Formalizing the Gödel Completeness Theorem

Julian Johannes Schlöder

Geboren am 20. Juni 1989 in Münster

8. Juli 2011

Bachelorarbeit Mathematik

Betreuer: Prof. Dr. Peter Koepke

Mathematisches Institut

Mathematisch-Naturwissenschaftliche Fakultät der

Rheinischen Friedrich-Wilhelms-Universität Bonn

Contents

1 Introduction

In 2005 Patrick Braselmann and Peter Koepke published a version of the Gödel Completeness Theorem[8], which can be automatically proof checked by the Mizar System. However, they proved only a special case of the theorem:

```
theorem :: GOEDELCP:38
for X being Subset of CQC-WFF
for p being Element of CQC-WFF st
still_not-bound_in X is finite & X |= p holds
X |- p
```

So the theorem only holds for *countable* languages and sets of well-formed formulae with a *finite number* of unbound variables. Since the used "CQC" language does not discriminate between variables and constants and each constant is counted as an unbound variable, the theorem actually applies only to a quite small number of sets of well-formed formulae. So it really is a severely restricted version of Gödels Theorem. Therefore I decided to lift these restrictions, i.e. prove the Completeness Theorem for *arbitrarily large* languages and arbitrary sets of well-formed formulae:

```
theorem :: GOEDCPOC:15
for Al being FO-alphabet
for PSI being Subset of CFO-WFF(Al)
for p being Element of CFO-WFF(Al) holds
PSI |= p implies PSI |- p;
```

Note the addition of the `FO-alphabet` which denotes the (arbitrarily large) set of valid symbols of which the well-formed formulae may be formed. Since these are now dependent on an alphabet, `Al` is added as an argument to `CFO-WFF(Al)`.

Braselmann and Koepke based their work on a number of existing articles in the Mizar Mathematical Library (MML) ([1, 9, 10, 12, 13, 15, 16]) which used a fixed, countable language[15]. So I needed to modify these articles to be able to talk about uncountably large languages. Also, since the language was fixed, it was not possible to extend it. But this technique is needed to prove the theorem for sets with an infinite number of unbound variables. So the limited version as quoted above really is the farthest one could get without diving deep into the existing knowledge base. In the end, I needed to modify 14 existing articles, including the 7 articles by Braselmann and Koepke([2, 3, 4, 5, 6, 7, 8]). To avoid conflict with the original articles in the MML, I renamed the language and all included definitions by replacing "QC" with "FO" (for "first order"). Other than that and my redefinition of the basic set of symbols I tried to stay as true as possible to the original framework. This led to some unnecessary baggage, which I guess is rooted in historical developments (most of the articles are over 20 years old and seem to have been quite heavily modified over time). I will discuss this in detail below.

I will furthermore discuss the major steps I formalized to prove the universal version of Gödels Completeness Theorem in Mizar. I followed the proof discussed in “Einführung in die Mathematische Logik” by Ebbinghaus et al.[14]. The results were split into two articles: FO_TRANS, proving some important preliminary results, and GOEDCPOC, discussing all the actual steps in the proof of the Completeness Theorem. I will now give the relevant proofs and definitions in an abstract form, representing their respective formalizations, so that the reader may have some kind of reference while reading the Mizar articles themselves. For this reason most definitions and variables in this abstract discussion have the same, or at least similar, names as in the Mizar articles. Furthermore I will describe where the formalization needed to deviate (sometimes in non-obvious ways) from the abstract proof, due to high-level abstract techniques.

2 The First Order Language

As mentioned before, I introduced the notion of *alphabets* into the language framework:

Definition 2.1 (FO_LANG1:def 1, def 2)**.** A **(First Order/FO-) alphabet** is a non empty set A such that $A = \mathbb{N} \times X$ for some set X with $\mathbb{N} \subseteq X$.

We call $\mathbb{N}$ the **indices** and $X =: \Sigma(A)$ the **(First Order / FO-) symbols** of A. Similarily, for every **letter** $x \in A$ (i.e. x being a tuple of $\mathbb{N}, X$) the first component is called the **index** and the second is called the **symbol** of x.

