Hence it remains to show that $(\{R_0\})(X) \subseteq X$. Assume $\psi \in (\{R_0\})(X)$. Then consider, by 1.5.0.6, $n \in \mathbb{N}$ and a finite $\Gamma \subseteq X$ such that $(\Gamma, \psi) \in \overline{\{R_0\}}^{(n+1)}(\emptyset) = R_0\left(\overline{\{R_0\}}^{(n)}(\emptyset)\right)$. This gives $\Gamma = \{\psi\}$ by definition of R_0. Hence thesis. □

Proposition 1.10.1.9. *If D_1 is strongly assumptive and $D_2 \geq_\emptyset D_1$, then D_2 is strongly assumptive.*

Proof. Given a D_2-consistent cover $X \subseteq F_S$, we must show that $D_2(X) = X$. First $X = D_1(X) \subseteq D_2(X)$. To show the reverse inclusion, $D_2(X) \subseteq X$, consider φ and suppose $\varphi \in D_2(X)$:

$$\varphi \in D_2(X) \Longrightarrow\downarrow \varphi\varphi \notin D_2(X) \Longrightarrow\downarrow \varphi\varphi \notin D_1 \Longrightarrow\downarrow \varphi\varphi \notin X \Longrightarrow \varphi \in X.$$

First implication is due to consistency, and last one to X being a cover. □

Definition 1.10.1.10. A ruleset D is cut-like if $\mathrm{Inc}_D(X \cup \{\varphi\})$ implies $X \overset{\downarrow}{\vdash}_D \varphi\varphi$ for every X, φ.

1.10.2 Witness-subjoining construction for countable languages

Definition 1.10.2.1. Given X, D and two mappings

$$l : \mathbb{N} \ni n \mapsto v_n \in \#^{-1}[\{0\}]$$
$$f : \mathbb{N} \ni n \mapsto \varphi_n \in F_S,$$

define recursively

$$X_0 := X$$
$$a_n := \#^{-1}[\{0\}] \setminus \lfloor X_n \cup \{\varphi_n\} \rfloor$$
$$X_{n+1} := \begin{cases} X_n \cup \left\{ \frac{l\left(\min l^{-1}[a_n]\right)}{v_n} \varphi_n \right\} & \text{if } \mathrm{Con}_D(X_n \cup \{v_n \varphi_n\}), \\ & X_n \cap W[\{v_n \varphi_n\}] = \emptyset \text{ and } a_n \neq \emptyset \\ X_n & \text{otherwise}, \end{cases}$$

and finally

$$\mathcal{W}_D^{l,f}(X) := \bigcup_{n \in \mathbb{N}} X_n.$$

Lemma 1.10.2.2. *Assume that*

1. *D is cut-like;*
2. *$D \geq_\emptyset \{R_=\}$;*
3. *$R_{\underset{\exists}{\leftarrow}} \in D$.*

If $\mathrm{Con}_D(X)$, *then* $\mathrm{Con}_D\left(\mathcal{W}_D^{l,f}(X)\right)$.

Proof. Suppose $\mathrm{Inc}_D\left(\mathcal{W}_D^{l,f}(X)\right)$. Then, referring to the objects introduced in 1.10.2.1, we can take the minimum m of the non empty subset of $\mathbb{N}$:

$$\{n \in \mathbb{N} | \mathrm{Inc}_D(X_n)\}.$$

If $m = 0$, then we are done. Otherwise, consider $k \in \mathbb{N} | m = k + 1$. Having set

$$v_k := l(k)$$
$$\varphi_k := f(k)$$
$$v_k' := l\left(\min l^{-1}[a_k]\right),$$

from definition 1.10.2.1 and that of minimum we must draw

$$\mathrm{Con}_D(X_k \cup \{v_k \varphi_k\}) \tag{1.17}$$
$$a_k \neq \emptyset \tag{1.18}$$
$$\mathrm{Inc}_D\left(X_k \cup \left\{\frac{v_k'}{v_k}\varphi_k\right\}\right). \tag{1.19}$$

By the last fact, we also have $\mathrm{Inc}_D\left(X_k \cup \left\{\frac{v_k'}{v_k}\varphi_k\right\} \cup \{\equiv \overline{vv}\}\right)$, which gives $X_k \cup \left\{\frac{v_k'}{v_k}\varphi_k\right\} \vdash_{\overline{D}\downarrow} \equiv \overline{vv} \equiv \overline{vv}$ by hypothesis (1). Hence consider a finite set of formulas

$\Gamma \subseteq X_k \cup \left\{\frac{v'_k}{v_k}\varphi_k\right\}$ such that $(\Gamma, \downarrow \equiv \overline{vv} \equiv \overline{vv}) \in \overline{D}^{(m+1)}(\emptyset)$ for some $m \in \mathbb{N}$. This in particular implies $\Gamma \vdash_{\overline{D}} \downarrow \equiv \overline{vv} \equiv \overline{vv}$; since it is also true that $\Gamma \vdash_{\overline{D}} \equiv \overline{vv}$ by hypothesis (2), it must be $\Gamma \not\subseteq X_k$, because $\mathrm{Con}_D(X_k)$ by (1.17). Then

$$(\Gamma' \cup \{v_k\varphi_k\}, \downarrow \equiv \overline{vv} \equiv \overline{vv}) \in R^{\leftarrow}_{\exists}(\{(\Gamma, \equiv \overline{vv})\}) \subseteq R^{\leftarrow}_{\exists}\left(\overline{D}^{(m+1)}(\emptyset)\right) \overset{1.5.0.3}{\subseteq} \overline{D}^{(1)}\left(\overline{D}^{(m+1)}(\emptyset)\right) = \overline{D}^{(m+2)}(\emptyset),$$

where first inclusion is given by monotonicity of $R^{\leftarrow}_{\exists}$, and we set $\Gamma' := \Gamma \setminus \left\{\frac{v'_k}{v_k}\varphi_k\right\} \subseteq X_k$. This contradicts (1.17). □

Notation 1.10.2.3. If S is a countable language, one can always find $l \in \left(\#_S^{-1}[\{0\}]\right)^{\mathbb{N}}$, $f \in (F_S)^{\mathbb{N}}$ such that

$$\mathbb{N} \ni n \mapsto l(n)\, f(n)$$

is onto $\mathrm{dom}\, W_S$. This surjectivity property aside, we will not be interested in how l and f actually work, and we will thus write $\mathcal{W}_D$ instead of $\mathcal{W}_D^{l,f}$ when dealing with a ruleset D of a countable language, implying l and f satisfy it.

Lemma 1.10.2.4. *Let D be a ruleset of a countable language S. Assume that the sets X, Y satisfy:*

1. $\mathrm{Con}_D(Y)$;
2. $\mathcal{W}_D(X) \subseteq Y$;
3. $\#_S^{-1}[\{0\}] \setminus \lfloor X \rfloor$ *is not finite.*

Then Y is S-witnessed.

Note that no particular request is placed on D.

Proof. $\mathcal{W}_D = \mathcal{W}_D^{l,f}$ for some pair of maps l, f. Thanks to hypothesis (3) we have, referring to 1.10.2.1:

$$a_m \neq \emptyset \qquad \forall m \in N. \tag{1.20}$$

Now assume $v\varphi \in Y$. By surjectivity, there is $n \in \mathbb{N}$ such that $v\varphi = l(n)\, f(n)$. Set

$$\begin{aligned} v_n &:= l(n) \\ \varphi_n &:= f(n) \\ v'_n &:= l\left(\min l^{-1}[a_n]\right). \end{aligned}$$

We have $X_n \subseteq \mathcal{W}_D(X) \subseteq Y$ and $\{v_n\varphi_n\} \subseteq Y$, whence $\mathrm{Con}_D(X_n \cup \{v_n\varphi_n\})$, which, together with (1.20), implies either

$$X_n \cup \left\{\frac{v'_n}{v_n}\varphi_n\right\} = X_{n+1} \subseteq \mathcal{W}_D(X) \subseteq Y$$

or

$$X_{n+1} = X_n \text{ and } X_n \cap W[\{v_n\varphi_n\}] \neq \emptyset$$

by definition 1.10.2.1. Given the arbitrariness of $v\varphi$, this yields thesis as demanded by 1.9.4.6. □

1.10.3 Consistent maximization for countable languages

Definition 1.10.3.1. Given a mapping $f : \mathbb{N} \ni n \mapsto \varphi_n \in F_S$, recursively define

$$X_0 := X$$
$$X_{n+1} := \begin{cases} X_n \cup \{\downarrow \varphi_n \varphi_n\} & \text{if } X_n \underset{D}{\vdash\!\!\!\!\!\!/}\downarrow \varphi_n \varphi_n \\ X_n \cup \{\varphi_n\} & \text{otherwise,} \end{cases}$$

and set

$$\mathcal{E}_D^f(X) := \bigcup_{n \in \mathbb{N}} X_n.$$

Lemma 1.10.3.2 (Lindenbaum's lemma)**.** *If D is a cut-like ruleset of S and $f \in (F_S)^{\mathbb{N}}$, then $\mathrm{Con}_D(X)$ implies $\mathrm{Con}_D\left(\mathcal{E}_D^f(X)\right)$ for any X.*

Proof. Assume $\mathrm{Inc}_D\left(\mathcal{E}_D^f(X)\right)$; then $\min\{n \in \mathbb{N} |\ \mathrm{Inc}_D(X_n)\} \in \mathbb{Z}^+$ (it cannot be zero because $X = X_0$ is consistent by hypothesis), so it equals $m+1$ for some $m \in \mathbb{N}$. Set $\varphi_m := f(m)$.

Now, it cannot be $X_{m+1} = X_m \cup \{\varphi_m\}$, for in this case we would get $X_m \nvdash_D \downarrow \varphi_m \varphi_m$ by definition 1.10.3.1, and, as a consequence, $\mathrm{Con}_D(X_m \cup \{\varphi_m\})$ by 1.10.1.10, while X_{m+1} is inconsistent. Hence the upper branch of definition 1.10.3.1 must be the one in charge, that is

$$X_{m+1} = X_m \cup \{\downarrow \varphi_m \varphi_m\}, \tag{1.21}$$

and consequently

$$X_m \vdash_D \downarrow \varphi_m \varphi_m. \tag{1.22}$$

On the other hand, from (1.21) and 1.10.1.10 it descends that

$$X_m \vdash_D \downarrow \downarrow \varphi_m \varphi_m \downarrow \varphi_m \varphi_m,$$

yielding, together with (1.22), that X_m is inconsistent according to definition 1.10.1.3, thus contradicting minimality of $m+1$.

□

Notation 1.10.3.3. If S is a countable language, one can always find $f \in (F_S)^{\mathbb{N}}$ being onto F_S. This surjectivity property aside, we will not be interested in how f actually works, and we will thus write $\mathcal{E}_D$ instead of $\mathcal{E}_D^f$ when dealing with a ruleset D of a countable language, implying f satisfies it.

Proposition 1.10.3.4. *Let D be a ruleset of a countable language S. $\mathcal{E}_D(X)$ is a cover of S.*

Proof. $\mathcal{E}_D = \mathcal{E}_D^f$ for some function f onto F_S. Consider $\varphi \in F_S$, and, by surjectivity, a natural number n such that $\varphi = \varphi_n := f(n)$.
Either $X_n \nvdash_D \downarrow \varphi_n \varphi_n$ or $X_n \vdash_D \downarrow \varphi_n \varphi_n$, where X_n is as from 1.10.3.1. Therefore, by 1.10.3.1, either $X_{n+1} = X_n \cup \{\varphi_n\}$ or $X_{n+1} = X_n \cup \{\downarrow \varphi_n \varphi_n\}$, and $X_{n+1} \subseteq \mathcal{E}_D(X)$. Thus at least one between

$$X_n \cup \{\varphi_n\}$$

and

$$X_n \cup \{\downarrow \varphi_n \varphi_n\}$$

is a subset of $\mathcal{E}_D(X)$, giving that at least one between $\downarrow \varphi_n \varphi_n$ and φ_n belongs to $\mathcal{E}_D(X)$. This, by the arbitrariness of φ and by definition 1.10.1.3, ends the proof. □

Remark 1.10.3.5. Together, 1.10.3.2 and 1.10.3.4 yield that a D-consistent set X can be completed to the D-consistent cover $\mathcal{E}_D(X)$. This, until one adds the request of D being weakly assumptive (see 1.10.1.5), does not generally imply that it can be completed to a maximally consistent set, which is the thesis of the standard formulation (see, e.g., [Smu95], section III.2 and [Che80], 2.19) of Lindenbaum's lemma.

1.11 Putting it all together

Lemma 1.11.0.6. *Let D be a ruleset of a countable language S, and X be a set; assume they comply with the following requirements:*

1. $R^{\leftarrow}_{\exists} \in D$;
2. *D is cut-like;*
3. $D \geq_{\emptyset} \{R_{=}\}$;
4. $\#_S^{-1}[\{0\}] \setminus \lfloor X \rfloor$ *is not finite;*
5. $\mathrm{Con}_D(X)$.

Then $\mathcal{E}_D(\mathcal{W}_D(X))$ is a witnessed, D-consistent S-cover.

Proof. Set

$$Y := \mathcal{W}_D(X) \qquad\qquad Z := \mathcal{E}_D(Y).$$

Z is a cover by 1.10.3.4. By (1), (2), (3), (5) and 1.10.2.2, Y is D-consistent. Consequently Z is D-consistent as well by 1.10.3.2, (2). This fact, fed together with (4) into 1.10.2.4, grants that Z is S-witnessed, ending the proof. □

Lemma 1.11.0.7. *Let D be a ruleset of the language S, and X be a set such that*

1. *S is countable;*
2. $\mathrm{Con}_D(X)$;
3. $\#^{-1}[\{0\}] \setminus \lfloor X \rfloor$ *is not finite;*
4. $R^{\leftarrow}_{\exists} \in D$;
5. *D is cut-like;*
6. $D \geq_{\emptyset} \left\{R_0, R_{=}, R_{\leftrightarrows}, R_{\rightrightarrows}, R_{+}, R_{\mathcal{R}}, R^{\rightarrow}_{\exists}, R_{\downarrow}\right\}$.

Then $\mathcal{H}_{D,\mathcal{E}_D(\mathcal{W}_D(X))} \models X$.

Proof. Set $Y := \mathcal{E}_D(\mathcal{W}_D(X)) \supseteq X$. By 1.7.0.10 and 1.7.0.9, $D \geq_\emptyset \{R_=\}$, so 1.11.0.6 can be invoked: Y is a witnessed, D-consistent S cover. Analogously, $D \geq_\emptyset \{R_0\}$, so that D is strongly assumptive by 1.10.1.8 and 1.10.1.9. By 1.10.1.5, then, Y is also a D-closed mincover. Hence, $\varphi \in Y \Leftrightarrow \overline{\mathcal{H}_{D,Y}}(\varphi) = 1$ by 1.9.4.8. In particular, $\mathcal{H}_{D,Y} \models X$. □

Proposition 1.11.0.8. *If $\{R_\cup, R_c\} \subseteq D$ and D is monotone, then D is cut-like.*

Proof. Consider a set X and a wff φ such that $\mathrm{Inc}_D(X \cup \{\varphi\})$. We have to show that $X \vdash_{\overline{D}} \neg\varphi$. By assumption, there are $\Gamma_1, \Gamma_2 \subseteq X \cup \{\varphi\}$ finite, and ψ such that $(\Gamma_1, \psi) \in \overline{D}^{(m)}(\emptyset)$ and $(\Gamma_2, \neg\psi) \in \overline{D}^{(n)}(\emptyset)$ for some $m, n \in \mathbb{N}$. Since D is monotone, by 1.7.0.7, we have $\{(\Gamma_1, \psi), (\Gamma_2, \neg\psi)\} \subseteq \overline{D}^{(m+n)}(\emptyset)$. So

$$\begin{aligned}(\Gamma, \neg\varphi) &\in R_c(\{(\Gamma \cup \{\varphi\}, \psi), (\Gamma \cup \{\varphi\}, \neg\psi)\}) \\ &\subseteq R_c(R_\cup(\{(\Gamma_1, \psi), (\Gamma_2, \neg\psi)\})) \subseteq R_c\left(R_\cup\left(\overline{D}^{(m+n)}(\emptyset)\right)\right) \\ &\subseteq R_c\left(\overline{D}\left(\overline{D}^{(m+n)}(\emptyset)\right)\right) \subseteq \overline{D}\left(\overline{D}\left(\overline{D}^{(m+n)}(\emptyset)\right)\right) = \overline{D}^{(2+m+n)}(\emptyset),\end{aligned}$$

where we set $\Gamma := \Gamma_1 \cup \Gamma_2 \setminus \{\varphi\}$. Hence $X \setminus \{\varphi\} \vdash_{\overline{D}} \neg\varphi$. □

Corollary 1.11.0.9. *Given a countable language S, and X such that $\#_S^{-1}[\{0\}] \setminus \lfloor X \rfloor$ is not finite, suppose X is D_0-consistent, where*

$$D_0 := \left\{R_0, R_=, R_{\overset{\leftrightarrow}{=}}, R_{\overset{\Rightarrow}{=}}, R_+, R_{\mathcal{R}}, R_\downarrow, R_{\overset{\rightarrow}{\exists}}, R_{\overset{\leftarrow}{\exists}}, R_c, R_\cup\right\}.$$

Then

$$\mathcal{H}_{D_0, \mathcal{E}_{D_0}\left(\mathcal{W}_{D_0}(X)\right)} \models X.$$

Proof. From the fact that D_0 is monotone we can draw two conclusions: $D_0 \geq_\emptyset \left\{R_0, R_=, R_{\overset{\leftrightarrow}{=}}, R_{\overset{\Rightarrow}{=}}, R_+, R_{\mathcal{R}}, R_\downarrow, R_{\overset{\rightarrow}{\exists}}\right\}$, by 1.7.0.10, and D_0 is cut-like by 1.11.0.8, so that 1.11.0.7 can be invoked. □

We now want to get rid of requirement (3) in the statement of 1.11.0.7. This will be accomplished by following the standard path of adjoining to (the symbol set of) the language S a countably infinite family N of fresh literals, enlarging it to a second language S_N; then 1.11.0.7 is applied to S_N, and carried on to S, being the latter a restriction of the former. To do this, we have to show the natural fact that satisfaction relation, 1.8.0.29, is preserved through such enlargements and restrictions:

Lemma 1.11.0.10 (Coincidence lemma)**.** *Let S_1, S_2 be languages. Let i_1 i_2 be interpretations, of S_1 and S_2 respectively, over the same universe U. Assume that*

1. $\equiv_{S_1} = \equiv_{S_2}$;
2. $\downarrow_{S_1} = \downarrow_{S_2}$;
3. $(\#_{S_1})|_{dom(\#_{S_1})} = (\#_{S_2})|_{dom(\#_{S_1})}$;

4. $i_1|_{dom(\#_{S_1})} = i_2|_{dom(\#_{S_1})}$.

Then $F_{S_1} \subseteq F_{S_2}$ *and* $\overline{i_1}|_{F_{S_1}} = \overline{i_2}|_{F_{S_2}}$.

Proof of 1.11.0.10 turns out to be tedious, giving rise to a 'de Bruijn surge': its proof in Mizar seem disproportionally verbose with respect to both its informal counterparts and the simplicity of the intuitive idea conveyed, so that its de Bruijn factor (see 3.4) sharply increases: that same proof takes less than one page in ([EFT84], III.5.1). Whether this fact depends inherently on the result or the chosen formalization system, or even on the coder not devising a better proof seems very hard to assess. The reader is thus referred to Mizar sources for that proof (`FOMODEL3.MIZ:12`).

Theorem 1.11.0.11 (Satisfiability theorem)**.** *Suppose that*

1. S *is a countable language;*
2. $X \subseteq F_S$*;*
3. $R^{\leftarrow}_{\exists} \in D$*;*
4. D *is cut-like;*
5. $D \geq_{\emptyset} \left\{R_0, R_=, R^{\leftrightarrow}_{=}, R^{\Rightarrow}_{=}, R_+, R_{\mathcal{R}}, R^{\rightarrow}_{\exists}, R_{\downarrow}\right\}$*;*
6. $\mathrm{Con}_D(X)$.

Then there is an interpretation of S *having a countable universe and satisfying* X.

Proof. Consider a countably infinite set N missing both S and $\lfloor X \rfloor$, and the language S_N extending S and obtained by setting

$$\begin{aligned} \equiv_{S_N} &:= \equiv_S \\ \downarrow_{S_N} &:= \downarrow_S \\ \#_{S_N} &:= N \times \{0\} \cup \#_S . \end{aligned}$$

By construction, S_N is countable (because S and N are) and $N \subseteq \#^{-1}_{S_N}(\{0\}) \setminus \lfloor X \rfloor$. Now set

$$D_N := \left\{R_{0,S_N}, R_{\cup,S_N}, R_{=,S_N}, R^{\leftrightarrow}_{=,S_N}, R^{\Rightarrow}_{=,S_N}, R_{+,S_N}, R_{\mathcal{R},S_N}, R_{\downarrow,S_N}, R^{\rightarrow}_{\exists,S_N}, R^{\leftarrow}_{\exists,S_N} R_{c,S_N}\right\}.$$

Suppose we manage to show

$$\mathrm{Con}_{D_N}(X). \tag{1.23}$$

Then we can deploy 1.11.0.9, and infer that

$$H_{D_N,\mathcal{E}_{D_N}}\left(\mathcal{W}_{D_N}(X)\right) \vDash X. \tag{1.24}$$

The very final step towards thesis is to realize that $H_{D_N,\mathcal{E}_{D_N}\left(\mathcal{W}_{D_N}(X)\right)}$ can be restricted to an interpretation i of S, and that this latter interpretation returns the same truth value as $H_{D_N,\mathcal{E}_{D_N}\left(\mathcal{W}_{D_N}(X)\right)}$ on every formula of X thanks to 1.11.0.10, so that $i \models\!\!\!\models X$ by (1.24).

Subproof for claim (1.23) It will suffice to show (1.23) holds for a generic *finite* $Y \subseteq X$:

$$\mathrm{Con}_{D_N}(Y). \tag{1.25}$$

Thus, let $Y \subseteq X$, Y being finite. Now, $\mathrm{Con}_D(Y)$ (use hypothesis (6)) and $\#_S^{-1}(\{0\}) \setminus \lfloor Y \rfloor$ is not finite, so $H_{D,\mathcal{E}_D\mathcal{W}_D Y} \vDash Y$ by 1.11.0.7 and hypotheses (3), (4) and (5). Consider an interpretation $i_{N,Y}$ of S_N obtained by extending $H_{D,\mathcal{E}_D\mathcal{W}_D Y}$ to S_N arbitrarily: we can do so keeping the universe of $i_{N,Y}$ the same as that of $H_{D,\mathcal{E}_D\mathcal{W}_D Y}$, so that, given $\varphi \in Y$, one has (again by 1.11.0.10) $\overline{i_{N,Y}}(\varphi) = \overline{H_{D,\mathcal{E}_D\mathcal{W}_D Y}}(\varphi)$; hence $i_{N,Y} \models\!\!\!\models_{S_N} Y$. This in the end implies $\mathrm{Con}_{D_N}(Y)$, as D_N is sound.

□

Corollary 1.11.0.12. *Let D be a ruleset of a countable language S, and $X \subseteq F_S$. Suppose*

1. $\left\{R_{\underset{\exists}{\leftarrow}}, R_c, R_\cup\right\} \subseteq D$;
2. *D is monotone;*
3. $D \geq_\emptyset \left\{R_0, R_=, R_{\underset{=}{\leftrightarrow}}, R_{\underset{=}{\Rightarrow}}, R_+, R_\mathcal{R}, R_{\underset{\exists}{\rightarrow}}, R_\downarrow\right\}$;
4. $\mathrm{Con}_D(X)$.

Then there is an interpretation of S having a countable universe and satisfying X.

Corollary 1.11.0.13 (Countable downward Löwenheim-Skolem theorem)**.** *Assume $X \subseteq F_S$ is countable, and suppose there is an interpretation i of S such that $i \models\!\!\!\models X$. Then there is an interpretation i' of S having a countable universe and satisfying X as well.*

Proof. Let N be a countably infinite subset of the symbol set of S such that $\lfloor X \rfloor \cup \{\equiv_S, \downarrow_S\} \subseteq N$. Restrict $\#_S$ and i to N, obtaining respectively a countable language S' and an interpretation i' of the latter over the same universe of i.

For any $\varphi \in X$, one has that φ is also a formula of S', and that $\overline{i}(\varphi) = \overline{i'}(\varphi)$ by construction and coincidence lemma, 1.11.0.10, so that $i' \models\!\!\!\models_{S'} X$, and hence $\mathrm{Con}_D(X)$, where we set

$$D := \left\{R_0, R_=, R_{\underset{=}{\leftrightarrow}}, R_{\underset{=}{\Rightarrow}}, R_+, R_\mathcal{R}, R_{\underset{\exists}{\rightarrow}}, R_\downarrow, R_{\underset{\exists}{\leftarrow}}, R_c, R_\cup\right\},$$

thanks to soundness. This allows to consider an interpretation j' of S' having a countable universe and satisfying X by 1.11.0.12. This latter interpretation can be arbitrarily enlarged to one of S with the same universe, preserving the satisfiability of X through it (again thanks to coincidence lemma), and thus yielding thesis. □

Remark 1.11.0.14. We note that the language S in 1.11.0.13 is not required to be countable.

Definition 1.11.0.15 (Entailment)**.** Given sets X, Y, we say that X *entails* Y with respect to the language S if any interpretation i of S satisfying X also satisfies Y.

In this case we write $X \models_S Y$ or just $X \models Y$. We also will usually write $X \models_S \varphi$ (or $X \models \varphi$) in lieu of $X \models_S \{\varphi\}$.

Remark 1.11.0.16. The symbol $\models$ results thus overloaded by definitions of satisfaction (1.8.0.29) and entailment (1.11.0.15). The type of the argument on its left will usually resolve which use is being made.

Corollary 1.11.0.17 (of 1.11.0.11)**.** *Let X be a subset of the set of formulas F_S of a countable language S, and D be a cut-like ruleset of S such that*

1. $R^{\leftarrow}_{\exists} \in D$
2. $D \geq_\emptyset \left\{R_0, R_=, R_{\leftrightarrows}, R_{\rightrightarrows}, R_+, R_{\mathcal{R}}, R^{\rightarrow}_{\exists}, R_\downarrow\right\}$.

Then $X \models\downarrow \varphi\varphi$ implies $X \vdash_D \downarrow \varphi\varphi$ for any $\varphi \in F_S$.

Proof. By contradiction. Suppose that $X \vdash_D \downarrow \varphi\varphi$ is false. Then, D being cut-like, $\mathrm{Con}_D(X \cup \{\varphi\})$. Hence, allowed by 1.11.0.11, let us consider an interpretation i of S such that

$$\bar{i}(\varphi) = 1 \tag{1.26}$$

$$i \models X. \tag{1.27}$$

Given the hypothesis, $X \models\downarrow \varphi\varphi$, so that by definition of entailment and (1.27), $\bar{i}(\downarrow \varphi\varphi) = 1$. Now, by 1.8.0.27, $\bar{i}(\varphi) = 0$, contradicting (1.26). □

Theorem 1.11.0.18 (Gödel's completeness theorem)**.**

$$X \models \varphi \qquad \textit{implies} \qquad X \vdash_{D_1} \varphi,$$

where we set

$$D_1 := \left\{R_0, R_=, R_{\leftrightarrows}, R_{\rightrightarrows}, R_+, R_{\mathcal{R}}, R^{\rightarrow}_{\exists}, R_\downarrow, R^{\leftarrow}_{\exists}, R_c, R_\cup, R_{\not\prec}\right\}.$$

Proof. Assume $X \models \varphi$. Then $X \models\downarrow\downarrow \varphi\varphi\downarrow \varphi\varphi$ by 1.8.0.27. This implies the existence of $\Gamma \subseteq X$ such that $(\Gamma, \downarrow\downarrow \varphi\varphi\downarrow \varphi\varphi) \in \overline{D_1 \setminus \{R_{\not\prec}\}}^{(\infty)}(\emptyset)$ by 1.11.0.17. Hence there is $k \in \mathbb{Z}^+$ such that $(\Gamma, \downarrow\downarrow \varphi\varphi\downarrow \varphi\varphi) \in \overline{D_1}^{(k)}(\emptyset)$, so that

$$\begin{aligned}(\Gamma, \varphi) \in R_{\not\prec}(\{(\Gamma, \downarrow\downarrow \varphi\varphi\downarrow \varphi\varphi)\}) &\subseteq R_{\not\prec}\left(\overline{D_1}^{(k)}(\emptyset)\right) \\ &\subseteq \overline{D_1}\left(\overline{D_1}^{(k)}(\emptyset)\right) = \overline{D_1}^{(k+1)}(\emptyset).\end{aligned}$$

□

1.12 Alternative rules

The attributes 'weakly assumptive', 'strongly assumptive', and 'cut-like' have been introduced to detach, to some extent, the main results proven from the particular choice of derivation rules. Indeed the rulesets occurring in hypotheses of main theorems we saw are often required to be applicable such attributes, rather than to include some specific rules. This means that if one of those results is valid for a given ruleset, it remains valid if we substitute in that ruleset some rules satisfying a given attribute with others, as long as the new rules still make the ruleset satisfy the corresponding attribute. As an example, consider the following pair of new rules.

Definition 1.12.0.19. Given a literal $\overline{v}$, define

$$R_{<\overline{v}} : G(S) \supseteq \Sigma \mapsto \{(\Gamma, \varphi) : \exists \Gamma_1, \Gamma_2, \psi_0, \psi | \quad (\Gamma_1, \psi_0), (\Gamma_2, \downarrow \psi_0 \psi_0) \in \Sigma \text{ and } \varphi = \downarrow \equiv \overline{v}\overline{v}\neg\psi_0 \text{ and } \Gamma = \Gamma_1 \cup \Gamma_2 \cup \{\psi\}\} \subseteq G(S)$$
$$R_{\overline{v}} : G(S) \supseteq \Sigma \mapsto$$
$$\{(\Gamma, \varphi) : \exists \psi, \psi_0 | (\Gamma \cup \{\psi\}, \downarrow \equiv \overline{v}\overline{v}\neg\psi_0) \in \Sigma \text{ and } \varphi = \neg\psi \text{ and } \Gamma \setminus \{\psi\} = \Gamma\}$$

Notation 1.12.0.20. We also give the diagram representation (introduced in section 1.6) for rules defined in 1.12.0.19:

$$R_{<\overline{v}} : \frac{\Gamma_1 \vdash \psi \qquad\qquad \Gamma_2 \vdash \neg\psi}{\Gamma_1 \quad \Gamma_2 \quad \varphi \vdash \downarrow \equiv \overline{v}\overline{v}\neg\psi}$$
$$R_{\overline{v}} : \frac{\Gamma \quad \varphi \vdash \downarrow \equiv \overline{v}\overline{v}\neg\psi}{\Gamma \qquad \vdash \qquad \neg\varphi}$$

The following result, mirroring 1.11.0.8, permits to replace $\{R_\cup, R_c\}$ with $\{R_{<\overline{v}}, R_{\overline{v}}\}$ in the statement of 1.11.0.18.

Proposition 1.12.0.21. *A monotone ruleset $D \geq \{R_{<\overline{v}}, R_{\overline{v}}\}$ is cut-like.*

Proof. Assumed $\mathrm{Inc}_D(X \cup \{\varphi\})$, we must show $X \overset{\vdash}{\not}_D \neg\varphi$. There are $\Gamma_1, \Gamma_2 \subseteq X \cup \{\varphi\}$, $m, n \in \mathbb{Z}^+$, ψ such that $(\Gamma_1, \psi) \in \overline{D}^{(m)}(\emptyset)$, $(\Gamma_2, \neg\psi) \in \overline{D}^{(n)}(\emptyset)$. By the fact that D is monotone, we have $(\Gamma_1, \psi), (\Gamma_2, \neg\psi) \in \overline{D}^{(p)}(\emptyset)$, where $p := \max\{m, n\}$. Now

$$(\Gamma_1 \cup \Gamma_2 \cup \{\varphi\}, \downarrow \equiv \overline{v}\overline{v}\neg\psi) \in R_{<\overline{v}}(\{(\Gamma_1, \psi), (\Gamma_2, \neg\psi)\}) \subseteq R_{<\overline{v}}\left(\overline{D}^{(p)}\right)$$
$$\subseteq \overline{D}^{(\infty)}\left(\overline{D}^{(p)}(\emptyset)\right) \Longrightarrow \exists q \in \mathbb{N} | \quad (\Gamma_1 \cup \Gamma_2 \cup \{\varphi\}, \downarrow \equiv \overline{v}\overline{v}\neg\psi) \in \overline{D}^{(q)}\left(\overline{D}^{(p)}(\emptyset)\right),$$

so that

$$(\Gamma_1 \cup \Gamma_2 \cup \{\varphi\} \setminus \{\varphi\}, \neg\varphi) \in R_{\overline{v}}(\{(\Gamma_1 \cup \Gamma_2 \cup \{\varphi\}, \downarrow \equiv \overline{v}\overline{v}\neg\psi)\})$$
$$\subseteq R_{\overline{v}}\left(\overline{D}^{(p+q)}(\emptyset)\right) \subseteq \overline{D}^{(\infty)}\left(\overline{D}^{(p+q)}(\emptyset)\right)$$
$$\Longrightarrow \exists l \in \mathbb{N} | \quad (\Gamma_1 \cup \Gamma_2 \cup \{\varphi\} \setminus \{\varphi\}, \neg\varphi) \in \overline{D}^{(l)}\left(\overline{D}^{(p+q)}(\emptyset)\right).$$

□

Chapter 2

The formalization

This chapter illustrates the actual Mizar implementation of the set-theoretical treatment of first-order languages built in chapter 1; it includes material from [Cam10] and [CR11]. Introductory sections 2.1 and 2.2 give background on proof checkers and on the particular proof checker chosen in our case, respectively.

2.1 Software for proving

Rigor and creativity are both essential qualities of mathematics. Logic supplies precise notions of rigor, and tools to attain it: for example, Zermelo-Fraenkel set theory with the axiom of choice (ZFC) is commonly accepted as a first-order axiom system in which most parts of current mathematics could be rendered; however, such renditions (commonly referred to as formalizations) are usually reputed to be tedious if not impracticable, and anyway a hindrance for the creative process, equally essential for mathematics. Thus, instead of actually formalize mathematics, the classical compromise is to supply a sketch of formalization in a variably rigorous pseudo-code, the purpose of which is to get accepted (and thus possibly trusted, relied on and employed, in the end) as a result of what is ultimately a social process: the one of persuading other people of its correctness ([AGN09]).

The success of Hilbert's program in thrusting towards formalization of mathematics and the advent of digital computers set the scene for a change. Virtually every scientific realm presents examples of endeavors which were unthinkable before the advent of computers: given the evident affinity between formalization and mechanization, one can arguably maintain that formalization of mathematics might well become such an endeavor ([Boy+94], [Wie07b]).[1] And indeed, since de Bruijn's Automath ([Bru70]), the software implementations of proof checkers proliferated.[2]

In the vast landscape of software born to carry out the old idea of mechanizing proofs, a first distinction can be drawn between *proof checkers* (like Mizar, Metamath, Twelf, Automath) and *automated theorem provers* (like E, ACL2, SPASS, Vampire). The latter *find* proofs, rather than merely certifying them. One of the first known

[1] Recent years have provided a further strong reason, probably not foreseeable at the time in which [Boy+94] was written, to be optimistic about the feasibility of this endeavor: the several blatant and huge successes brought by the commons-based peer production model ([Ben06]), like, most notably, the GNU/Linux operating system and the Wikipedia project.

[2] http://www.cs.ru.nl/~freek/digimath/

concrete computer programs developed for proving, namely *Logic Theorist*, [NS56], was a representative of this category.

Proof assistants, or *interactive theorem provers* (Coq, Isar, Matita, PhoX, to name a few), stand between the two ends, requiring some user intervention, the amount and form of which varies greatly among different systems, to guide the proof, yet saving him to spell out a full proof.

There is a further family of recent projects ([Cra+10][3], [HR10], [Sch+12]) taking an alternative, 'linguistic' approach: the very rough idea is to supply a 'controlled natural language' coupled with some automated prover which validate the formal language extracted from the higher-level natural language. This would relief mathematicians from both the burdens of proving the trivial details and of facing a language less friendly than the common mathematical language, with the controlled natural language acting as an interface with both the automated prover and the formal language backends. Less ambitiously, ProofCheck (see [NA09]) embeds a low-level proof checker directly into the TEX and LATEX languages via additional TEX macros.

The largest digital libraries of already formalized mathematics are those written with the proof checkers Mizar, HOL Light, Coq and Isabelle. Mizar is the most mathematically-oriented one, adopting a grammar resembling common mathematical language, a declarative style, and being based on set theory.

2.2 An overview of Mizar

The Mizar project (http://www.mizar.org) delivers a few provisions:

1. Mizar *language* permits to write formulas in first-order set theory which read close to common mathematical language. For example, the formula

$$X \neq \emptyset \Longrightarrow \exists x (x \in X)$$

is written

```
X <> {} implies ex x st x in X;
```

In addition to the few reserved words pertaining to the first-order alphabet of set theory, the language specifies grammar and reserved words to invoke the verifier (see point 2) and to exploit advanced features of the system.

2. Mizar *verifier* (PC Mizar) is a software certifying whether one such formula can be deduced (according to some formal system for classical logic, see sections 2.2.1 and 3.5 of [GKN10]) from other given formulas, specified via the keyword `by` of the Mizar language:

```
A1: x in X;
A2: for y being set holds y in X\/Y iff (y in X or y in Y);
x in X\/Y by A1, A2;
```

[3] At the time of writing, Naproche seems the only one in this family having made tangible progress, to the point of offering a web interface: http://naproche.net/inc/webinterface.php, with LATEX support.

3. The Mizar Mathematical Library (MML) builds on the components (1) and (2) above to provide a mass of Mizar language formulas certified, by Mizar verifier, to be derivable from a handful of set-theoretical axioms affine to ZFC axioms. The set theory resulting from these axioms, Tarski-Grothendieck (TG), is an extension of ZFC, and more on it can be found in [RT99].

MML is made up of Mizar source files called *articles*, and its latest version is always browsable at `http://mizar.uwb.edu.pl/version/current/mml/`. In the following, we will be using typewriter font for referencing articles and results inside MML: for example, `XBOOLE_1:4` denotes the fourth theorem appearing in the MML article xboole_1.miz, which is thus viewable at `http://mizar.uwb.edu.pl/version/current/mml/xboole_1.miz`. We will also adopt typewriter font for Mizar code, as already done in point (1) of the numbered list above.

2.2.1 Types and definitions

The primitive workflow consisting of writing set-theoretical formulas, linking them together via the `by` keyword, and invoking the verifier on them, as depicted in section 2.2, would theoretically suffice to accomplish a great deal of first-order formalization tasks. In practice, one cannot actually get very far without higher-level abstractions to structure the code. Among others, Mizar supplies (soft) *types* and *definitions*:

Types A term can be assigned a type (via the reserved word `let`); as a consequence, the type of a term can be the subject of a first-order atomic formula. The special first-order relation symbol `is` has exactly this use:

```
let x be Function;
x is Function;
```

The formulas based on the special relation symbol `is`, as the one above, present the distinctive property of needing no justification: they are a way to query Mizar type system. This means that the last line of code in the example above is accepted by the verifier without the need of a `by` statement (see item (2) on page 40). The basic type `set` is applicable to any term.

Functors New function symbols (called *functors* in Mizar jargon) can be added to the first order language via the reserved word `func`. This can be done in two ways:

1. either in a macro-like fashion:

   ```
   definition
     let x, y be set;
     func [x,y] equals { { x,y }, { x } };
     ...
   ```

   In this case, the keyword `equals` is used.

2. or by stating some formula the new object must satisfy, subject to the proof that exactly one term exists for which this happens:

```
definition
  let X, Y be set;
  func X /\ Y -> set means
   for x being set holds x in it iff x in X & x in Y;
  existence
  proof
    ...
  end;
  uniqueness
  proof
    ...
  end;
end;
```

In this case, the keyword `means` is used, and the entity to be defined is denoted by the keyword `it` in the definiens, as seen above.

The two functionalities just introduced can work together, meaning that the definition of a functor can accept as arguments a finite list of *typed* arguments; and, viceversa, the term obtained by the application of the defined functor can be associated a type (keyword `->`):

```
definition
  let R be Relation;
  func R~ -> Relation means
    [x,y] in it iff [y,x] in R;
  ...
```

After this association, the verifier will know the type returned by any application of that functor. This suggests that the very presence of types can be a first, seminal step to some form of automation: some methods we shall see in section 3.1 rely on the capability of the system to know the type of each term straightaway, and all of them somehow revolve around the type system.

Sometimes, the abstraction of types hides the fact that two functors behave the same way at the underlying set-theoretical level, even if they operate on, or yield, different types; in this case one can make the verifier aware that the results coincide, using the keyword `identify`. For example:

```
registration
  let x,y be real number, a,b be complex number;
  identify x+y with a+b when x = a, y = b;
  compatibility
  proof
    ...
  end;
end;
```

This correspondence can be achieved because the type system implemented in the verifier is a soft one ([Wie07a]): terms are actually untyped sets, and one can always forget about their type, which is offered for a matter of convenience.