Trivially, an alphabet is at least countably infinite (since it is always a superset of $\mathbb{N}^2$) and may be arbitrarily large. We now differentiate distinct types of letters (e.g. junctors, variables, predicate symbols) purely by their index. This allows us to use the same symbol in different syntactical functions, which is actually quite close to common practice in natural-language mathematics, where, for instance, one may canonically use π as a constant (the number) or a function symbol (the prime-counting function). It is also worth mentioning that the definitions of “letter” and “index” are never explicitly given in the Mizar articles. This is because the built in types `Element of NAT` and `Element of A` (for a `FO-alphabet A`) serve their purpose just as well and the indices of A are always `NAT` anyway.

The formalization as a cartesian product is adapted from the previous work in first order languages, where the language is formalized as $\mathbb{N} \times \mathbb{N}$[15]. As mentioned, I wanted to deviate as little as possible from the original framework, so the extension of one component to allow uncountable languages seemed sensible. However, this method of formalization appears to be quite good and I probably would have used it even if I had decided to modify the existing articles further.

Convention 2.2. From now on A denotes an Alphabet.

The following definitions are all slightly modified versions of their respective equivalent ones in QC_LANG1[15].

Definition 2.3 (FO_LANG1:def 14)**.** The letter $(0, 0)$ of A is considered as $\top$ (the **truth symbol**) (this corresponds to [15, def 12])

$\top$ is styled as `VERUM(A)` in Mizar. For purely technical reasons, the Alphabet needed to be made an argument of `VERUM` in Mizar.

Definition 2.4 (FO_LANG1:def 15)**.** The letter $(1, 0)$ of A is considered as $\neg$ (the **negation**) (this corresponds to [15, def 13])

$\neg$ is styled as `'not'` (note the apostrophes) in Mizar.

Definition 2.5 (FO_LANG1:def 16)**.** The letter $(2, 0)$ of A is considered as $\wedge$ (the **conjunction**) (this corresponds to [15, def 14])

$\wedge$ is styled as `'&'` in Mizar.

Definition 2.6 (FO_LANG1:def 17)**.** The letter $(3, 0)$ of A is considered as $\forall$ (the **universal quantifier**) (this corresponds to [15, def 15])

$\forall$ is styled as `All` in Mizar.

Definition 2.7 (FO_LANG1:def 4)**.** A letter x of A is considered a **literal** iff the index of x equals 4. $\Lambda(A)$ is the set of all literals in A. (this corresponds to [15, def 2])

Literals serve thrice as symbols for bound and free variables as well as constant symbols. For purely historical reasons, literals are designated as `bound_FO-variables` in Mizar, since the most basic articles[1, 9, 15] reserve different sets of symbols for free, bound and fixed (presumably constant) variable symbols respectively, but this concept was abandoned later[10] and henceforth the set reserved for bound variables is used in all three roles. It is reasonable to assume that discrimination of three different types of literals proved to be too tedious to work with; especially with the prospect of variable substitution and structural induction in more complex proofs. In any way, since variable and constant symbols follow the exact same syntax, we may just consider them to be the same at this point.

Note that now we may use uncountably many variable symbols if we consider an uncountable Alphabet. This is normally not done in classical First Order logic (in particular, Ebbinghaus et al. enumerate the variable symbols[14, p. 12]). One might set as an convention that a literal with index in $\mathbb{N}$ is considered to be variable symbol and otherwise to be a constant symbol, but this would mean that one would be able to quantify over a literal which is considered a constant symbol. No notion like that is implemented or enforced anywhere anyway. Again, since currently only the syntactical properties are discussed and we can have uncountably many constant symbols in any case, this may be neglected for the time being.

Definition 2.8 (FO_LANG1:def 7, def 8, def 9)**.** A letter P of A is considered a **predicate symbol** iff the index i of P is greater than or equal 7. The **arity** of P equals $i - 7$. $\Pi(A)$ is the set of all predicate symbols in A, $\Pi(A, k)$ is the set of all predicate symbols in A with arity k, where $k \in \mathbb{N}$. (this corresponds to [15, def 5, def 6, def 7])

In this special case, the index serves an additional purpose by holding the information about the arity of a given predicate symbol.

There is no identity-symbol or function symbol in the language; it is intended that these shall be simulated by predicate symbols. Also there are some "unused" or syntactically invalid letters in the alphabet: (m, n) for $n \in \Sigma(A)$, $n \neq 0$, $m = 0, 1, 2, 3$, since the indices $0, 1, 2, 3$ are reserved for special syntactic letters and $(5, k), (6, k)$ for $k \in \Sigma(A)$ since the indices 5 and 6 were used for fixed and free variable symbols[15, def 3, def 4] historically. This has no further ramifications; one only has to look out for it when considering arbitrary letters, but this is usually done in the context of a well-formed formula, where only valid letters can appear. Now we can consider the usual definition of well-formed formulae over A as finite sequences of letters using polish notation.