2.2.2 Attributes and registrations

Functorial registrations are a further form of Mizar automation, and one of the most powerful and least restricted. To see how it works, we need to introduce attributes.

Attributes

Attributes are a flexible and natural way to define types; they are used to qualify and restrict a given type (called *radix* type) by just prefixing it with the attribute name (or with its name preceded by the keyword `non`, to negate it). For example, article `XBOOLE_0` defines the attribute `empty`, applicable to any term, so that one can write:

```
{} is empty set;
```

It is important to note that this juxtaposition is a subtype of the radix, and therefore can be treated like it under many aspects; at the same time, being itself a type, attributes can in turn be applied to it. To put it differently, attributes can be clustered:

```
{} is empty finite set;
```

This flexibility is a first reason to prefer them to the standard way of defining types seen in section 2.2.1.

Functorial registrations

Functorial registrations automatically attach an attribute to all terms presenting a given syntactic form or pattern, once one proves (keyword `coherence` in the snippet below) that terms of that form can be assigned the given attribute. For example:[4].

```
registration
  let X be set;
  cluster (bool X) \ X -> non empty for set;
  coherence
  proof
    ...
  end;
end;
```

Note that the term in the example above contains two nested functors; there are no limitations on the syntactical complexity of a term being applied a functorial registration. This kind of registration will have a fundamental role in doing sequent calculus in Mizar (section 2.6.8) and in implementing custom Mizar automations (section 3.1).

[4] `bool X` is the power set of `X`. See appendix B

Attribute registrations

Attribute registrations works in a way similar to functorial registrations: the attribute on the right of the keyword `->` gets automatically attached to a term (which must have the type appearing on the right of the keyword `for`) based on the condition expressed by the matter on the left of that special symbol. What is different is how this condition works: instead of checking that a term has a given shape to apply the automation, now it is applied when a term of a given type possesses a given attribute. So this registration has the form

`cluster` *attribute*1 `->` *attribute*2 `for` *type*. (2.1)

Once such a registration is enforced, for any term of type *type* one has that if the checker knows this term enjoys *attribute*1, the checker also knows this term enjoys *attribute*2. Note that, contrary to functorial registrations, the left hand side of `->` can be empty, which means that the checker will attach an attribute to any term of a given type, regardless of the term being applicable a further attribute. Of course, upon registering, one has to prove the corresponding first order formula

`for X being` *type* `st X is` *attribute*1 `holds X is` *attribute*2.

Such proofs has to be enclosed in a `coherence` block immediately following the registration statement. For example

```
registration
  cluster empty -> one-to-one for Function-like (Relation-like set);
  coherence
  proof
    ...
  end;
end;
```

2.2.3 Predicates

In many formulations of first-order languages, as in the one seen in chapter 1, one has operation symbols (also said function symbols) each operating on terms and yielding a term; correspondingly there are predicate symbols (also said relation symbols) each operating on terms and yielding truth values. In the same manner, besides functors, which yield terms, Mizar offers predicates, which yield truth values. Alongside of the basic predicates `in` (the primitive binary relation of ZFC and TG set theories) and `is` (introduced in section 2.2.1), one of the most pervasive relations in set theory is that of inclusion, which we take as an instance to show how Mizar predicates work:

```
definition
 let X,Y be set;
 pred X c= Y means
  for x being set st x in X holds x in Y;
end;
```

Note that Mizar does not provide for predicates forms of automations as powerful as those seen in section 2.2.2 for attributes. For example, the following can be automated

```
let X, Y be set; X /\ Y \ X is empty;
```

while the predicate-based equivalent formula

```
let X, Y be set; X /\ Y c= X;
```

cannot. We will detail on such topics in chapter 3.

2.3 First-order logic in MML

Inside Mizar Mathematical Library there are at least three strains hosting articles of content suitable for the treatment of first-order logic:

1. A series of articles supplying a language apt to describe set theory according to Zermelo-Fraenkel axioms, started with [Ban90].
2. A series of articles supplying a general language for first-order logic, started with [RT90].
3. A series of articles supplying terminology and results about universal algebras, started with [KMK92].

Most of the classical results of first order logic have, during the years, found their way in strain (2): building on those articles a fairly equipped gear of formalizations has been created.
There are treatments about the most elementary syntactical properties (those of variables and free variables in a formula (`QC_LANG3`), of subformulas (`QC_LANG2`, `QC_LANG4`), of substitution (`CQC_LANG`,`SUBSTUT1`,`SUBSTUT2`), of similarity between formulas (`CQC_SIM1`)), which in turn allow for less and less elementary results, regarding: propositional calculus (`PROCAL_1`, `LUKASI_1`), interpretation and satisfiability (`VALUAT_1`), Gentzen-style sequent calculus (`CALCUL_1`, `CALCUL_2`), up to a basic version of Gödel's completeness theorem (`HENMODEL`,`GOEDELCP`).

Unfortunately, the coding of the first order language adopted from the very beginning in [RT90] is somewhat rigid: roughly sketching the situation, strings of first-order language are represented as tuples of couples of natural numbers, with special symbols (quantifiers, connectives, truth symbol) represented by couples in which the first component is a reserved (small) natural.

This inherently prevents treating uncountable languages, which, alas, would be quite the point for developing even the most fundamental results of model theory, starting with Löwenheim-Skolem and compactness theorems.

Also, the completeness theorem currently present in MML has some limitations that look hardly removable in the established framework. For example, it is restricted to equality-lacking languages, while it would be of interest to talk about languages with equality: Mizar first-order language itself is furnished with equality, and the option of possibly applying results worked out to Mizar itself is desirable.

The following is an account of how a fully developed codebase for model theory in Mizar has been laid down, given the considerations above. They imposed reformulating things from scratch with a hopefully more flexible approach.

This codebase culminates, as a testbed for itself, with formalizations of the fundamental Gödel's completeness and Löwenheim-Skolem theorems, restricted to the case of a generic countable language, and has been submitted to MML Library Committee for peer-reviewing; after triple refereeing, it got accepted in MML in January 2011, with the corresponding five articles ([Cam11d], [Cam11a], [Cam11b], [Cam11c], [Cam11e]) published on 'Formalized Mathematics' in 2011. A 'dynamic' (i.e. constantly updated) version of it is accessible at the author's homepage[5] . More precisely, among the many flavors of Löwenheim-Skolem theorem, the one checked is the 'downward' flavor, like the one stated in 1.11.0.13. Its Mizar statement sounds like:

```
for
  U2 being non empty set, S being Language,
  X being countable Subset of AllFormulasOf S,
  I2 being Element of U2-InterpretersOf S st X is I2-satisfied
ex U1 being countable non empty set,
   I1 being Element of U1-InterpretersOf S st
X is I1-satisfied;
```

Let us report the Mizar statement of satisfiability theorem (compare 1.11.0.12), too:

```
for C being countable Language st
  X is (C-rules)-consistent & X c= AllFormulasOf C
    ex U being non empty countable set,
    I being Element of U-InterpretersOf C st
      X is I-satisfied;
```

Finally, the completeness theorem (see 1.11.0.18) runs thus:

```
for C being countable Language,
phi wff string of C, X being set st
  X c= AllFormulasOf C & phi is X-implied
holds
  phi is X-provable;
```

Note that this last restriction to countable languages is a mere matter of convenience: the whole work was set up to treat an arbitrary language up to Henkin's theorem (see 1.9.4.8); on the other hand, reducing to the least-cardinality case was desirable in order to have the job done more quickly (under the urge of demonstrating its usability), without having to handle complications related to the axiom of choice and the likes.

Those theorems are here regarded as significant goals because of their fundamental role in mathematical logic. In particular, the family of Löwenheim-Skolem theorems have a fruitful interplay with the cardinality of the language, which the ability to deal with, as said, was a starting, motivating point for the present work. Moreover,

[5] http://www.mat.uniroma1.it/people/caminati

this latter kind of results seem to be underrepresented in the global repository of mechanically checked mathematics: the only work sharing the aims of the present which the author is aware of is [Har98]; both the checker and the proof techniques used there are entirely different than what we are going to deploy here, however. Additionally, that work is subject to the issue, hinted in the introduction, of being stated in a language far from the standard mathematical one. Finally, this is the only known presentation of several fundamental theorems for model theory and proof theory formalized together and in a coherent, unitary framework.

2.4 Organization of the codebase

With a total of about $700k$ bytes and $19k$ lines of Mizar code, this turned out to be a fairly complex project, so care has been constantly taken to orderly arrange the various results according to their scope into five separate Mizar articles, each depending on the previous ones and hosting affine themes:

- `FOMODEL0.MIZ` is the receptacle of all results of broader scope stemmed during the various formalizations, with results and registrations about objects already in MML and quite few dependencies.
- `FOMODEL1.MIZ` introduces the type `Language`, the classification of symbols according to their arity and of terms according to their depth, and the functor to extract subterms from a term or an atomic formula. The bulk of syntax (section 1.2) is done here and in next article.
- `FOMODEL2.MIZ` (corresponding roughly to sections 1.2 and 1.8) deals with syntax of non atomic formulas and all the semantics by giving the following constructions: the definition of an interpretation I relative to a non empty set U (universe), the constructions saying how to evaluate a term in U, how to evaluate an atomic formula in $\{0, 1\}$, what can be regarded as a generic wff formula, how to evaluate it in $\{0, 1\}$ according to I, and how to evaluate its depth. Also, the functor to obtain another interpretation in the same universe U from I by changing the evaluation of a single literal symbol of the language (reassignment), and the definitions of satisfaction and of entailment are given.
- `FOMODEL3.MIZ` (mainly mirroring sections 1.8 and 1.9.1) supplies a toolkit of constructions to work with languages and interpretations, and results relating them: the free interpretation of a language, having as a universe the set of terms of the language itself, is defined; the quotient of an interpretation with respect to an equivalence relation is built, and shown to remain an interpretation when the relation respects it. Both the concepts of quotient and of respecting relation are defined in broadest terms, with respect to objects as general as possible. This is arguably the most 'technical' article in the tier.
- `FOMODEL4.MIZ` (reflecting material from sections 1.9.2, 1.9.3, 1.9.4, 1.10 and 1.11) introduces the proof-theoretical notions and binds all together. As a first more general task, it defines what a sequent and a rule are, and what means for a rule to be correct. Then, using these definitions, it builds the particular set of derivation rules we chose in 1.4.1.1. Among many other results, satisfiability

theorem is proven. Finally, restricting to countable languages, completeness and downward Löwenheim-Skolem are proved.

Having sketched the themes dealt with in each article, now the idea is that each formalized result should be placed in the lowest article in which the entities to enunciate it are available, so to give a precise criterion for the arraying of Mizar code among the five articles.

About one sixth of the code dwells in `FOMODEL0.MIZ`, thus applying to already-defined Mizar entities; also, the results located there tend to be shorter and more numerous than the lemmas showing up in subsequent articles. This is a clue of a general separation and modularization design policy pursued across the whole work, aiming at

- stating results in terms of the most general possible Mizar entities;
- breaking statements into smaller lemmas, especially if the latter as a result get applicable to a broader class of objects or if the smaller lemmas can be put together in more than a way to get significant theorems. The same applies to definitions.

As an example, take the construction of the already discussed Henkin model. In [EFT84], it is introduced just before the proof of the satisfiability theorem, and so, given the rather instrumental nature of its role, its definition is quite condensed. Here, on the other hand, it has been split into the pair of definitions of free interpretation, 1.8.0.24, and of quotient interpretation, 1.9.1.9, with a twofold benefit. First, the former object gets reused to define the term substitution in 1.8.0.32, and hence one of the deduction rules in 1.8.0.34. On the other hand, the latter applies not only to the former, but to any interpretation. What's more, the quotient functor is defined more generally as quotient of a relation by a pair of equivalence relations. Relations are more general than equivalence relations, which are in turn more general than functions, which finally are more general than interpretations, if one call an entity more general than another when the latter is defined in terms of the former.

Accordingly, the various results needed for the Henkin interpretation break into smaller and more general statements, sometimes of interest themselves, or occurring more than once in building further theorems, or maybe just hopefully useful to a possible coder in the future: having stated them in less restrictive terms increases the probability that this will be the case.

This process of separation and modularization may provide a further benefit: in breaking a statement into smaller steps, a fine-grained analysis of which assumptions are needed for each step is encouraged. This blatantly occurs in chopping down satisfiability theorem: in section 1.9 each step specifies which derivation rules are needed for it to hold (see also section 2.6.5). Indeed, keeping track of which result traces back to which rules did provide the main guidance in forming our ruleset. In the sequel, other, more specific occurrences of this attitude will be given: see especially section 3.2.

Here, another facet of this policy is examined: closely related to the just discussed tendency to predicate about as less specialized entities as possible is the choice of encoding formulas in simple strings of symbols.[6] As for a generic language, this

[6] In the context of Mizar formalizations, we will use the synonyms 'string' and 'finite sequence' (`FinSequence`) for the notion of 'tuple' defined in 1.1.0.9.

concrete syntax can be opposed by some representation-agnostic device describing the abstract syntax, in the same spirit of de Bruijn indexes ([Bru72]) or parse trees ([CH07], pages 34-36) approaches, which directly model the semantics and thus inherently dispense one from undergoing the twofold labor of first specifying the syntax rules for well-formedness and then give a way to attach a meaning to each formula. This is surely a strong plus for them.

We maintain that using 'plain text', as done here, presents advantages, too. A first advantage is readability: as strings require little assumed knowledge to be understood and have simple notations, the results worked out here are themselves very readable. Indeed plain text, concrete syntax is arguably one of the best representations of any data to be read by a human, in most diverse contexts ranging from didactic expositions of formal languages to software design (classical Unix philosophy advocates it as an universal interface, [Sal94], p.52). This is of importance especially for a project like Mizar which, besides verifying, also aims at building a library of mathematical knowledge straightforwardly accessible to humans.

Secondly, in the same vein of what has just been discussed, all the results worked out here are likely to produce sub-lemmas of interest to more Mizar coders than if we assume we chose parse trees: indeed, there is a series of Mizar articles supplying the machinery of parse trees in the context of formal languages (`DTCONSTR.MIZ`), and in this assumption, many of the general results in `FOMODEL0.MIZ` would have been in a form available only to the users of that machinery. This is a two-way phenomenon, of course: the author, using plain sequences instead of parse trees, has been able to take advantage of the massive amount of pre-existing results about the mode `FinSequence`. As an example of a 'by-product' of the present formalization which could be of more general interest, and which has been brought out because of the choice of using strings instead of more abstract representations, we pick a result regarding monoids and prefixes (see (2.2) in section 2.5); it is one of the numerous results got by treating sub-terms.

As a last argument supporting our choice, we remark a fundamental quality of our treatment of first order languages notably alleviating one arguably major drawback typically encountered when using 'plain text'; that is, the study of *free* occurrences of variables in strings, faced generally when studying the semantics of a previously defined syntax. In the present framework, one does not even need to *introduce* the concept of free occurrence, because our sequent calculus only demand to watch for simple occurrences of literals inside formulas (rule $R^{\exists}_{\leftarrow}$). The issues of free occurrences and of substitution are two related hindrances when describing or teaching (see [Tar65]) a formal language. They are related because when doing, or formalizing, substitution, attention is to be paid to prevent the capture of free variables: see [EFT84], III.8 for a standard exposition and for the typical complications arising.

In our case, we managed to devise a sequent calculus not needing this concept, and, on the other hand, substitution is resolved using a novel formalization approach, to the best of author's knowledge, that is, reusing the functors `-freeInterpreter`, `-TermEval` and `ReassignIn`, which sets the scene for the complete disposal of the former notion.

It should be noted that the issue of free occurrences can be arguably regarded as a

hindrance, with several papers either devoted to mitigate (or even eliminate) the problem:

> The relatively complex character of these two [the second being that of term substitution] notions is a source of certain inconveniences of both practical and theoretical nature ... we shall show in this paper that ... we can simplify the formalization in such a way that the use of the notions discussed proves to be considerably reduced or even entirely eliminated ...
>
> [Tar65],

or merely devoted to treat the problem; to limit ourselves to MML: `QC_LANG3`, `QC_LANG2`, `QC_LANG4`, `CQC_LANG`, `SUBSTUT1`, `SUBSTUT2`, `CQC_SIM1`.

The argument above does not imply, of course, that introducing the concept of free occurrence of a variable in a formula is not worth the toil; it just stands as a grant (certified by machine checking) that it is not needed to provide a complete sequent calculus.

2.5 Dealing with subterms

In key points of any treatment of first-order logic, one has to extract the subterms of a term or of an atomic formula (see, e.g., 1.8.0.26 and 1.8.0.32), hence the formalization supplies a functor `SubTerms` doing this.
It is used crucially in the definition of `TermEval` and `TruthEval` functors, see section 2.6.3. Its coding will not be explicitly shown here for space reasons.

Here, we want to discuss how its construction slightly departs from standard treatments. The task at hand is plain dull: one usually does it recursively starting from literals and iterating through operational symbols, and there is not much room from alternative approaches. However, since the language is presently constructed in terms of strings and concatenation, we tried to do the job at the more general level of monoids and associative operations. We discuss briefly the idea, without displaying Mizar code.

Take a monoid $(M, \Box)$. One can easily extend the operation $\Box$ to a function $\Box\Box$ taking any finite number of arguments iteratively, for example setting

$$\Box\Box(a,b,c) := (a\Box b)\Box c, \qquad \Box\Box(a,b,c,d) := (\Box\Box(a,b,c))\Box d,$$

and so on. To do this in Mizar we introduced the functor `MultPlace`, which actually takes any binary operation (associativity is not needed yet). Consider any $X \subseteq M$, and call it *unambiguous* (similarly to [Lot02], 1.2.1) if the restriction of $\Box$ to $X \times M$ is injective:

$$\Box(x_1,m_1) = \Box(x_2,m_2) \Rightarrow x_1 = x_2, m_1 = m_2 \qquad x_1,x_2 \in X, m_1,m_2 \in M$$

Now associativity comes into play for the result:

$$\Box \text{ associative and } X \text{ unambiguous } \Rightarrow \Box\Box|_{X^n} \text{ is injective} \qquad \forall n \in \mathbb{N}, \qquad (2.2)$$

that is, unambiguity is sort-of preserved for n-tuples. Now, taking the case $M = S^*$, where S is a language, and taking as $\Box$ the concatenation (which is associative), it

is easy to show that $T_{S,0}$ is unambiguous; indeed, any one-letter strings subset of a language is unambiguous with respect to concatenation. Starting from that, and using (2.2), it is easily shown by induction that any $T_{S,m}$ is unambiguous, too; and finally:

Theorem 2.5.0.1. *T_S is unambiguous.*

Proof. Suppose $t, t' \in T_S$ and $y, y' \in S^*$ are such that $ty = t'y'$. Call m the greater among the depths of t and t'. Since $t, t' \in T_{S,m}$ and $T_{S,m}$ is unambiguous, it must be $t = t'$ and $y = y'$. □

This permits defining subterms of a term t as the n-tuple of terms $t_1, \ldots, t_n$ such that:

$$t = **(o, t_1, \ldots, t_n),$$

where o is the first operation symbol, of arity n, of the string t. Since we know that $t_1, \ldots, t_n$ all belong to T_S, which is unambiguous, we can again apply (2.2) to decree their uniqueness, which is the point. We have discussed the general idea, the exact formulation is contained inside Mizar articles.

2.6 Encoding in Mizar

In reporting here Mizar formalizations, some minor typographic changes to the original code have been made to accommodate it and make it more readable; thus the snippets reported here should not be expected to compile correctly. For the real code, please refer to Mizar articles.
For a concise reminder of the Mizar notations we will be using, refer to appendix B. An extensive tutorial specific to Mizar is [Wie06], while a systematic, up-to-date user manual is [GKN10].

2.6.1 The Language type

Here the ground mode `Language` we will be talking about all the time is defined; it is the Mizar counterpart of the structure 'language' introduced in 1.2.0.14. There is good support in MML for finite sequences (articles `FINSEQ_1` through `FINSEQ_8`), so it is natural to identify the strings of the language we are defining with the finite sequences over its carrier. The same was done originally in [RT90]. The difference is that there it has been imposed to use exclusively sequences of Kuratowski pairs of natural numbers. Moreover, the encoding of special logical symbols is "hardwired" into that scheme. Then a layer of functors and modes definitions is added to be able to refer to these pairs with more suggestive names instead of using directly the encoding.
However, there is no apparent need to impose preemptively how a first-order language should be encoded into sets, rather it seems more sensible to work only at the level of Mizar types, leaving freedom to choose what actual symbol set to use to the instantiator of the type.
Indeed, we will see that such a rigidity, imposing how to encode even only pieces of

the language happens to be troublesome for further development (see page 53). So let us start by introducing a preparatory type named `Language-like`:

```
definition
  struct (ZeroOneStr) Language-like
  (#carrier->set, ZeroF, OneF->Element of the carrier,
  adicity->Function of the carrier\{the OneF}, INT#);
end;
```

In this definition there appears yet another provision of Mizar to cope with types. `struct` is a "structured type", similar in spirit to the ones found in many programming languages (called something like aggregates, records, structures, as appropriate). It is a concise way to group a finite number of types into one entity which becomes a new type. Each entry, or selector, of the new type is denoted by an arbitrary type name. In our case, we took a pre-defined (see `STRUCT_0`) structure type, called `ZeroOneStr`, inherited all of its fields and added one more. So we end up with a quadruple consisting of an alphabet (the carrier), two distinguished symbols of it, and a arity (adicity) function. For brevity, a couple of devices are introduced here: first, `OneF` will serve as our logical connective Nor ($\downarrow$), and it will turn out convenient not to have the arity defined on it; secondly, we agree that a negative arity will denote a relation symbol, a positive arity an operation symbol, and a zero arity a literal; these two points had been already introduced in section 1.3. With this in mind, the following definitions are obvious shorthands:

```
definition
  let S be Language-like;
  func AllSymbolsOf S equals the carrier of S;
  func LettersOf S equals (the adicity of S) " {0};
  func OpSymbolsOf S equals (the adicity of S) " (NAT \ {0});
  func RelSymbolsOf S equals (the adicity of S) " (INT \ NAT);
  func TermSymbolsOf S equals (the adicity of S) " NAT;
  func LowerCompoundersOf S equals
       (the adicity of S) " (INT \ {0});
  func TheEqSymbOf S equals the ZeroF of S;
  func TheNorSymbOf S equals the OneF of S;
  func OwnSymbolsOf S equals
  (the carrier of S)\{the ZeroF of S,the OneF of S};
end;
definition
  let S be Language-like;
  mode Element of S is Element of (AllSymbolsOf S);
  func AtomicFormulaSymbolsOf S equals
    AllSymbolsOf S\{TheNorSymbOf S};
  func AtomicTermsOf S equals 1-tuples_on (LettersOf S);
end;
```

This almost suffices to encode any first-order language. We only add a couple of further features we wish to endow our new type with:

```
definition
  let S be Language-like;
  attr S is eligible means LettersOf S is infinite &
  (the adicity of S).(TheEqSymbOf S)=-2;
end;
```

These two requests impose to have access to an infinite number of letters (we do not know the length of the terms and formulas we will need to write down), and that the arity of the equality symbol is -2, as already discussed in section 1.3, and as dictated by 1.2.0.14. This automatically likens equality symbol to any other predicate symbol. However, this is true only at this stage of syntax. The equality symbol will acquire of course special meaning in evaluation, as discussed in section 2.6.4. Finally, `Language` type is:

```
definition
  mode Language is eligible (non degenerated Language-like);
end;
```

`degenerated` is an attribute inherited from the type `ZeroOneStr`, and means that the `ZeroF` and the `OneF` coincide. So we are requesting that the equality symbol and the logical connective symbol are distinguishable. For a more elegant formalization and a purely technical convenience (the deployment of registrations, see section 2.2.2), we also translate definitions in 2.6.1 attribute-wise:

```
definition
  let S be Language-like;
  let s be Element of S;
  attr s is literal means s in LettersOf S;
  attr s is low-compounding means  s in LowerCompoundersOf S;
  attr s is operational means  s in OpSymbolsOf S;
  attr s is relational means s in RelSymbolsOf S;
  attr s is termal means  s in TermSymbolsOf S;
  attr s is own means  s in OwnSymbolsOf S;
  attr s is ofAtomicFormula means s in AtomicFormulaSymbolsOf S;
end;
```

Too simple an encoding

We want to hint at an alternative definition for the `Language` type, which originally was adopted for its further simplicity, but then deprecated and removed for reasons we will discuss. It was modeled after the idea that, looking at definition 1.2.0.14, there is no reason to separate the concept of a language and its arity, with the latter being able to carry an almost full description of the language itself in ZF. So, instead of using a higher level, structured type to declare the type `-Language`, initially the code relied on a simpler definition based on the *Function* type, which is one of the most basic and rich in already-made results inside MML:

```
definition
let f be Function;
```

```
attr f is eligible means :DefEli: f"{0} is infinite;
end;
definition
mode lang is eligible INT-valued Function;
end;
definition
let S be lang;
func OwnSymbolsOf S equals dom S;
coherence;
end;
notation
let S be lang;
synonym TheEqSymbOf S for OwnSymbolsOf S;
end;
end;
definition
let S be lang;
func TheNorSymbOf S equals {TheEqSymbOf S};
coherence;
end;
definition
let S be lang;
func AllSymbolsOf S equals
OwnSymbolsOf S \/ {TheEqSymbOf S} \/ {TheNorSymbOf S};
coherence;
end;
definition
let S be lang;
mode Element of S is Element of AllSymbolsOf S;
end;
```

This definition presents some nice aspects:

- Relying straightforward on `Function` type, the type `lang` presents a terse definition, and, thus and most importantly, carries very little work to show existence of entities: it is to be noted that in Mizar one has to prove, in the end, existence of any construct he introduces.
- The conditions

  1. `TheEqSymbOf S <> TheNorSymbOf S` (see request (2) of 1.2.0.14),
  2. `not TheEqSymbOf S in OwnSymbolsOf S`, and
  3. `not TheNorSymbOf S in OwnSymbolsOf S`

  are automatically honored, since Tarski-Grothendieck axioms easily allow to show, respectively:

  1. `X <> {X}`,
  2. `not X in X`,

3. `not {X} in X`

for any set `X`.

So we have conditions (2) and (3) of definition 1.2.0.14 already satisfied, the former automatically and the latter via an explicit, yet posing little difficulties to be existentially proved, attribute `eligible`, thus fulfilling the same tasks of the attribute of the same name in the ultimate Mizar code. The remaining condition (1) in definition 1.2.0.14 was actually not imposed at all; rather, the arity of the language was successively overlaid with an `ar` functor based on it, and which was subsequently used in its place:

```
definition
let S be lang, s be Element of S;
attr s is own means :DefOwn: s in OwnSymbolsOf S;
attr s is ofAtomicFormula means s in AtomicFormulaSymbolsOf S;
end;
definition
let S be lang;
let s be ofAtomicFormula Element of S;
func ar s equals
S.s if s is own
otherwise -2;
coherence;
consistency;
end;
```

Actually, an utterly similar `ar` functor, for the respective `Language` mode, is still present in current Mizar code and largely preferred to direct invocation of `adicity` function because the former is handier to typewrite and leaves to Mizar the burden of checking its argument having the correct type. It looks like the original definition of language given above was neater and required less preliminary work, so why has it been replaced by `Language`? The trouble with this definition becomes apparent when trying to restrict or extend a language. In a handful of key steps along the proof of satisfiability theorem, and of Löwenheim-Skolem, we needed to apply the following scheme: take two languages agreeing on some common symbols (typically because one is the restriction/extension of the other), and apply coincidence lemma on a formula consisting only of some of those symbols to conclude that it is a formula in both languages, and that its evaluations in two interpretation of the respective languages coincide. This kind of reasoning is fundamental in the following points:

- In eliminating the demand for $\#^{-1}(\{0\})$ to be infinite from 1.11.0.7 in proof of 1.11.0.11. In turn, the coincidence lemma occurs twice there, once in the main proof, to pass through *restriction* from an interpretation of S_N to one of S, and once in the subproof, to pass through *extension* from an interpretation of S to an interpretation of S_N, thus in the opposite verse as before.
- In the proof of 1.11.0.13, to restrict a generic language to the countable one made by the symbols appearing in a countable set of formulas, suitable to be applied 1.11.0.12, and in extending it back, to supply the interpretation thus found as the witness for the thesis.

Obviously, for the coincidence lemma to work, the special symbols, that is $\equiv$ and $\downarrow$, of the two languages must coincide (see 1.11.0.10). This fails to hold in the definition above; indeed, one is *granted* that this will not happen, unless the two languages are the same. Indeed, explicitly constructing the Mizar representations of $\equiv$ and $\downarrow$ from a given language is a form of the rigid "hardwiring" we wanted to depart from, as explained in motivating our work: see the beginning of section 2.6.1.

2.6.2 Syntax and semantics

The main objects introduced in this section are the three functors `-termsOfMaxDepth`, `-formulasOfMaxDepth`, `-TruthEval` and the type `Interpreter`. They are the counterparts of the entities presented in 1.2.0.16, 1.2.0.18, 1.8.0.27 and 1.8.0.20, respectively, and have the fundamental roles of describing the sets of terms and formulas of a given (or smaller) depth, of defining what is an interpretation, and of evaluating a term or a formula in a given interpretation. For the sake of convenience, let us introduce a dedicated type for the generic S-string:

```
definition
  let S be Language;
  mode string of S is Element of ((AllSymbolsOf S)*\{{}});
end;
```

The present construction will be split in stages: first atomic terms (already introduced in 2.6.1), then terms inductively, and finally atomic formulas. Let us start with an auxiliary function performing the basic construction for polish notation, that is, appending an n-tuple of strings to a leading symbol according to its arity:

```
definition
  let S be Language,s be ofAtomicFormula Element of S;
  let Strings be set;
  func ar(s) -> Element of INT equals (the adicity of S).s;
  func Compound(s,Strings) -> Subset of (AllSymbolsOf S)*\{{}}
  equals
    {<*s*> ^ ((S-multiCat).StringTuple) where
    StringTuple is Element of (AllSymbolsOf S)**:
    rng StringTuple c= Strings &
    StringTuple is (abs(ar(s)))-long};
end;
```

Here, `S-multiCat` is a dedicated function which concatenates tuples of strings, and renders the mapping $**$ introduced on page 3. Roughly speaking, it is the finite iteration of the functor `^`. Now recursive construction of terms is straightforward:

```
definition
  let S be Language;
  func S-termsOfMaxDepth ->
  Function of NAT,bool((AllSymbolsOf S)*\{{}})
  means dom it=NAT & it.0 = (AtomicTermsOf S) & for n being Nat
    holds it.(n+1) = (union {Compound(s,it.n)
```

```
    where s is ofAtomicFormula Element of S:s is operational}
    ) \/ it.n;
  func AllTermsOf S equals union rng (S-termsOfMaxDepth);
end;
```

Again, let us rephrase above definitions in terms of attributes:

```
definition
  let m be Nat, S be Language, w be string of S;
  attr w is m-termal means  w in S-termsOfMaxDepth.m;
  let w be string of S;
  attr w is termal means w in AllTermsOf S;
  attr w is atomic means
    ex s being relational Element of S,
    V being abs(ar(s))-long Element of (AllTermsOf S)* st
      w=<*s*>^(S-multiCat.V);
end;
```

2.6.3 Saving work: completing syntax and doing semantics, concurrently

Definitions in 2.6.2 are the Mizar version of definitions up to 1.2.0.16. Now, instead of proceeding with the syntax of non-atomic formulas, we digress to start concurrently putting forth some building blocks of semantics. We will then be able to define both syntax and semantics of non-atomic formulas in one shot, taking advantage of the fact that, in contrast to the building of terms, the compounders to derive higher-level formulas from lower-level ones are fixed and well-known. The fact of having reduced them to just two types (that is, one logical connective and one existential quantifier) will ease the job. This strategy saves a good deal of work for our purpose. First, we start with defining what is an interpretation of a `Language S` in a non empty set `U` (standing for universe). The definition is similar to the one given in [EFT84], only since we don't make distinction between 0-arity compounders (constants) and variables symbols, the distinction made there between interpretation, structure and assignment vanishes too. Also, we separate the universe from the interpretation (the corresponding type is called `Interpreter`; in informal talking we will use both words), more precisely, we make the latter a type dependent on the former. Here, too, we proceed gradually:

```
definition
  let S be Language, U be non empty set,
  s be ofAtomicFormula Element of S;
  mode Interpreter of s, U ->
  Function of (abs(ar(s)))-tuples_on U, U\/BOOLEAN means
    it is Function of (abs(ar(s)))-tuples_on U, BOOLEAN
  if s is relational otherwise
    it is Function of (abs(ar(s)))-tuples_on U, U;
end;
```

It is worth noting that in case of a literal (0-arity) symbol `s`, the interpreter of s,U reduces to a function from $\emptyset$ into an element of U. So, the assignment of a literal, instead of being directly a constant of u of U, is rendered as a function `{{}} --> u`, see 1.8.0.23. This is convenient for reducing the cases in subsequent proofs and definitions from three (positive, negative and zero arity) to two (negative and non negative arity). Now the definition of an interpreter of the whole alphabet is straightforward:

```
definition
  let S be Language, U be non empty set;
  mode Interpreter of S, U -> Function means
  for s being own Element of S holds
    it.s is Interpreter of s, U;
end;
definition
  let S be Language, U be non empty set, f be Function;
  attr f is (S,U)-interpreter-like means
  f is Interpreter of S,U & f is Function-yielding;
:: Function-yielding not fundamental;
:: added for technical convenience
end;
definition
  let S be Language, U be non empty set;
  func U-InterpretersOf S equals {f where f is
  Element of Funcs(OwnSymbolsOf S, PFuncs(U*,U\/BOOLEAN)):
  f is (S,U)-interpreter-like};
end;
```

Before going on we introduce two further constructs: the first is the standard Mizar functor (`FUNCT_4:def 1`) `+*` which 'pastes' two function `f` and `g` into a function `f +* g` defined on the union of their domains, with `g` (the right term) prevailing in case of conflicts: a generalization of it to relations was introduced in 1.1.0.6.

The second is the functor `ReassignIn` which implements the operator changing the assignment of a single literal in a given interpretation, defined in 1.8.0.25 and examined thoroughly in section 3.2.

Now, building a functor `I-AtomicEval phi` yielding the truth value of the *atomic* formula `phi` in the interpretation `I` is standard practice, and the corresponding code is omitted here. As anticipated, we rather want to indulge on the interpretation of non atomic formulas. Usually, one has to do first a recursive definition of the set of wffs, then another recursive definition to evaluate a wff in a given interpretation. The idea here is to do both in one single recursive definition. This technically can be done by having, as an object of the recursive definition, a partial function, here called `F` provisionally for brevity, such that, for any natural `mm`, `F.mm`

- It has as a domain exactly the cartesian product of `U-InterpretersOf S` with the set of wff of depth not exceeding `mm`.
- On that domain it maps a pair (interpretation, string) into the right truth value.

We are thus working on a higher level, where also the interpreter `I` is a variable which gets evaluated together with a wff to return a truth value; only `L` and `U` are fixed parameters. For this reason, we first need a tedious but necessary step to transform `I-AtomicEval phi` from a functor into a *function* of `I` and `phi`, named `S-TruthEval U` (its name is regretfully not too descriptive):

```
definition
  let S,U;
  func S-TruthEval(U) -> Function of
  [: U-InterpretersOf S, AtomicFormulasOf S :], BOOLEAN
  means for I being Element of U-InterpretersOf S,
  phi being Element of AtomicFormulasOf S holds
    it.(I,phi)=I-AtomicEval(phi);
end;
```

For the same reason, in Mizar code the name of the functor `F` contains only `S` and `U`, and is `(S,U)-TruthEval`; so we can get the expected behaviour for it via the fundamental definition:

```
definition
  let S be Language, U be non empty set;
  func (S,U)-TruthEval -> Function of NAT, PFuncs
  ([:U-InterpretersOf S, (AllSymbolsOf S)*\{{}}:], BOOLEAN)
  means it.0=S-TruthEval(U) & for mm being Element of NAT holds
  it.(mm+1)=G(it.mm) +* it.mm;
end;
```

At each step the partial function `(S,U)-TruthEval.mm`, which applied to the generic pair `[:I, phi:]` yields a defined, and correct, truth value if and only if `phi` is of depth not exceeding `mm`, is extended by the operator `G`, which of course must yield a partial function of domain extended to the wffs of depth `mm+1`. So the task is now the construction of `G`. We divide the problem in two simpler parts, taking care respectively of the existential symbol and of the NOR symbol separately, so that `G(it.mm)` in the actual Mizar definition is written as

```
ExIterator(it.mm) +* NorIterator(it.mm)
```

Let us illustrate only the construction of `ExIterator g` alone: the idea behind the other half is the same. Here `g` is a generic, appropriate `PartFunc`. We said that `ExIterator` has to take care simultaneously that the `PartFunc` it returns has both the right domain and the right output on it, based on `g`. This does not mean that we cannot further divide the problem into simpler parts: the definition of `ExIterator g` will actually specify only the correct domain, delegating the evaluation to yet another functor `-ExFunctor`:

```
definition
  let S be Language, U be non empty set;
  let g be Element of PFuncs
  ([:U-InterpretersOf S, (AllSymbolsOf S)*\{{}}:], BOOLEAN);
  func ExIterator(g) -> PartFunc of
```

```
  [:U-InterpretersOf S, (AllSymbolsOf S)*\{{}}:],BOOLEAN means
    (for x being Element of U-InterpretersOf S,
    y being Element of (AllSymbolsOf S)*\{{}} holds
    ([x,y] in dom it iff (
    ex v being literal Element of S, w being string of S st
    [x,w] in dom g & y=<*v*>^w
     ))) &
  (for x being Element of U-InterpretersOf S,
  y being Element of (AllSymbolsOf S)*\{{}} st [x,y] in dom it
    holds it.(x,y)=g-ExFunctor(x,y));
end;
```

We have indented the part of definition which actually does something (i.e. the specification of the domain, as we were just saying); it does that something quite trivially, too. Also trivial is the action of the functor `-ExFunctor(x,y)` to which we delegated the semantical part:

```
definition
  let S be Language, U be non empty set, f be PartFunc of
  [:U-InterpretersOf S, (AllSymbolsOf S)*\{{}}:], BOOLEAN;
  let I be Element of U-InterpretersOf S;
  let phi be Element of(AllSymbolsOf S)*\{{}};
  func f-ExFunctor(I,phi) -> Element of BOOLEAN equals
  TRUE if ex u being Element of U, v being literal Element of S
    st (phi.1=v & f.((v,u) ReassignIn I, phi/^1)=TRUE)
  otherwise FALSE;
end;
```

Just notice that this functor is expected to be accurate only when yielding `TRUE`, since otherwise it could yield `FALSE` when actually it is supposed to be undefined. This is not a problem anymore, since the previous definition already took care of that matter.

Now the significant part of the work is done: all the syntactical and semantical knowledge is thus stored in `(S,U)-TruthEval`, we just may want to rearrange it in a more accessible way, a task with which we end this section. First, we can go back to the lower level and get a function of just the string we want to evaluate:

```
definition
  let S be Language, U be non empty set, m be Nat;
  let I be Element of U-InterpretersOf S;
  func (I,m)-TruthEval ->
  Element of PFuncs((AllSymbolsOf S)*\{{}},BOOLEAN)
  equals (curry ((S,U)-TruthEval.m)).I;
end;
```

Information about both syntax and semantics is now carried by `(I,m)-TruthEval` in respectively its domain and its return value, so:

```
definition
```

```
  let S be Language, m be Nat, w be string of S;
  func S-formulasOfMaxDepth m ->
  Subset of ((AllSymbolsOf S)*\{{}}) means
  for U being non empty set,
  I being Element of U-InterpretersOf S holds
    it=dom (I,m)-TruthEval;
  attr w is m-wff means  w in S-formulasOfMaxDepth m;
  attr w is wff means ex m st w is m-wff;
  func AllFormulasOf S equals
  {x where x is string of S: ex m st x is m-wff};
end;

definition
  let S be Language, U be non empty set;
  let I be Element of U-InterpretersOf S, w be wff string of S;
  func I-TruthEval w -> Element of BOOLEAN means
  for m being Nat st w is m-wff holds it=((I,m)-TruthEval).w;
end;
```

Here only the independence of `dom (I,m)-TruthEval` on `I` and `U` needs to be shown to finally be able to evaluate the truth value of a wff formula, which is omitted here. Let us end this part with stating the remaining semantical definitions implied in the statement of Löwenheim-Skolem and completeness theorems, both traditionally indicated by the double turnstile $\models$; the satisfaction relation (cmp. 1.8.0.29):

```
definition
  let U be non empty set, S be Language;
  let I be Element of U-InterpretersOf S; let X be set;
  attr X is I-satisfied means
  for phi being wff string of S st phi in X holds
    I-TruthEval phi=1;
end;
```

and the logical implication (entailment, cmp. 1.11.0.15):

```
definition
  let X be set, S be Language, phi be wff string of S;
  attr phi is X-implied means
  for U being non empty set,
  I being Element of U-InterpretersOf S st
  X is I-satisfied holds I-TruthEval phi=1;
end;
```

2.6.4 Free interpretation

The free interpreter of a given operational symbol s of arity n of a Language S is the operation on the set of n-tuples of terms of S obtained by concatenating the tuple and appending it to the symbol s. Obviously the result is again an element of the set of all terms of S, which now acts as a universe and makes this operation an interpreter as of 2.6.3.

If we add to the picture an arbitrary set X of formulas of S we can talk also of the free interpreter of a relational symbols r of S, of arity $-n \in \mathbb{Z}^-$. In this case an n-tuple of terms is evaluated TRUE if and only if the atomic formula obtained by concatenating and appending to r (the same job done in previous case) belongs to X.

```
definition
  let X be set, S be Language;
  let s be ofAtomicFormula Element of S;
  func X-freeInterpreter(s) -> Interpreter of s,(AllTermsOf S)
    equals s-compound |(abs(ar(s))-tuples_on(AllTermsOf S))
  if not s is relational otherwise
    chi(X,AtomicFormulasOf S) *
    (s-compound | (abs(ar(s))-tuples_on (AllTermsOf S)));
end;
```

It is worth noting that this definition is also applicable to the equality symbol. This does not matter since, for *any* interpreter, the evaluation of any $\equiv$ atomic formula is overridden at the level of the definition of `-TruthEval` to give the correct value. This is indeed what is meant when talking about a language with equality. The functor `-compound` appearing above is introduced to aid the typing and has a trivial definition (see 2.6.2 for `-multiCat`):

```
definition
  let S be Language, s be Element of S;
  func s-compound -> Function of ((AllSymbolsOf S)*\{{}})*,
  (AllSymbolsOf S)*\{{}} means for V being Element of
  ((AllSymbolsOf S)*\{{}})* holds it.V = <*s*>^(S-multiCat.V);
end;
```

And finally here is the free interpretation over all the symbols of S, with `AllTermsOf S` as universe.

```
definition
  let S be Language, X be set;
  func (S,X)-freeInterpreter ->
  Element of (AllTermsOf S)-InterpretersOf S means
  dom it=OwnSymbolsOf S & for s being own Element of S holds
    it.s=X-freeInterpreter(s);
end;
```

2.6.5 Justification of ruleset choice

The complete ruleset appearing in the statement of 1.11.0.18 has formed as a result of the process of Mizaring completeness theorem. This means that, as the proof of the latter is staged into a string of roughly escalating results, each rule has been gradually introduced when the previously introduced ones no longer sufficed to proceed. This way, a tight bound between each intermediate result and the corresponding needed subset of rules have been established, and consequently a hierarchy among rules have been established; for example:

1. rules $R_{=}, R_{\underset{=}{\leftrightarrow}}, R_{\underset{=}{\Rightarrow}}$ are needed for $\overset{D}{\underset{X}{\sim}}$ to be an equivalence relation (see 1.9.2.7),
2. $R_{\underset{=}{\leftrightarrow}}, R_{=}, R_{+}, R_{\mathcal{R}}$ are needed for it to be compatible with Φ_X (see 1.9.3.2), so that
3. rules $R_{=}, R_{\underset{=}{\leftrightarrow}}, R_{\underset{=}{\Rightarrow}}, R_{+}, R_{\mathcal{R}}$ are needed to merely *define* the Henkin interpretation,
4. rules $R_0, R_{=}, R_{\underset{=}{\leftrightarrow}}, R_{\underset{=}{\Rightarrow}}, R_{+}, R_{\mathcal{R}}$ are needed for this interpretation to be a model of the *atomic* formulas of X (1.9.4.3),
5. rule $R_{\underset{\exists}{\rightarrow}}$ permits extension of result as from point (4) to existential formulas like $v\varphi$, while
6. rule $R_{\downarrow}$ permits to extend point (4) to non-existential, non-atomic formulas like $\downarrow \varphi_1\varphi_2$.
7. Since the extension as from points (5) and (6) pertain to a witnessed and expanded theory, we use only rules $R_{\cup}, R_c, R_{\underset{\exists}{\leftarrow}}, R_{=}$ to complete a theory with witnesses, and
8. we use only rules $R_{\cup}, R_c$ to expand a theory into a closed one, so that
9. the ruleset appearing in satisfiability theorem's statement, 1.11.0.9, are exactly the one needed to prove it.
10. Rule $R_{\not\vdash}$ has to be added to the remaining only to prove non-negative formulas entailed by a consistent theory (1.11.0.18).

Rules can thus be precisely tiered according to their functional role during the various proofs.

Moreover, each single subjunction of a new rule in such stepped enlargement of the ruleset was made trying to comply with secondary criteria such as simplicity and minimality: axioms (that is, rules with no input sequents) have been preferred over rules having one, and, even more, over rules having two premisses; rules involving atomic formulas have been preferred over rules involving non-atomic formulas.

Some rules (in particular R_{+} and $R_{\mathcal{R}}$), besides complying with the above ideas, are also more formalization-friendly than the ones initially conceived (see [Cam09]), so that how to formalize back-influenced what to formalize, a phenomenon occurred several times along the realization of the whole project. Instead of the one-way dynamics (from human to machine) one could expect when starting digging into formalization, this turned into a sort of feedback leading the human to rethink and rephrase along the way what he is formalizing. Every time this happened, the final outcome was always tidier and neater than the initial idea; some reflections on this facet of formalization are in section 3.5.

Admittedly, R_{+} and $R_{\mathcal{R}}$ are a bit clumsy to write down, but their proof-theoretical weakness turned out to be quite helpful in easing formalization.
Anyway, writing derivation rules in the manner above is like drawing diagrams, in that their goal is to communicate to another human how the rule works; what matters is the formalizability, and maybe the computability (which is likely to be

good if the former is), so we should not worry about the appearance of those two rules.

Given the guiding ideas according to which we formed our ruleset, and for the reasons exposed in section 2.4, it is therefore natural to wonder whether we can dispense from these notions, or if we can provide simplified versions of them. We could not help using the notion of term substitution in $R^{\rightarrow}_{\exists}$; however, the form of $R^{\leftarrow}_{\exists}$ presents two notable simplifications:

- Only the trivial literal-with-literal form of substitution (simple substitution, 1.1.0.8) appears.
- There is no request on the freeness of the occurrence of the substituted letter.

2.6.6 Sequents and rules

We first define what sequents are in just a plain way:

```
definition
  let S be Language; func S-sequents equals
  {[antecedent,succedent] where
  antecedent is Subset of AllFormulasOf S,
  succedent is wff string of S: antecedent is finite};
end;
```

Only observe that `antecedent` is an (unsorted) finite set, not a n-tuple or a bag.

Since the common way of representing sequent derivation rules, as already noticed, has more the nature of a diagram rather than that of a precise formulation, encoding them has presented a number of fundamental design choices. When starting from scratch, as in this case, one should put an effort in laying down a structure with enough flexibility and generality to last in time and possibly be reused for other purposes.

The first decision regarded modularization: the framework specifying what a rule is and its general properties has been separated from the description itself of the single rule *and* from the definition of derivability. MML presents at least two further formalizations of a proof system: see definitions of `is_a_correct_step_wrt` inside `CQC_THE1` and of `is_a_correct_step` inside `CALCUL_1`. Both adopt a monolithic, less articulated approach, simply hardcoding inside the definition itself the possible cases admitted by each single calculus rule via Mizar `if` statements. A proof is deemed correct if each step of it is correct according to the above cluster of cases. This is arguably another instance of rigidity in a basic definition, like the one we complained about in justifying the introduction of a new encoding of language (see section 2.6.1).

Here are some benefits brought by our modular approach:

- Definitions are terse and readable, compared with other approaches like those of `CALCUL_1` and `CQC_THE1`, see below.
- The effect of allowing or forbidding the use of a rule can be studied. Indeed, here for each result proved the single rules needed are resolved.

- Possible expansion upon this schemes would be feasible; e.g. for applying logic flavors other than classical one.

So we first define a framework in which to deal with rules by specifying an abstract `Rule` type as done in 1.4.0.22:

```
definition
  let S be Language;
  mode Rule of S is
    Element of Funcs (bool (S-sequents), bool (S-sequents));
  mode RuleSet of S is
    Subset of Funcs (bool (S-sequents), bool (S-sequents));
end;
```

One should think of a `Rule` as the function mapping a set X of sequents into the set of all sequents obtainable by applying the rule to all the sequents in X .

Having to do generally with deductions using several rules in succession, we introduce the functor `OneStep` to specify all the sequents derivable from some starting sequents using only one rule of a given `RuleSet D`, as in 1.5.0.3.

```
definition
  let D be RuleSet of S;
  func OneStep(D) -> Rule of S means
  dom it = bool (S-sequents) &
  for Seqs being set st Seqs in dom it holds
    it.Seqs = union ((union D) .: {Seqs});
end;
```

With that, we have started specifying how to pass from rules to derivations, and the next definition will complete the job. Sequent calculus separates the concepts of formal derivability and of provability, so we have two distinct, corresponding attributes as well; the first (to be compared with 1.5.0.4) is applied to a sequent and certifies it to be derivable from an initial set of sequents, while the second (see 1.5.0.5) applies to a formula and witnesses it is the tail of a sequent derivable from no assumptions and whose premises are given:

```
definition
  let S be Language, D be RuleSet of S, Seqs1, Seqs2 be set;
  attr Seqs2 is (Seqs1,D)-derivable means
    Seqs2 c= union (((OneStep D) [*]) .: {Seqs1});
  let X,phi be set;
  attr phi is (X,D)-provable means
    ex seqt being set st
      (seqt‘1 c= X & seqt‘2 = phi & {seqt} is ({},D)-derivable)
end;
```

Note how the passage from `OneStep` to derivability leverages some most general constructs as `union`, `[*]` and `.:` (cfr appendix (B) for their standard notation equivalents). This would have not been possible without having detached the notion

of rule from that of provability. Had not we done that, we probably would have ended up to setting some dedicated construction to describe a derivation, including in it an in-line (and verbose) condition of correctness, as it happens in `CQC_THE1` (see definitions of `Proof_Step_Kinds` and `is_a_correct_step_wrt`) and in `CALCUL_1` (see the definition of `is_a_correct_step`). This latter kind of formalizations is not likely to bring any formalization useful outside of its scope and seems much harder to work with. It seems arguable, however, that the original choice of rigidly encoding the language (see 2.6.1) encourages rigidity as in the constructs just cited. On the other hand, as stressed in other circumstances, our approach leads to possibly useful by-products of general interest regarding the general objects occurring in definitions: see section 3.2.

Now we want to actually code the rules given in section 1.4.1.1 in this framework. The difficulties in encoding a general definition of derivation rule arise from how they are customarily represented; that is, in a diagrammatic form leveraging on the excellent pattern-matching capabilities of the human reader. These diagrams operatively represent the mechanics of a rule by representing how formulas, or parts of formulas, get altered when passing from the input to the output of a rule. Usually the manipulations thus represented are limited to string concatenations and substitutions, and are possibly 'decorated' with side-conditions (typically regarding the demand of some literal not occurring free inside some formulas occurring in sequents). In other proof checkers (e.g. Isabelle and HOL variants in general, see section 1.2 of [Wie09]) there is stronger support for computations and automation, which is just what we would need here (as done in [Gor09] with Isabelle).

In Mizar, however, there is just set theory: we have therefore to express a rule in this language; one does not have a provision to compute a function, one can just describe a function by encoding its graph in set theory. Similarly, we cannot compute a rule as its diagram suggest; instead, we must set-theoretically describe what sequents it can associate to a given set of sequents. This is why the type `Rule` has been defined as from Mizar code above. With such an approach, doing even most elementary derivations becomes extremely tiresome: every single rule application must be validated by formally checking it satisfies the corresponding Mizar predicate (see section 2.6.7). With no other provision to do sequent calculus, any subsequent Mizar formalization would probably have been much tougher. Luckily, we will find out a scheme to overlay raw rule definitions with a much more friendly calculus based on Mizar's functorial registrations: see section 2.6.8. On the other hand, even without this overlay, this merely descriptive method presents at least one advantage over the computational method:

> The disadvantage is that there is no explicit encoding of a derivation. The derivation is kept implicitly by the proof-assistant and we cannot manipulate its structure. [Gor09]

We, on the contrary, have full control on a derivation: indeed each derivation will be hand-crafted into single rule application steps.

2.6.7 How to define a single specific rule

A slight nuisance we have to face preliminarily is given by the fact that the symbol set of Mizar is pure ASCII, which forced to translate the names of the rules introduced

in 1.4.1.1 and elsewhere into plain text, as from the following table

Rule0	R_0
Rule1	$R_\cup$
Rule2	$R_=$
Rule3a	$R_{\underset{=}{\Rightarrow}}$
Rule3b	$R_{\underset{=}{\leftrightarrow}}$
Rule3d	R_+
Rule3e	$R_\mathcal{R}$
Rule4	$R_{\underset{\exists}{\rightarrow}}$
Rule5	$R_{\underset{\exists}{\leftarrow}}$
RuleNor	$R_\downarrow$
Rule8	R_c
Rule9	$R_\not$

We try to separate the jobs of typing from that of actually specifying how a rule works, by proceeding in stages.

First we specify the core of the rules as Mizar predicates (which were introduced in section 2.2.3); compare this with their definition 1.4.1.1 and with their customary representation of page 10:

```
definition
let Seqts be set; let S be Language; let seqt be S-null set;

pred seqt Rule0 Seqts means seqt‘2 in seqt‘1;

pred seqt Rule1 Seqts means ex y being set st y in Seqts &
y‘1 c= seqt‘1 & seqt‘2 = y‘2;

pred seqt Rule2 Seqts means seqt‘1 is empty &
ex t being termal string of S st
  seqt‘2 = <* TheEqSymbOf S *> ^ t ^ t;

pred seqt Rule3a Seqts means
ex t1,t2,t3 being termal string of S, x being set st
(seqt=[{<*TheEqSymbOf S*>^t1^t2,<*TheEqSymbOf S*>^t2^t3},
<*TheEqSymbOf S*>^t1^t3]);

pred seqt Rule3b Seqts means
ex t1,t2 being termal string of S st
  seqt‘1 = {<*TheEqSymbOf S*>^t1^t2} &
  seqt‘2 = <*TheEqSymbOf S*>^t2^t1;

pred seqt Rule3d Seqts means
ex s being low-compounding Element of S,
T,U being (abs(ar(s)))-element Element of (AllTermsOf S)* st
  (s is operational & seqt‘1=
  {<*TheEqSymbOf S*>^(TT.j)^(UU.j) where
```

```
  j is Element of Seg abs(ar(s)),
  TT,UU is Function of Seg abs(ar(s)), (AllSymbolsOf S)*\{{}}
  : TT=T & UU=U}
  & seqt‘2=<*TheEqSymbOf S*>^(s-compound(T))^(s-compound(U)));

pred seqt Rule3e Seqts means
ex s being relational Element of S,
T,U being (abs(ar(s)))-element Element of (AllTermsOf S)* st
  (seqt‘1={s-compound(T)} \/
  {<*TheEqSymbOf S*>^(TT.j)^(UU.j) where
  j is Element of Seg abs(ar(s)),
  TT,UU is Function of Seg abs(ar(s)), (AllSymbolsOf S)*\{{}}
  : TT=T & UU=U}
  & seqt‘2=s-compound(U));

pred seqt Rule4 Seqts means
ex l being literal Element of S,
phi being wff string of S,
t being termal string of S st
  seqt‘1={(l,t) SubstIn phi} & seqt‘2=<*l*>^phi;

pred seqt Rule5 Seqts means ex v1,v2 being
(literal Element of S), x being set, p being FinSequence st
seqt‘1=x \/ {<*v1*>^p} & v2 is (x\/{p}\/{seqt‘2})-absent &
[x\/{(v1 SubstWith v2).p},seqt‘2] in Seqts;

pred seqt RuleNor Seqts means
ex phi1, phi2, phi3, phi4 being wff string of S st seqt=
[{<*TheNorSymbOf S*>^phi1^phi2, <*TheNorSymbOf S*>^phi3^phi4},
<*TheNorSymbOf S*>^phi2^phi3];

pred seqt Rule8 Seqts means
ex y1,y2 being set, phi,phi1 being wff string of S st
y1 in Seqts & y2 in Seqts & y1‘1=y2‘1 & y1‘2=phi1 &
y2‘2 = <* TheNorSymbOf S *> ^ phi1 ^ phi1 &
seqt‘1\/{phi}=y1‘1 & seqt‘2=<*TheNorSymbOf S*>^phi^phi;

pred seqt Rule9 Seqts means
ex y being set, phi being wff string of S st
y in Seqts & seqt‘2=phi & y‘1=seqt‘1 & y‘2=xnot (xnot phi);
end;
```

In the definiens of last rule we took advantage, for a matter of convenience, of the Mizar analog of the map seen in 1.10.1.1:

```
definition
let S be Language, w be string of S;
func xnot w -> string of S equals <*TheNorSymbOf S*>^w^w;
end;
```

We want at this stage to reduce at a minimum the role of types, to concentrate on the mechanics of the rule, so we declare the starting sequents, represented by `Seqts`, as an untyped variable (a set); at the same time, to do the correct typing later, we need to preserve a link to the type of the specific language `S` we are referring to, so we introduce a fake attribute `-null`, and save it in the variable `seqt`, which represents the derived sequent (the "denominator") of the rule.

Now we pass from the predicate `RuleX` to a rule as specified by `Rule` type; let us take `Rule0` for example:

```
definition
  let S be Language,
  R be Relation of bool (S-sequents), S-sequents;
  func FuncRule(R) -> Rule of S means
  for inseqs being set st inseqs in bool (S-sequents) holds
    it.inseqs=
    {x where x is Element of S-sequents:[inseqs,x] in R};
end;
registration
  let S be Language;
  cluster -> S-null Element of S-sequents;
end;
definition
  let S be Language;
  func P0(S) -> Relation of bool (S-sequents), S-sequents
  means for Seqts being Element of bool (S-sequents),
  seqt being Element of (S-sequents) holds
    [ Seqts, seqt ] in it iff seqt Rule0 Seqts;
end;

definition
let S be Language;
func R0(S) -> Rule of S equals FuncRule(P0(S));
end;
```

When having to code many rules this scheme is convenient because one needs only to define a Mizar predicate without much worrying about typing; afterwards, the rule is easily, and standardly, converted into a `Relation` and finally applied `FuncRule`. The last couple of definitions have to be manually repeated verbatim inside Mizar code, only changing `P0(S)` to `P1(S)` and `R0(S)` to `R1(S)` (and so on for each rule...), because Mizar lacks second-order definitions. The code contains the proofs of soundness and monotonicity for all the rules above. We warn the reader that in it, the attribute `isotone` is used, since the keyword `monotone` was already in use.

2.6.8 Exploiting Mizar's functorial registrations to restore a sequent calculus

As discussed earlier, there is only one other proof checker in which a sequent calculus has been encoded, to the best of author's knowledge: Isabelle (or variants, [DG10],

[Gor09], [CMU08]), probably due to some nice facilities provided, as inductive definitions and structured proofs ([Nip03]). Mizar has fewer provisions to actually calculate things apart from small integer arithmetics; thus, the idea is to exploit its functorial registrations (see section 2.2.2), which actually do some pattern matching on a term of the first order language of Mizar: we can try to employ this capability to recognize whether a sequent is derivable from another using a given rule. Once finished, we will have adapted Mizar's powerful registrations to gain back some resemblance to a calculus, lost with the purely descriptive *definition* of derivation rules in the set theory of Mizar (given in section 2.6.6) as opposed to their computational *application* possible in Isabelle.

Preliminarily, however, we need to make more precise the definition of `-derivable` attribute: in that definition, derivability is assessed first taking all sequents derivable from an initial set of sequents using one rule of D, and exactly once (`OneStep D`). The sequents derivable from a fixed initial set of sequents are those obtainable by iterating the scheme above a finite number of times, that is its transitive closure (`[*]`). Now we want to be able to resolve that finite number of times, by defining, in parallel with 1.5.0.4:

```
definition
let S be Language, D be RuleSet of S, m be Nat;
func (m,D)-derivables -> Rule of S equals iter(OneStep D,m);
end;
```

and

```
definition
let m be Nat, S be Language, D be RuleSet of S;
let Seqts,seqt be set;
attr seqt is (m,Seqts,D)-derivable means
seqt in (m,D)-derivables.Seqts;
end;
```

This at first looked straightforward, since it seemed sufficient to replace the transitive closure operator with the iteration operator: we have constantly advocated the use of as general objects as possible also as good practice in such situations. Indeed, it turned out to be sufficient, the only shame being that no ready-made result connecting those two operators existed in MML strong enough to be useful in this case. As we insistently maintained, however, there is a good side also in this worst case, that is: some additional work had to be done, but there is good chance somebody else will use it in the future. The general result we obtained is reported in section 3.2. Here, it permits:

```
Lm18: union (((OneStep D)[*]).:{X}) = union
{(mm,D)-derivables.X where mm is Element of NAT:
not contradiction};
```

and finally, the redefinition:

```
definition
let S be Language, D be RuleSet of S; let X,x be set;
redefine attr x is (X,D)-provable means
ex H being set, m st H c= X & [H,x] is (m,{},D)-derivable;
```

The redefinition above allows to exhibit derivations (and hence proofs) in single steps, and allow finally to render most of our derivation rules as functorial registrations (which were introduced in section 2.2.2).

```
definition
let x be set; let S be Language;
attr x is S-premises-like means
x c= AllFormulasOf S & x is finite;
end;

registration
let S be Language; let H1, H2 be S-premises-like set;
let l, l1 be literal Element of S;
let phi, phi1, phi2 be wff string of S;
let t, t1, t2 be termal string of S;
cluster [Phi \/ {phi}, phi] -> (1,{},{R0(S)})-derivable set;

cluster [H1\/H2, phi] -> (1,{[H1,phi]},{R1(S)})-derivable set;

cluster {[{},<*TheEqSymbOf S*>^t^t]} -> {R2(S)}-derivable set;

cluster
[{<*TheEqSymbOf S*>^t^t1,
<*TheEqSymbOf S*>^t1^t2}, <*TheEqSymbOf S*>^t^t2]
-> (1,{},{R3a(S)})-derivable set;

cluster [{(l,t) SubstIn phi}, <*l*>^phi] ->
  (1,{},{R4(S)})-derivable set;

let l2 be (H\/{phi1}\/{phi2})-absent literal Element of S;
cluster [(H\/{<*l1*>^phi1}) null l2, phi2] ->
(1,{[H\/{(l1,l2)-SymbolSubstIn phi1},phi2]},{R5(S)})-derivable
set;

cluster [{<*TheNorSymbOf s*>^phi1^phi1, <*TheNorSymbOfs*>^phi2^phi2},
<*TheNorSymbOf s*>^phi1^phi2] ->
(1,{},{RNor(S)})-derivable set;

cluster
[{<*TheNorSymbOf S*>^phi1^phi2}, <*TheNorSymbOf S*>^phi2^phi1]
-> (1,{},{RNor(S)})-derivable set;
```

```
cluster [H null (phi1^phi2),xnot phi] -> (1,
{[H\/{phi},phi1],[H\/{phi},<*TheNorSymbOf S*>^phi1^phi2]},
{R8(S)})-derivable set;

cluster [H, phi] null 1 ->
(1,{[H, xnot (xnot phi)]},{RD(S)})-derivable set;

end;
```

Please see section 3.1 for remarks on the `null` functor, which ignores the operands on its right and serves merely syntactical, technical purposes connected with some Mizar idiosyncrasies.

Combining the one-step derivations above, one can perform standard multi-step derivations; additionally, if some particular multi-step derivation is found to occur recurrently, one can of course register it in turn into a composite, macro-like derivation (often called derived rule); for example, the following registration might be handy:

```
registration
let S be Language, t be termal string of S;
let phi be wff string of S;
cluster [{phi}, <*TheEqSymbOf S*>^t^t] ->
(2, {}, {R1(S),R2(S)})-derivable set;
end;
```

Once he has a decent set of clustered rules, one can perform a derivation in a very natural manner, close to a standard derivation of sequent calculus, especially combining them together, which is essential in calculations, permitting to transitively concatenate derivations, and moreover keeping precise track of their *depth*: the latter results stowed in the first argument of the `-derivable` attribute at the end of the derivation chain.

Here is a sample taken from `FOMODEL4` and rendering a simplest chained derivation:

```
[H1\/H2, phi] is (n+1,{[H1, phi]},{R1(S)})-derivable &
[(H1\/H2)\/(H1\/H2),phi] is
(1,{[H1\/H2,phi]},{R1(S)})-derivable; then
[H1\/H2,phi] is
(n+1+1,{[H1,phi]},{R1(S)}\/{R1(S)})-derivable by Lm28;
```

The lastly derived sequent's attribute always stores the depth of the respective derivation, in this case `n+2`. Notice that invariably, when combining at least two rules to perform multi-step derivations or to obtain a derived rule, one needs monotonicity (see definition 1.6.1.1), which accounts for the invoking of `Lm28` above.

Clearly, our original predicate-based definitions of rules, given in section 2.6.7, are much more obnoxious to deal with than this device exploiting Mizar clusters, and serve only to validate the latter, being doomed to disuse after that.

2.6.9 Definitions for readability

Tinkering with rulesets, as we did by weighing the exact needed rules in statements of results from chapter 1, is not a common practice. Usually, the ruleset is fixed in advance, with everything thereafter meant relative to that unique ruleset. As a reward, statement of theorems result terser. We of course can regain back that same advantage by introducing shorthand Mizar definitions, which make possible to state completeness theorem in the concise form seen on page 46.

```
definition
let S be Language;
func S-rules -> RuleSet of S equals
{R0(S), R1(S), R2(S), R3a(S), R3b(S), R3d(S), R3e(S), R4(S)} \/
{R5(S), RNor(S), R8(S)};
coherence;
end;

definition
let X be set, S be Language, phi be wff string of S;
attr phi is X-provable means
phi is (X,{R9(S)}\/S-rules)-provable;
end;
```

These can be regarded as placeholders, introduced to make theorem statements more mainstream, so that a casual reader will better grasp an idea of what a theorem deals with upon reading it. This is important for MML, which aims to supply a library of mathematics being human-readable, besides being machine-verified.

As a side-note, we observe that the keyword `-provable` now results overloaded to denote two distinct attributes (compare definition above with that on page 71). Mizar has no problem with that, being able to resolve which use is being made by looking at the number of the arguments accompanying the identifier (the *format*); in case this is not sufficient, it looks at both the number of arguments and at their type (the *pattern*).

Chapter 3

The formalization from a technical point of view

This chapter provides techniques and practical considerations, pertaining the practice of writing Mizar code and formalizations in general, accrued while working with the system. It features material from [CR11].

3.1 Custom automations in Mizar

When writing a Mizar formalization, a significant amount of the user's time usually goes into browsing the Mizar Mathematical Library (MML) for those results that he needs and that are already proved. Here a few techniques to reduce this time are illustrated. Let us begin by pointing out two shortcomings related to the Mizar verifier, which was introduced in section 2.2:

1. At a low level, a Mizar user has no practical way to specify the logic the Mizar verifier applies to approve an inference: no full programmability is provided, besides tweaking the source code, to plug in alternative proof systems.

2. At a higher level, there is no general provision to instruct the verifier to 'know' a generic custom-defined formula already proved, in order to avoid to list explicitly some, or all, of the labels following the keyword `by` when the writer perceives the inference as obvious, natural, or recurring so often to deserve some kind of automation.
   For example, one might want to program the verifier to 'know' the trivial set-theoretical inclusion

   $$X \cap Y \subseteq X, \tag{3.1}$$

   so as not to have to '`by`' the corresponding MML theorem in reasonings involving it.

We will not discuss the reasons and implications of these design choices: considerations on such topics can be found in [Urb06a]. Rather, we will focus on how certain Mizar features can be exploited to mitigate issue 2, which is relevant to a user from a purely practical point of view: it is frequently the case that the user knows the

steps to lay down a proof, or the statements of the needed theorems (especially when trivial or natural) and then must go and dig into the vastities of the MML to justify each of them. While this can turn out to be a highly instructive experience, it also leads to distraction and to longer formalization times, and urged the creation of a range of tools to aid the user in facing this task ([RU11], [BU04], [Urb06b], [CG07]). Here, a different, possibly complementary, approach is proposed aiming instead at reducing the occasions when he faces such a task.

Ideally, to a generic inference submitted to the verifier, one or more finite sets can be associated, each made of premisses strictly needed for the inference to be accepted (the references one *must* list following the keyword `by`).

We adopt the term *automation* to loosely indicate any device or mechanism enabling to reduce such a set, even if possibly only for some kinds of inferences.

First of all, it must be said that indeed Mizar does supply some automations natively. However, they present several constraints: they are not strong enough to instruct the verifier to blindly accept *any* already proved formula. To be more precise, the automations called *requirements*, imported using the eponymous keyword, are powerful enough to do exactly this, which is what we fancied of in item (2) of the above list. The point is that requirements are out of reach of most users, because they are hard-coded in verifier's sources by developers ([NB04], [Nau07]). The remaining Mizar provisions (see section 2.2) to introduce automations are less general, and mostly embedded in its type system; however, they are the building blocks of the methods we will see.

3.1.1 Type clustering to avoid redefinitions

Let us return to the example automation in (3.1): we would like to teach the verifier that

$$X \cap Y \subseteq X.$$

A first naive way to do that would be to redefine the output type of the functor `/\`. This can be done for whatever functor via the keyword `redefine`, subject of course to the appropriate proof. This process of 'type recasting', however, is destructive: only the last (re)definition is retained by the verifier. And indeed, MML already provides (in article`SUBSET_1`) yet another redefinition of `/\`:

```
definition
  let E, X be set; let A be Subset of E;
  redefine func A /\ X -> Subset of E;
  coherence
  proof
    ...
  end;
end;
```

which we do not want to lose. The idea then is to combine the ability of Mizar to recognize *one* type for a given term with the identification scheme seen at the end of section 2.2.1, to 'funnel' several recognized types into a single term as a result. Following an example taken, as others in the sequel, from [Cam11d] we introduce

a dummy functor symbol, a 'shadow' of the main functor symbol `/\`, let us call it `typed/\`:

```
definition
  let X,Y be set;
  func X typed/\ Y -> Subset of X equals X /\ Y;
  coherence;
end;
```

Now, if we make Mizar identify (see section 2.2.1) `X typed/\ Y` with `X /\ Y`:

```
registration
  let X,Y be set;
  identify X /\ Y with X typed/\ Y;
  compatibility;
  identify X typed/\ Y with X /\ Y;
  compatibility;
end;
```

then the two distinct typing we wanted do simultaneously co-exist:

```
now
  let Z be set; let X, Y be Subset of Z;
  X/\Y is Subset of Z; :: thanks to redefinition in article SUBSET_1
  X/\Y is Subset of X; :: thanks to registration above
end;
```

The verifier accepts both the formulas above without justification. What happens is clear: the term `X/\Y` occurring in last formula is identified with `X typed/\ Y`, which has the right type, convincing the verifier. A couple of musings:

- Generally, when employing the `identify` registration, we always do it in both verses, as above. This is to be on the safe side, as `identify` works in a not completely symmetrical manner ([GKN10], section 2.7). As observed in practice, the second identification in such cases always comes for free; that is, once the `compatibility` condition for the first one is secured, the second `compatibility` statement is validated without proof, even without starting a new `registration` ... `end;` block. Hence, not requiring much additional time, it is useful to do double identification each time. In subsequent examples we sometimes will omit transcribing the second identification, though.

- There is already an automation granting `X /\ Y = Y /\ X` without justification (this is achieved via so-called *properties*, more on which can be found in [GKN10], section 2.5). Thence, one could expect he has obtained for free also the automation `X /\ Y is Subset of Y`, via the ideal chain:

  ```
  X /\ Y = Y /\ X = Y typed/\ X.
  ```

  This will not work straightaway, however. There are two possibilities:

1. Introduce a further identification between `X typed/\ Y` and `Y typed/\ X`.
2. Introduce a further functor `/\typed` working symmetrically with respect to `typed/\`:

```
definition
  let X,Y be set;
  func X /\typed Y -> Subset of Y equals X/\Y;
  coherence;
end;
```

and then proceed with the suitable registrations.

Both approaches solve the problem providing the automation

```
X /\ Y is Subset of Y;
```

As a passing note, method (1) above suggests that identifications may replace properties in some circumstances: Mizar can be made aware of the commutativity of a given functor either via properties (as done in MML for `/\`) or by identifying a functor application with the application obtained by swapping its arguments. It would be interesting to know to what extent these two approaches are equivalent. One simple remark is that the latter has wider applicability: upon establishing commutativity property when defining `typed/\`, one gets the error:
`The result type is not invariant under swapping the arguments`,
while an identification does the job.

3.1.2 Type clustering with dummy arguments: combining type clustering with notations

We would like to repeat the scheme above for the (trivial) set-theoretical property

$$Y \subseteq X \Rightarrow X \cap Y = Y.$$

Here, however, we face a limitation of the `identify` construct we have not mentioned yet: there are formal restrictions on the functors being identified. In particular, they must have the same number of arguments, so we cannot just write:

```
registration
  let X be set, Y be Subset of X;
  identify X /\ Y with Y;
```

We just introduce a functor `null` whose only (for the time being) utility is formally to take a second argument for the mere sake of balancing things:

```
definition
  let X,Y be set;
  func X null Y equals X;
  coherence;
end;
```

```
registration
  let X be set; let Y be Subset of X;
  identify X /\ Y with Y null X;
  compatibility by XBOOLE_1:28;
  identify Y null X with X /\ Y;
  compatibility;
end;
```

The final effect is not as neat as that of section 3.1.2, in that we cannot submit the verifier simply

```
let X be set, Y be Subset of X;
X/\Y = Y;
```

This is because the verifier of course cannot guess that writing `Y` we mean `Y null X`: although the argument `X` is semantically thrown away by `null`, its presence supplies information. Indeed, Mizar can understand things the other way round:

```
let X be set, Y be Subset of X;
X /\ Y = Y null X; then
X /\ Y = Y;
```

This works.[1] Again, we have some remarks:

- The last inference works because the definition of `null` is done via `equals` rather than via `means` (see item (1) on page 41): the corresponding definition being a macro permits to take advantage of Mizar's *equals expansion*, see section 2.3.4 of [GKN10]. Note that, in order to take advantage of equals expansion for a given functor outside the file in which it is defined, that file must be imported via the `definitions` directive.
- As we said before, the aim of automations is to reduce the time devoted to searching MML, rather than to save keypresses. So this scheme is still arguably worth being applied: no `by` is needed.

The following sort of a dual of the previous registration:

```
registration
  let X be set; let Y be Subset of X;
  identify X \/ Y with X null Y;
  compatibility by XBOOLE_1:12;
  identify X null Y with X \/ Y;
  compatibility;
end;
```

permits

```
let X; let Y be Subset of X;
X \/ Y = X null Y; then X \/ Y = X;
```

[1] `then` can replace `by` when referring to the immediately preceding formula.

3.1.3 Combining dummy arguments and type clustering

The dummy argument of the functor `null` can be more than a placeholder to satisfy `identify`'s requirements. It can be used to control the desired type of a term. For example, we could redefine `X null Y` to be a `Subset of X\/Y`, and then be able to automate properties like:

```
let X, Y be set;
X null Y is Subset of X \/ Y; then X is Subset of X \/ Y;
```

However, one can do better: recall that type redefinitions are destructive, while we might want in the future `null` not to have that type. It is natural then to resort to type clustering, just seen in section 3.1.2; for example:

```
definition
  let X, Y be set;
  func X \typed/ Y -> Subset of X \/ Y equals X;
  coherence by XBOOLE_1:7;
end;

registration
  let X, Y be set;
  identify X \typed/ Y with X null Y;
  compatibility;
  identify X null Y with X \typed/ Y;
  compatibility;
end;
```

and the wanted automation is in charge.

3.1.4 Reference redirection via functorial registrations

Since functorial registration, seen in section 2.2.2, are so powerful, the idea is to reduce the most used first-order relation symbols to attributes in order to save lookups into MML.

Translating set-theoretical equality, `=`, via attribute `empty`

Let us start with the Mizar equality symbol, `=`. It can be rendered via the functor `\+\`[2] and the attribute `empty` via the result (FOMODEL0:29):

```
for X, Y being set holds X \+\ Y is empty iff X=Y;
```

This means that for every theorem in MML whose statement has the form

```
B1: term1 = term 2;
```
(3.2)

one can produce a translation like

[2] `\+\` is the set-theoretical symmetric difference, commonly denoted as Δ: $X \Delta Y = X \backslash Y \cup (Y \backslash X)$. See also appendix B.

```
term1 \+\ term2 is empty by B1, FOMODEL0:29;
```
(3.3)

This latter version has the advantage of being applicable as a functorial registration, which allows to use it without justification in subsequent proofs. Even if one needs the original version of the theorem, one can get it by referring back to FOMODEL0:29. This gives the possibility of remembering just one reference (FOMODEL0:29) in place of several references, one for each needed theorem: of course, the more theorems are translated in registrable form (3.3), the more convenient this scheme gets. As an example, XBOOLE_1:4 states associativity of \/. We then register the following:

```
registration
  let X, Y, Z be set;
  cluster ((X \/ Y) \/ Z) \+\ ( X \/ (Y \/ Z) ) -> empty for set;
  coherence by XBOOLE_1:4, FOMODEL0:29;
end;
```

Now, when we need this theorem we write:

```
let X,Y,Z be set; ((X\/Y)\/Z) \+\ (X\/(Y\/Z)) is empty; then
(X\/Y)\/Z = X\/(Y\/Z) by FOMODEL0:29;
```

XBOOLE_1 contains many such elementary results, frequently employed and having form (3.2), so it is arguably convenient to turn them into registrations. After doing that, each time the user invokes one of them, he will only need to remember at most FOMODEL0:29. Here is a list of some registrations of this kind introduced and deployed in Mizar articles FOMODEL0-4 (to save space, environments and type declarations are mostly omitted):

```
cluster ([x,y]`1) \+\ x -> empty for set;
cluster ([x,y]`2) \+\ y -> empty for set;
cluster (id {x}) \+\ {[x,x]} -> empty for set;
cluster (x.-->y) \+\ {[x,y]} -> empty for set;
cluster (id {x}) \+\ (x.-->x) -> empty for set;
cluster <*x*> \+\ {[1,x]} -> empty for set;
let p be FinSequence; cluster (<*x*>^p).1 \+\ x -> empty for set;
let m be Nat;
cluster m-tuples_on X \+\ Funcs(Seg m,X) -> empty for set;
let f,g be Function;
cluster (f+*g) \+\ (f \ [:dom g, rng f:] \/ g) -> empty for set;
cluster (f+*g) \+\ f|(dom f \ dom g) \/ g -> empty for set;
cluster (f+*g) \+\ ((f|(dom f) \ (f|(dom g))) \/ g) -> empty for set;
```

Translating set-theoretical inclusion, c=, via attribute empty

A similar translation can be done for the inclusion symbol c= into the functor \ and the attribute empty via XBOOLE_1:37:

```
X \ Y = {} iff X c= Y;
```

Here are some examples of registrations for this case:

```
cluster {x}\{x,y} -> empty for set;
cluster NAT\INT -> empty for set;
let X be set; let F be Subset of bool X;
cluster union F \ X -> empty for set;
let X,Y be set; let x be Subset of X, y be Subset of Y;
cluster x\Y \ (X\y) -> empty for set;
let m be Nat; cluster (m-tuples_on X) \ (X*) -> empty for set;
```

Translating set-theoretical membership, `in`, via attribute `empty`

The same goes with the rendering of relation symbol `in` via functors `{ }`, `\` and again attribute `empty`, thanks to:

```
for x, X being set holds x in X iff {x} \ X is empty;
```

Also for this scheme we give some examples of registrations:

```
let U be non empty set, u be Element of U;
cluster {(id U).u} \ U -> empty set;
let m,n be Nat; let p be (m+1+n)-long Element of U*;
cluster {p.(m+1)} \ U -> empty set;
```

Translating basic arithmetics into attributes

The same idea can be adapted to a broad scope of contexts. Here, it was exploited when needing some very basic arithmetical identities, like:

```
let z be zero (integer number);
cluster abs(z) -> zero (integer number);
let z1 be non zero (complex number);
cluster abs(z1) -> positive (real number);
let x,y be real number;
cluster max(x,y)-x -> non negative (real number);
```

As another application, request 1 in definition 1.2.0.14 was translated as follows for easier reference:

```
let S be Language; cluster ar(TheEqSymbOf S) + 2 -> zero number;
cluster abs(ar(TheEqSymbOf S)) - 2 -> zero number;
```

Similarly, other trivial arithmetical facts were rendered thus:

```
let v be literal Element of S; cluster ar(v) -> zero number;
let m0 be zero number; let t be m0-termal string of S;
cluster Depth t -> zero number;
let phi0 be m0-wff string of S;
cluster Depth phi0 -> zero number;
let m be Nat; let phi be m-wff string of S;
cluster m - (Depth phi) -> non negative (real number);
let phi1 be non 0wff (wff string of S);
cluster Depth phi1 -> non zero Nat;
```

We omit any further detail; some more examples are in articles `FOMODEL0-4`.

3.1.5 Definiens clustering: combining identification and equals expansion

Consider the last three registrations of section 3.1.4 involving the functor +*: recalling the idea of that section, they express three set-theoretical equalities which, as all other equalities of this form, can be used remembering just one MML reference, FOMODEL0:29, once registered. There is also a way to avoid even the need to refer to this single theorem, and make Mizar accept the corresponding equalities:

```
f \ [:dom g, rng f:] \/ g) = (f +* g);
f|(dom f \ dom g) \/ g = (f +* g);
((f|(dom f) \ (f|(dom g))) \/ g) = (f +* g);
```

straightaway. Note that MML's original definition of +* is done via means, so equals expansion cannot be used. One could redefine +* with one of the equalities above, but this would exclude the other two from automation. Instead, it is possible to keep the original definition and proceed as follows:

```
definition
  let P,Q be Relation;
  func P +*1 Q equals P \ [:dom Q, rng P:] \/ Q;
  coherence;
  func P +*2 Q equals P|(dom P \ dom Q) \/ Q;
  coherence;
  func P +*3 Q equals ((P|(dom P) \ (P|(dom Q))) \/ Q);
  coherence;
end;
```

Note that the shadow functors +*1, +*2, +*3 all accept more general arguments than its forefront functor +*: every Function is a Relation, but the opposite does not hold. For this reason we first proceed with the mutual identification of the functors defined above:

```
registration
  let P, Q be Relation;
  identify P +*1 Q with P +*2 Q;
  compatibility
    proof
    ...
    end;
  identify P +*2 Q with P +*3 Q;
  compatibility by RELAT_1:109;
end;
```

Having done so, Mizar now accepts equalities like:

```
let P, Q be Relation; P +*3 Q = P \[:dom Q, rng P:] \/ Q;
```

This means, in particular, that identifications work transitively: we have identified +*1 with +*2 and +*2 with +*3, but not +*1 with +*3. Finally, we can bind all these identifications with the forefront functor +*, and then forget about the others:

```
registration
  let f, g be Function;
  identify f +*1 g with f+*g;
  compatibility
    proof
    ...
    end;
  identify f+*g with f +*1 g;
  compatibility;
end;
```

Now the following works without justifications:

```
let f, g be Function;
f+*g = f\[:dom g, rng f:] \/ g;
f+*g = f|(dom f \ dom g) \/ g;
```

We have thus 'clustered' several definientia into the single functor `+*`.

3.2 Considerations on some formalization design issues

Awareness that thoroughly calibrating types when spelling out definitions is a key factor for a well-structured proof grew steadily during the work. If one goes too strong, by being too fussy in specifying what type of arguments a functor takes, and at some point faces the need, for example, to apply the same functor to two arguments which differ little, but do not have the same type, in this case he is forced to do double work; also, sometimes a job can be made lighter by adapting an existing type to an affine situation, and base on ready-made formalizations, instead of creating a brand new world of types and having to re-invent the wheel. On the other hand, being too light with typing one loses the advantages of a tidy formalization given by Mizar. As an example, compare the definitions of atomic wff in [RT90] and in the present work:

```
definition                          definition
  let F be Element of QC-WFF;       let S be Language;
  attr F is atomic means            let phi be string of S;
                                    attr phi is 0wff means
⋮                                   ⋮
```

The definition on the right applies to any string, and not to anything less only because inside the body of the definition there are functors requiring a string (a `FinSequence`) as arguments; on the other hand the left definition restricts the objects to which atomic attribute can be applied. This is likely to complicate forthcoming treatments. One could object that the first solution has the strength of ensuring that 'atomic' implies 'wff'. But this can be attained also in the second case by clustering (see section 2.2), which is indeed done in the formalization:

```
registration
let S be Language;
cluster 0-wff -> atomic string of S;
cluster atomic -> 0-wff string of S;
let m be Nat;
cluster m-wff -> wff string of S;
let n be Nat;
cluster (m+0*n)-wff -> (m+n)-wff (string of S);
end;
```

The heavy adoption of attributes and clusters is a trait of the present formalization[3]. Their use has a few advantages: first, a technical one, for they permit to automatically and implicitly reach conclusions which otherwise should be made explicit with a `by` statement; this also brings an advantage in terms of terseness and legibility; finally, they make type-trimming easier, allowing rich typing with relative ease.

In the present case, this is especially true for the classification of the various types of alphabet symbols: literal, compounder, relational, etc... (see 2.6.1), and for the classification of well-formed tuples, as in the example above.

A further character of this formalization is the effort to find definitions based on `equals` and `is`, avoiding those based on `means` when possible. It seems that the former encourage the reusing of pre-existing objects (functors, modes or attributes), at the price of doing the preparatory work of translating the definition to be expressed in terms of those other objects. Definitions thus obtained are arguably more neat and readable, although sometimes less immediate. For sure "`equals`" definitions have a technical advantage resembling that of attributes: they are grasped automatically by Mizar if included in the `definitions` directive, again making life easier and code terser. See [Kor09], section 3. Good examples of this method could be the definitions of the functors `===` (not reviewed here, needed in construction of `-TruthEval`), `X-freeInterpreter` (see 2.6.4), `(I,m)-TruthEval` (see 2.6.3), and `ReassignIn` (see sections 2.6.3 and 2.6.5).

The last example is interesting because it also honors the ideas introduced in section 2.4: indeed, besides having a clean, `equals`-based definition, it is first introduced for arguments of more general types than we need for our particular case:

```
definition
let x,y be set, f be Function;
func (x,y) ReassignIn f -> Function equals
f +* (x .--> ({} .--> y));
end;
```

Recalling the action of `+*` functor and how we encoded the interpretation of a literal symbol (section 2.6.3), its way of working should be clear. We are leaning of course on a definition (`+*`) given elsewhere, but this permits to use more general tools, avoid restating things, reduce the length of the definition, and, above all, reuse

[3] FOMODEL0 is the single registration-richest article in the whole MML, as checked at http://mmlquery.mizar.org/mmlquery/fillin.php?filledfilename=registrations.mqt&argument=number+1 on 31st March 2011

possible results already proven about `+*`. Even if these results were not already available in MML, proving them for a more general, pre-defined object is always better than providing a specialized result framed in a narrower context: somebody else could take advantage of them for developing possibly different areas of MML. Again, as in the first example of this section, we adapt this general definition to our needs by showing this functor returns the expected type when applied to the types we will feed it, using the powerful tool of functorial clustering (section 2.2):

```
registration
  let S be Language,U be non empty set,
  I be (S,U)-interpreter-like Function;
  let x be literal Element of S, u be Element of U;
  cluster (x,u) ReassignIn I -> (S,U)-interpreter-like;
end;
```

Indeed, as noted in section 2.4, some developments needed in the present work produced results regarding only pre-existing, more general objects: as examples, one could consider the introduction of the `-unambiguous` attribute for generic binary operations, and the related results for the generic monoids, sketched in section 2.5. Here, two more examples, taken again from `FOMODEL0` and which were missing from MML, are exhibited in view of their concise and general statement; they both derived from investigations on how to formalize sequent calculus.

The first regards the transitive closure `R[*]` of a relation `R` and states that it is both transitive and reflexive:

```
registration
  let R be Relation;
  cluster R[*] -> transitive Relation;
  cluster R[*] -> reflexive Relation;
end;
```

The second binds together the transitive closure and the iteration of a function:

```
for f being Function st rng f c= dom f holds f[*] = union
  {iter(f,mm) where mm is Element of NAT: not contradiction};
```

3.3 About the specialization of existing results

In proving 1.10.3.2, we implicitly employed the following intuitive fact:

$$\left.\begin{array}{r} Y \text{ finite} \\ \forall n \in \mathbb{N}\ X_n \subseteq X_{n+1} \\ Y \subseteq \bigcup\limits_{n\in\mathbb{N}} X_n \end{array}\right\} \quad \Rightarrow \quad \exists \overline{n} \in \mathbb{N}\ |Y \subseteq X_{\overline{n}}$$

Initially, we relied on `HENMODEL:3`, which in turn employs the ad-hoc results `HENMODEL:1` and `HENMODEL:2`, for a total of more than 250 lines of dedicated Mizar code. Actually, such specific propositions could have not been written at all, for they are predated by the more general result `COHSP_1:13`:

```
for X being non empty set, Y being set st
X is c=directed & Y c= union X & Y is finite
ex Z being set st Z in X & Y c= Z;
```

where `c=directed` substantially means somehow closed with respect to finite union, as from definition `COHSP_1:def 3`:

```
definition
  let X be set;
  attr X is c=directed means
  for Y being finite Subset of X ex a being set st
  union Y c= a & a in X;
end;
```

Now consider the theorem `COHSP_1:6` coupled with `COHSP_1:13` reported above:

```
for X being non empty set st
(for a,b being set st a in X & b in X
ex c being set st a \/ b c= c & c in X) holds X is c=directed;
```

Clearly these two results generalize `HENMODEL:3`, which runs like:

```
for f being Function of NAT,C, X being finite set st
(for n,m st m in dom f & n in dom f & n < m holds
f.n c= f.m) & X c= union rng f
ex k st X c= f.k,
```

and whose authors could have saved a fair amount of work by leveraging `COHSP_1:13` and `COHSP_1:6`. Other instances of duplicated work inside MML were noticed during the work, with this being probably the most blatant. What is more, the excessive specialization of duplicate results in `HENMODEL` makes their statement inelegant, e.g., obfuscating the simple meaning expressed by `COHSP_1:13` with unnecessary objects like `f`, `m`, `n`, `k` appearing in `HENMODEL:3`. Duplication is a serious issue, because it bloats MML, creates confusion in it, dissipates people's work, while often, like in this case, reusing existing code as much as possible results in more elegant and general formalizations (if the pre-existing code is already elegant and general enough). A major cause of this issue is the problematic browsing and mastering of such an extensive corpus like MML. Various attempts at delivering tools to assist Mizar authors in browsing it have been made ([Urb06b], [BU04] and [BR03]). Let us note that, in turn, `COHSP_1:13` itself is susceptible of what, in the writer's opinion, are improvements: indeed, in `FOMODEL0`, that same result, indeed stated in a slightly more general form

```
for Y being set st Y is c=directed holds
for X being finite Subset of union Y
ex y being set st y in Y & X c= y;
```

is proved by slicing it into six small and general propositions, for an amount of 66 lines of Mizar code versus the 68 lines of the original proof. Obviously the only purpose of this computation is to show that the two proofs are comparably long, what actually matters is the bunch of auxiliary results obtained 'for free':

```
Th60: for X, Y being set st union X c= Y holds X c= bool Y;

Th61: for X being set holds
A is_finer_than B & X is_finer_than Y implies
  A\/X is_finer_than B\/Y;

Th62: for A, B being set st A is_finer_than B holds
  A\/B is_finer_than B;

Th63: for A, B being set st
B is c=directed & A is_finer_than B holds
  A\/B is c=directed;

Th64: for X, Y being set holds
  INTERSECTION(X,Y) is_finer_than X,
```

also reverberating on other, even more general, Mizar articles. Indeed, `INTERSECTION` and `is_finer_than` are introduced in `SETFAM_1`:

```
definition
  let SFX,SFY be set;
  pred SFX is_finer_than SFY means
  for X being set st X in SFX ex Y being set st
    Y in SFY & X c= Y;
end;

definition
  let SFX,SFY be set;
  func INTERSECTION (SFX,SFY) means
  for Z being set holds
  (Z in it iff
    ex X,Y being set st X in SFX & Y in SFY & Z = X /\ Y);
  existence;
  uniqueness;
end;
```

This kind of trimming is here regarded as important for MML, for reasons previously discussed in similar cases in which the proof of a given fact led to a string of by-products of independent interest.

3.4 Numerically characterizing the formalization

We want to estimate formalization cost and de Bruijn factor ([Wie00; ASC10; Nau06]).
There are huge spaces of discretionality, which will be discussed below, in both calculations, so we will make some arbitrary choices, hoping they will result sensible and acceptable.

Two figures are to be estimated in order to trigger calculations: the amount of man hours devoted to formalization and a number measuring the size of a non-formal, human-targeted mathematical text carrying information grossly equivalent to the one formalized.

3.4.1 Estimating formalizing time

A significant amount of work regarded preliminary reformulation ([Cam09]) rather than Mizar formalization, as seen in chapter 1. This portion of work was carried on largely before Mizar formalization even started, however its results were revised 'dynamically' during the formalization as a result of the 'feedback' cited in section 2.6.5, and as confirmed by the differences noticeable between Mizar code and [Cam09]. Thus, formalization time assessment will be affected by some excess due to this auxiliary work subtracting time to effective coding, and to the fact that the workflow was rather irregular and interleaved with idle periods due to extraneous activities; this last issue is probably common to most formalization time estimations.

With the foregoing cautionary remarks, evolution of the codebase is as follow, using Mizar public repository on author's homepage as a development history record. The first Mizar file ever written by the author dates back to 24th January 2010, and, since then, formalization and Mizar learning efforts went on concurrently; the first codebase including Gödel's completeness theorem was successfully checked on 12th October 2010.

Löwenheim-Skolem theorem was first successfully compiled on 5th November 2010. As a conclusion, formalizing time can be estimated in 284 days.

3.4.2 Establishing a non-formal, equivalent mathematical source text

For the reasons exposed in section 3.4.1, choosing a denominator to compute de Bruijn factor is not so straightforward in this case. The nearest treatment would obviously be [Cam09], which, however, merely highlights the points in the proof which are novel and less trivial, and silently assumes a lot of prerequisites. Instead, the low starting point of this formalization demands we choose a more thorough treatment as a fairer reference, with an exposition starting from scratch (alphabets, strings, etc...) as this formalization does, and not omitting the tedious and 'trivial' details. Since [EFT84], being an undergraduate text book, arguably satisfies these requirements and was the original source of inspiration, it seems a good candidate. Specifically, we OCRed[4] its scans and selected the excerpt going from section II.1 ('Alphabets', page 10) through section VI.1 ('The Löwenheim-Skolem Theorem', ending on page 89), taking the resulting ASCII text as our non-formal source text. It is available on author's home page for reference. We have not removed the dispensable bits occurring in this source (exercises, historical notes, examples); first, they can be considered quantitatively negligible for our purposes, especially if one consider how arbitrary the whole matter is; secondarily, if one regards de Bruijn factor as a fundamental ratio between how much information is needed for a machine to accept statements and how much information is needed for a human to accept the same statements, rather than a totally empirical indicator to practically compare formalization verbosities, he could consider those bits as effectively useful for that human reader to accept (assimilate, he would say) those statements.

[4] Optical character recognition, usually abbreviated to OCR, is the mechanical or electronic translation of scanned images of handwritten, typewritten or printed text into machine-encoded text.

3.4.3 Results

The formalization cost is then calculated to be

$$\frac{\frac{284}{7}}{89 - 10 + 1} = 0.5 \text{ weeks per page}$$

The de Bruijn factor is shown below:

	informal (bytes)	formal (bytes)	de Bruijn factor	
uncompressed	132495	710144	5.4	apparent
gzipped	46839	153399	3.3	intrinsic

3.5 Formalization can bring insight

Various reasons supporting the endeavour of formalizing the body of known mathematics have been given in several expositions. After doing such an extensive formalization, we would like to explicitly state an often overlooked, though merely potential, one: formalizing a proof can and should increase the amount of information the proof itself brings with it, with respect to the same proof in its 'paper' version one has when starting mechanizing it.
To elaborate on such a vague assertion, let us give specific cases, annotated with references to the present formalization:

- One is strongly encouraged to variously simplify things to make them digestible by a machine. This is likely to lead to a finer discern about what notions are really needed for a result to hold or event to be stated. For example, we note that the notion of consistency was not needed until Henkin's theorem, 1.9.4.8.
- One is strongly encouraged to modularize and reuse. This can possibly bring to previously unknown, or at least not clearly stated, or maybe just obvious but useful in cutting down redundancies, relations between results. This is of particular relevance in case of community-developed, self-referencing repositories such as the MML. See the discussion on page 48.
- Combining the two points above, one could, for example, obtain more, smaller propositions with less/weaker hypotheses, with the possible side effect of an escalation of their total number; as an example take what done in section 3.3.
- As for other kinds of computation, a machine can help the human keeping track of a large amount of data, as could be a large number of hypotheses among which a minimal set is to be isolated to make a theorem hold; maybe this set of hypotheses has grown after some application of previous point. In our case, we had to filter out what derivation rules were needed corresponding to various lemmas, see section 2.6.5.

Of course, the 'final user' of a theorem is often little interested in this kind of internals; on the other hand, if a theorem is regarded as a particle of information, this collateral, supplementary information pursued in refining it can be deemed some value; which indeed happens when dealing with foundational issues, as in, e.g., reverse mathematics.

Appendix A

Proof of the Substitution Lemma

Proposition A.0.0.1. *Given an interpretation i, a literal v and a term t of the language S, and given a set X, it holds:*

$$\bar{i} \circ \overline{\frac{t}{v}\Phi_X}\Big|_{T_{S,n}} = \overline{\frac{\bar{i}(t)}{v}i}\Big|_{T_{S,n}} \tag{A.1}$$

for every $n \in \mathbb{N}$.

Proof. Let $U \neq \emptyset$ be the universe of i, and set $u := \bar{i}(t) \in U$, $I := \frac{t}{v}\Phi_X$. The proof is by induction on n. First, consider $t_0 \in T_{S,0}$, and show that $\bar{i}\left(\bar{I}(t_0)\right) = \overline{\frac{u}{v}i}(t_0)$ as follows. Set $v_0 := t_0(0) \in \#^{-1}[\{0\}]$ and proceed by cases.

Case $v_0 = v$ Then

$$\bar{i}\left(\bar{I}(t_0)\right) \overset{1.8.0.26}{=} \bar{i}\left((I(v))(0)\right) \overset{1.8.0.25}{=} \bar{i}\left(\{(0,t)\}(0)\right) = u$$
$$\overset{1.8.0.25}{=} \left(\frac{u}{v}i\left(\{(0,v)\}\right)\right)(0) \overset{1.8.0.26}{=} \overline{\frac{u}{v}i}\left(\{(0,v)\}\right) = \overline{\frac{u}{v}i}(t_0).$$

Case $v_0 \neq v$

$$\bar{i}\left(\bar{I}(t_0)\right) \overset{1.8.0.26}{=} \bar{i}\left((I(v_0))(0)\right) \overset{1.8.0.25}{=} \bar{i}\left((\Phi_X(v_0))(0)\right) \overset{1.8.0.24}{=} \bar{i}(t_0)$$
$$\overset{1.8.0.26}{=} (i(v_0))(0) \overset{1.8.0.25}{=} \left(\left(\frac{u}{v}i\right)(v_0)\right)(0) \overset{1.8.0.26}{=} \overline{\frac{u}{v}i}(t_0).$$

Now suppose (A.1) is verified for every $n \leq m$. Consider $t' \in T_{S,m+1}$. It will suffice to show

$$\bar{i}\left(\bar{I}(t')\right) = \overline{\frac{u}{v}i}(t'). \tag{A.2}$$

Set $s := t'(0)$.

Left hand side of (A.2) can be rewritten thus by 1.8.0.26:

$$\bar{i}\left((I(s))\left(\bar{I} \circ \overrightarrow{t'}\right)\right) = \bar{i}\left((\Phi_X(s))\left(\bar{I} \circ \overrightarrow{t'}\right)\right) \overset{1.8.0.24}{=} \bar{i}\left(\{(0,s)\} * \left(** \left(\bar{I} \circ \overrightarrow{t'}\right)\right)\right),$$

where the first step is justified by $v \neq s$. After setting $t'' := \{(0,s)\} * \left(** \left(\overline{I} \circ \overrightarrow{t'}\right)\right) \in T_S$, we notice that $\overrightarrow{t''} = \overline{I} \circ \overrightarrow{t'}$ by definition 1.8.0.14, so that left side of (A.2) becomes, recalling 1.8.0.26,

$$(i\,(s))\left(\overline{i} \circ \overrightarrow{t''}\right) = (i\,(s))\left(\overline{i} \circ \left(\overline{I} \circ \overrightarrow{t'}\right)\right). \tag{A.3}$$

We now perform calculations on right hand of (A.2) as well:

$$\overline{\frac{u}{v}i}\,(t') = \left(\frac{u}{v}i\,(s)\right)\left(\overline{\frac{u}{v}i} \circ \overrightarrow{t'}\right)$$
$$\overset{(A.1)}{=} \overline{\frac{u}{v}i}\,(t') = \left(\frac{u}{v}i\,(s)\right)\left(\overline{i} \circ \overline{I} \circ \overrightarrow{t'}\right) = (i\,(s))\left(\overline{i} \circ \overline{I} \circ \overrightarrow{t'}\right),$$

with last equality justified again by $v \neq s$. Comparing this with (A.3) yields the thesis. □

Proposition A.0.0.2. *Given an interpretation i, a literal v and a term t of the language S*

1. *For any formula ψ, $|\psi\,[v/t]| = 0$ if and only if $|\psi| = 0$.*

2. $\overline{i} \circ [v/t]\big|_{F_{S,0}} = \left.\overline{\frac{\overline{i}(t)}{v}i}\right|_{F_{S,0}}$.

Proof. First thesis descends immediately from 1.8.0.32. Consider $\psi_0 \in F_S$, $|\psi_0| = 0$. We have to show $\overline{i} \circ [v/t]\,(\psi_0) = \overline{\frac{\overline{i}(t)}{v}i}\,(\psi_0)$. Set $r := \psi_0\,(0)$ and go by cases.

$r \neq \equiv$
: Then

$$\left(\overline{i} \circ [v/t]\right)(\psi_0) = \overline{i}\,(\psi_0\,[v/t]) \overset{1.8.0.32,\ 1.8.0.26}{=} (i\,(r))\left(\overline{i} \circ \left(\overline{\frac{t}{v}\Phi_\emptyset}\right) \circ \overrightarrow{\psi_0}\right)$$
$$\overset{A.0.0.1}{=} (i\,(r))\left(\overline{\frac{\overline{i}\,(t)}{v}i} \circ \overrightarrow{\psi_0}\right) = \left(\overline{\frac{\overline{i}\,(t)}{v}i}\,(r)\right)\left(\overline{\frac{\overline{i}\,(t)}{v}i} \circ \overrightarrow{\psi_0}\right) \overset{1.8.0.26}{=} \overline{\frac{\overline{i}\,(t)}{v}i}\,(\psi_0),$$

where the second last step took into account that $v \neq r$ (this is because $\#\,(v) = 0$ while $\#\,r < 0$).

$r = \equiv$ This case is similar to the one above. It can be retrieved inside `FOMODEL3:8`.

□

Proposition A.0.0.3. $|\psi\,[v/t]| = |\psi|$.

Proof. It is an easy induction exploiting A.0.0.2 and 1.8.0.32. □

Lemma A.0.0.4. *Given $n \in \mathbb{N}$, a set $U \neq \emptyset$, a language S, a literal v and a term t of S:*

for every interpretation i of S having U as universe, it holds

$$\overline{i} \circ [v/t]\big|_{F_{S,n}} = \left.\overline{\frac{\overline{i}\,(t)}{v}i}\right|_{F_{S,n}}. \tag{A.4}$$

Proof. Set $f := [v/t]$ (see definition 1.8.0.32). By induction on n. The base case $n = 0$ is given by A.0.0.2. Assume (A.4) holds for any $n \leq m$, then consider $\psi \in F_{S,m+1}$ and an interpretation i of S having universe U. It suffices to show $\overline{i}\,(f\,(\psi)) = \overline{\frac{\overline{i}(t)}{v} i}\,(\psi)$. Set $s := \psi\,(0)$. We can assume $|\psi| > 0$, and proceed by cases.

Case 1): $s \neq \downarrow$.
Then $s = v_1 \in \#^{-1}\,[\{0\}]$, and $\psi = \{(0, v_1)\} * \varphi$ for some $\varphi \in F_{S,m}$. By 1.8.0.32, $f\,(\psi) = \{(0, v_2)\} * f\left(\frac{v_2}{v_1}\varphi\right)$, with

$$v_2 \notin \{v\} \cup \lfloor t, \varphi \rfloor\,. \tag{A.5}$$

Assume $\overline{i}\,(f\,(\psi)) = 1$. Then, by 1.8.0.27, consider $u_2 \in U$ such that

$$\begin{aligned} 1 = \overline{\frac{u_2}{v_2} i}\left(f\left(\frac{v_2}{v_1}\varphi\right)\right) &\overset{\text{A.0.0.3}}{=} \overline{\frac{\overline{i_2}\,(t)}{v}\frac{u_2}{v_2} i}\left(\frac{v_2}{v_1}\varphi\right) \\ &\overset{\text{(A.5)}}{=} \overline{\frac{u_2}{v_2}\frac{\overline{i_2}\,(t)}{v} i}\left(\frac{v_2}{v_1}\varphi\right) \overset{\text{1.9.4.4, (A.5)}}{=} \overline{\frac{u_2}{v_1}\frac{\overline{i_2}\,(t)}{v} i}\,(\varphi)\,, \end{aligned}$$

where we set $i_2 := \frac{u_2}{v_2} i$, and A.0.0.3 is invoked to trigger induction. Hence, by 1.8.0.27

$$1 = \overline{\frac{\overline{i_2}\,(t)}{v} i}\,(\{(0, v_1)\} * \varphi) = \overline{\frac{\overline{i}\,(t)}{v} i}\,(\psi)\,,$$

where last step is due to $v_2 \notin \operatorname{ran} t$. The proof of $\overline{\frac{\overline{i}(t)}{v} i}\,(\psi) = 1 \Longrightarrow \overline{i}\,(f\,(\psi)) = 1$ is very similar.

Case 2): $s = \downarrow$.
Then consider $\psi_1, \psi_2 \in F_{S,m}$ such that $\psi = \{(0, \downarrow)\} * \psi_1 * \psi_2$.

$$\begin{aligned} \overline{i}\,(f(\psi)) &\overset{\text{1.8.0.32}}{=} \overline{i}\,((0, \downarrow) * f\,(\psi_1) * f\,(\psi_2)) \overset{\text{1.8.0.27}}{=} N\,((\overline{i}\,(f\,(\psi_1)), \overline{i}\,(f\,(\psi_2)))) \\ &\overset{\text{A.0.0.3}}{=} N\left(\left(\overline{\frac{\overline{i}\,(t)}{v} i}\,(\psi_1), \overline{\frac{\overline{i}\,(t)}{v} i}\,(\psi_2)\right)\right) \overset{\text{1.8.0.27}}{=} \overline{\frac{\overline{i}\,(t)}{v} i}\,(\{(0, \downarrow)\} * \psi_1 * \psi_2)\,. \end{aligned}$$

Again, A.0.0.3 is needed to deploy induction, and N is a shorthand for the map $1^{2\times 2}_{\{(0,0)\}}$. □

Appendix B

Mizar functors used in the text

`f"X`	preimage of the set X through f	$f^{-1}[X]$
`X/\Y`	set-theoretical intersection	$X \cap Y$
`X\/Y`	set-theoretical union	$X \cup Y$
`X\Y`	set-theoretical difference	$X \backslash Y$
`X\+\Y`	symmetric difference	$(A \backslash B) \cup (B \backslash A)$
`[x,y]`	Kuratowski ordered pair	(x, y)
`[:X,Y:]`	cartesian product of sets	$X \times Y$
`NAT, INT`	natural numbers and integers	$\mathbb{N}, \mathbb{Z}$
`X*`	tuples on X	X^*
`n-tuples_on X`	tuples of n letters in X	X^n
`Seg n`		$\{1, \ldots, n\}$
`<*s*>`	the tuple made of the char s	$\{(0, s)\}$
`p^q`	concatenation of tuples p and q	$p * q$
`dom R, rng R`	domain, range of relation R	
`p/^n`	the tuple p with the first n chars removed	
`bool X`	the power set of X	2^X
`f.x`	the value of the function f in x	$f(x)$
`id X`	the identity function on X	$\bigcup_{x \in X} \{x\} \times \{x\}$
`f +* g`	the pasting of functions f, g	$f \lhd g$
`curry`	currying	$x \mapsto \lambda x.f(x, y)$
`f * g`	functional composition	$f \circ g$
`f.:X`	image of the set X through f	$f[X]$
`[x,y]‘1  [x,y]‘2`	projectors for Kuratowski pairs	$(x, y) \mapsto x$ $(x, y) \mapsto y$
`Funcs(X,Y)`	the set of functions from X to Y	Y^X
`PFuncs(X,Y)`	the set of partial functions from X to Y	$\bigcup_{x \subseteq X} Y^x$
`iter(f,n)`	n-th iteration of a function f	$f^{(n)}$
`R[*]`	transitive closure of R	
`X -> y`	the y-constant function on X	$X \to \{y\}$
`x .-> y`	function between two singletons	$\{(x, y)\}$
`chi(Y,X)`	characteristic function of $Y \subseteq X$	1_Y^X

Bibliography

[AGN09] A. Asperti, H. Geuvers, and R. Natarajan. "Social processes, program verification and all that". In: *Mathematical Structures in Computer Science* 19.05 (2009), pp. 877–896. ISSN: 1469-8072.

[ASC10] A. Asperti and C. Sacerdoti Coen. "Some Considerations on the Usability of Interactive Provers". In: *Intelligent Computer Mathematics: 10th International Conference, Aisc 2010, 17th Symposium, Calculemus 2010, and 9th International Conference, Mkm 2010, Paris, France, July 5-10, 2010. Proceedings*. 2010, p. 147. ISBN: 3642141277.

[Ban90] G. Bancerek. "A model of ZF set theory language". In: *Formalized Mathematics* 1.1 (1990), pp. 131–145. ISSN: 1426-2630.

[Ben06] Y. Benkler. *The wealth of networks: How social production transforms markets and freedom*. Yale Univ Pr, 2006. ISBN: 0300110561.

[BK05] P. Braselmann and P. Koepke. "Gödel's Completeness Theorem". In: *Formalized Mathematics* 13.1 (2005), pp. 49–53. ISSN: 1426-2630.

[Boy+94] R. Boyer et al. "The QED Manifesto". In: *Automated deduction, CADE-12*. Vol. 12. 1994, pp. 238–251. ISBN: 9783540581567.

[BR03] G. Bancerek and P. Rudnicki. "Information retrieval in MML". In: *Mathematical Knowledge Management: Second International Conference, MKM 2003, Bertinoro, Italy, February 16-18, 2003. Proceedings*. Springer. 2003, pp. 119–132.

[Bru70] N. de Bruijn. "The mathematical language AUTOMATH, its usage, and some of its extensions". In: *Proceedings of the Automatic Demonstration Symposium*. Springer. 1970, pp. 29–61. ISBN: 9780387049144.

[Bru72] N. G. de Bruijn. "Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem". In: *Nederl. Akad. Wetensch. Proc. Ser. A* ***75***=*Indag. Math.* 34 (1972), pp. 381–392.

[BU04] G. Bancerek and J. Urban. "Integrated semantic browsing of the Mizar Mathematical Library for authoring Mizar articles". In: *Mathematical Knowledge Management*. Springer. 2004, pp. 44–57. ISBN: 9783540230298.

[Cam09] M. Caminati. "Yet another proof of Gödel's completeness theorem for first-order classical logic". In: *Arxiv preprint arXiv:0910.2059* (2009).

[Cam10] M. Caminati. "Basic first-order model theory in Mizar". In: *Journal of Formalized Reasoning* 3.1 (2010), pp. 49–77. ISSN: 1972-5787.

[Cam11a] M. Caminati. “Definition of first order language with arbitrary alphabet”. In: *Formalized Mathematics* 19.3 (2011). ISSN: 1426-2630.

[Cam11b] M. Caminati. “First order languages: syntax, part two; semantics”. In: *Formalized Mathematics* 19.3 (2011). ISSN: 1426-2630.

[Cam11c] M. Caminati. “Free interpretation, quotient interpretation and substitution of a letter with a term for first order languages”. In: *Formalized Mathematics* 19.3 (2011). ISSN: 1426-2630.

[Cam11d] M. Caminati. “Preliminaries to Classical First-order Model Theory”. In: *Formalized Mathematics* 19.3 (2011). ISSN: 1426-2630.

[Cam11e] M. Caminati. “Sequent calculus, derivability, provability. Gödel’s completeness theorem”. In: *Formalized Mathematics* 19.3 (2011). ISSN: 1426-2630.

[CG07] P. Cairns and J. Gow. “Integrating searching and authoring in Mizar”. In: *Journal of Automated Reasoning* 39.2 (2007), pp. 141–160. ISSN: 0168-7433.

[CH07] I. Chiswell and W. Hodges. *Mathematical logic*. Vol. 3. Oxford Texts in Logic. Oxford: Oxford University Press, 2007, pp. viii+250. ISBN: 978-0-19-921562-1.

[Che80] B. Chellas. *Modal logic: an introduction*. Cambridge Univ Press, 1980. ISBN: 0521224764.

[CMU08] P. Chapman, J. McKinna, and C. Urban. “Mechanising a Proof of Craig’s Interpolation Theorem for Intuitionistic Logic in Nominal Isabelle”. In: Springer, 2008, pp. 38–52. ISBN: 9783540851097.

[CR11] M. Caminati and G. Rosolini. “Custom automations in Mizar”. In: *Journal of Automated Reasoning* (2011). Invited article for the special issue ‘Formal Mathematics for Mathematicians: Developing Large Repositories of Advanced Mathematics’. Under review as of November 15th, 2011.

[Cra+10] M. Cramer et al. “The naproche project controlled natural language proof checking of mathematical texts”. In: Springer, 2010, pp. 170–186. ISBN: 9783642144172.

[DG10] J. Dawson and R. Goré. “Generic methods for formalising sequent calculi applied to provability logic”. In: *Logic for Programming, Artificial Intelligence, and Reasoning*. Springer. 2010, pp. 263–277. ISBN: 9783642162411.

[EFT84] H. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic*. Second. Undergraduate Texts in Mathematics. Springer-Verlag, 1984. ISBN: 0387908951.

[GKN10] A. Grabowski, A. Korniłowicz, and A. Naumowicz. “Mizar in a Nutshell”. In: *Journal of Formalized Reasoning* 3.2 (2010), pp. 153–245. ISSN: 1972-5787.

[Gon08] G. Gonthier. “Formal proof—the four-color theorem”. In: *Notices Amer. Math. Soc.* 55.11 (2008), pp. 1382–1393. ISSN: 0002-9920.

[Gor09] R. Goré. "Machine checking proof theory: an application of logic to logic". In: *Logic and its applications*. Vol. 5378. Lecture Notes in Comput. Sci. Berlin: Springer, 2009, pp. 23–35. DOI: 10.1007/978-3-540-92701-3_2. URL: http://dx.doi.org/10.1007/978-3-540-92701-3_2.

[Har98] J. Harrison. "Formalizing basic first order model theory". In: *Theorem Proving in Higher Order Logics*. Springer, 1998, pp. 153–170. ISBN: 9783540649878.

[Hed04] S. C. Hedman. *A first course in logic*. Vol. 1. Oxford Texts in Logic. An introduction to model theory, proof theory, computability, and complexity. Oxford: Oxford University Press, 2004, pp. xx+431. ISBN: 0-19-852981-3.

[Her96] I. N. Herstein. *Abstract algebra*. Third. With a preface by Barbara Cortzen and David J. Winter. Upper Saddle River, NJ: Prentice Hall Inc., 1996, pp. xviii+249. ISBN: 0-13-374562-7.

[HR10] M. Humayoun and C. Raffalli. "Mathnat-mathematical text in a controlled natural language". In: *Special issue: Natural Language Processing and its Applications, Journal on Research in Computing Science* 46 (2010), pp. 293–310. ISSN: 1870-4069.

[KMK92] J. Kotowicz, B. Madras, and M. Korolkiewicz. "Basic notation of universal algebra". In: *Journal of Formalized Mathematics* 4 (1992). ISSN: 1426-2630.

[Knu97] D. E. Knuth. *Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley Professional, 1997. ISBN: 0201896834.

[Kor09] A. Korniłowicz. "How to Define Terms in Mizar Effectively". In: *Studies in Logic, Grammar and Rhetoric* 18.31 (2009), pp. 67–77. ISSN: 0860-150X.

[KP81] B. W. Kernighan and P. J. Plauger. *Software tools in pascal*. Addison-Wesley, 1981. ISBN: 0201103427.

[Lot02] M. Lothaire. *Algebraic combinatorics on words*. Cambridge Univ Pr, 2002. ISBN: 0521812208.

[MVW98] A. Mikhajlova and J. Von Wright. "Proving isomorphism of first-order logic proof systems in HOL". In: *Theorem Proving in Higher Order Logics*. Springer, 1998, pp. 295–314. ISBN: 9783540649878.

[NA09] B. Neveln and B. Alps. "ProofCheck: Writing and checking complete proofs in LaTeX". In: *TUGboat* 30.2 (2009), pp. 191–195. ISSN: 0896-3207.

[Nau06] A. Naumowicz. "An example of formalizing recent mathematical results in Mizar". In: *Journal of Applied Logic* 4.4 (2006), pp. 396–413. ISSN: 1570-8683.

[Nau07] A. Naumowicz. "Evaluating Prospective Built-in Elements of Computer Algebra in Mizar". In: *Studies in Logic, Grammar and Rhetoric* 10.23 (2007), pp. 191–200. ISSN: 0860-150X.

[NB04] A. Naumowicz and C. Byliński. "Improving Mizar texts with properties and requirements". In: *Mathematical Knowledge Management*. Springer. 2004, pp. 290–301. ISBN: 9783540230298.

[Nip03] T. Nipkow. "Structured proofs in Isar/HOL". In: *Types for Proofs and Programs* (2003), pp. 619–620. URL: http://dx.doi.org/10.1007/3-540-39185-1.

[NS56] A. Newell and H. Simon. "The logic theory machine–A complex information processing system". In: *Information Theory, IRE Transactions on* 2.3 (1956), pp. 61–79.

[RT90] P. Rudnicki and A. Trybulec. "A first order language". In: *Formalized Mathematics* 1.2 (1990), pp. 303–311. ISSN: 1426-2630.

[RT99] P. Rudnicki and A. Trybulec. "On equivalents of well-foundedness". In: *Journal of Automated Reasoning* 23.3 (1999), pp. 197–234. ISSN: 0168-7433.

[RU11] P. Rudnicki and J. Urban. "Escape to ATP for Mizar". In: *First Workshop on Proof eXchange for Theorem Proving*. 2011. URL: http://pxtp2011.loria.fr/.

[Sal94] P. Salus. *A quarter century of UNIX*. Addison-Wesley, 1994. ISBN: 0201547775.

[Sch+12] P. Schodl et al. "Towards a Self-reflective, Context-aware Semantic Representation of Mathematical Specifications". In: *Algebraic Modeling Systems - Modeling and Solving Real World Optimization Problems. Springer, to appear*. 2012.

[Smu95] R. M. Smullyan. *First-order logic*. Corrected reprint of the 1968 original. New York: Dover Publications Inc., 1995, pp. xii+158. ISBN: 0-486-68370-2.

[Tar28] A. Tarski. "On some fundamental concepts of metamathematics". In: *[Tar56]*. 1928, pp. 30–37.

[Tar30] A. Tarski. "Fundamental concepts of the methodology of the deductive sciences". In: *[Tar56]*. 1930, pp. 60–109.

[Tar35] A. Tarski. "Foundations of the calculus of systems". In: *[Tar56]*. 1935, pp. 342–383.

[Tar56] A. Tarski. *Logic, semantics, metamathematics: papers from 1923 to 1938*. Oxford At The Clarendon Press, 1956. URL: http://www.questia.com/PM.qst?a=o&d=91287094.

[Tar65] A. Tarski. "A simplified formalization of predicate logic with identity". In: *Arch. Math. Logik Grundlagenforsch* 7 (1965), 61–79 (1965). ISSN: 0003-9268.

[Try] A. Trybulec. "Revising UPROOTS: Global Choine [sic]". Message to the Mizar mailing list, sent on 2008/10/13. URL: http://mizar.uwb.edu.pl/forum/archive/0810/msg00004.html.

[TS96] A. S. Troelstra and H. Schwichtenberg. *Basic proof theory*. Vol. 43. Cambridge Tracts in Theoretical Computer Science. Cambridge: Cambridge University Press, 1996, pp. xii+343. ISBN: 0-521-57223-1.

[Urb06a] J. Urban. "MizarMode–an integrated proof assistance tool for the Mizar way of formalizing mathematics". In: *Journal of Applied Logic* 4.4 (2006), pp. 414–427. ISSN: 1570-8683.

[Urb06b] J. Urban. "MoMM-fast interreduction and retrieval in large libraries of formalized mathematics". In: *International Journal on Artificial Intelligence Tools* 15.1 (2006), p. 109. ISSN: 0218-2130.

[Wie00] F. Wiedijk. "The De Bruijn Factor". In: *preprint* (2000). URL: http://www.cs.ru.nl/~freek/factor/factor.pdf.

[Wie06] F. Wiedijk. "Writing a Mizar article in nine easy steps". 2006. URL: http://www.cs.ru.nl/~freek/mizar/mizman.pdf.

[Wie07a] F. Wiedijk. "Mizar's soft type system". In: *Proceedings of the 20th international conference on Theorem proving in higher order logics*. Springer-Verlag. 2007, pp. 383–399. ISBN: 3540745904.

[Wie07b] F. Wiedijk. "The QED manifesto revisited". In: *Studies in Logic, Grammar and Rhetoric* 10.23 (2007), pp. 121–133. ISSN: 0860- 150X.

[Wie09] F. Wiedijk. "Formalizing Arrow's theorem". In: *Sādhanā* 34.1 (2009), pp. 193–220. ISSN: 0256-2499. DOI: 10.1007/s12046-009-0005-1. URL: http://dx.doi.org/10.1007/s12046-009-0005-1.