Definition 2.9 (FO_LANG1:def 10, def 11, CFO_LANG:def 2)**.** The **well-formed formulae** $\Omega(A)$ are the smallest set such that

i. $\Omega(A) \subseteq A^*$,

ii. $(\top) \in \Omega(A)$,

iii. for any $k \in \mathbb{N}$, any predicate symbol P of A with arity k and any finite sequence l of literals of A with length k, $(P)^\frown l \in \Omega(A)$,

iv. for every $p \in \Omega(A)$, $(\neg)^\frown p \in \Omega(A)$,

v. for every $p, q \in \Omega(A)$, $(\wedge)^\frown p^\frown q \in \Omega(A)$,

vi. for every $p \in \Omega(A)$ and every x being a literal of A is $(\forall)^\frown(x)^\frown p \in \Omega(A)$.

(this corresponds to [15, def 8, def 9], [10, def 2])

The well-formed formulae of A are styled as `CFO-WFF(A)` in Mizar. Now all usual basic properties of such a formal language are formulated and proven in [1, 9, 10, 12, 13, 15, 16], most notably the structural induction[10, sch 1] is introduced as a Second Order predicate. At this point, the work of modifying the articles was mostly confined to introducing an arbitrary alphabet A at the beginning of an article and then adding “of A” or “(A)” as parameters to all alphabet-dependent types and functions.

3 Models and Interpretations

There were no changes made to the existing notions of satisfiability, models and interpretations in the MML[16] other than adding the notion of arbitrary alphabets instead of one fixed language, but since we do not discriminate between variables and constants in a syntactic way, their semantics deserve a closer look.

Definition 3.1 (FO_VALUA:def 1)**.** Let D be a non empty set. A **valuation** of A in D is any function $v : \Lambda(A) \to D$. (this corresponds to [16, def 1])

Definition 3.2 (FO_VALUA:def 5)**.** Let D be a non empty set. An **interpretation** of A in D is any function $I : \Pi(A) \to \{r : r \text{ is relation on } D\}$ such that for every $P \in \Pi(A)$, $I(P) = \emptyset$ or P and $I(P)$ have the same arity. (this corresponds to [16, def 5])

The modeling relation is then defined recursively over $\Omega(A)$ as usual. Now for a closer look at variables and constants: Classically variables are valued in the *valuation* whereas constants are defined in the *interpretation*. However, these two concepts always appear together and complement each other. The definition of constants may be moved from the interpretation to the valuation without loss of general logic soundness: If one can find a model of a given formula, where constants are defined in the interpretation, one may evaluate the constants in the valuation instead (i.e. reconsider them to be variables) and it is still a model, and vice versa. So the semantics (in relation to models) of variables and constants, while not being necessarily identical, is equivalent.

Also, it seems unproblematic to have a possibly uncountable number of variabels at one's disposal. A single formula contains only finitely many variables anyway and it is semantically irrelevant how they are named. To obtain a difference from the uncountableness, we need to take a uncountably large set of formulae whose variables would “use up” a countable number. For instance the set $\{\neg x = y : x, y \text{ being variables such that } x \neq y\}$ would only be satisfiable by uncountable models if there were uncountably many variables, yet be satisfiable by countable models if there were countable many variables. But this effect may be simulated by adding a equally large uncountable number of constant symbols to the language with countable many variable symbols and considering the set $\{\neg x = y : x, y \text{ being constant symbols such that } x \neq y\}$. By analogy one may recreate every effect that results from uncountably many variable symbols by adding equally many constant symbols to a language with countably many variable symbols. As seen in the following section, we may extend and reduce languages arbitrarily (as long as the formulae we talk about are still well-formed in a reduced language) without regard to satisfiabilty or consistency.

4 Extended and Reduced Languages

Since it plays a major role in this section, I will briefly explain how *typing* works in Mizar. As in most first order languages, one can define *Functions* (called *functors*) and *predicates* in Mizar; it is important to discern *functors*, defined in the language, from *functions*, constructed *inside* the universe of the language. Therefore the former will only be called "functors". Both functors and predicates are *typed*: In Mizar there are *modes* (base types) and *attributes* (adjectives, restrictions of a mode) which may be assigned to any object, e.g. `set` is a mode and `empty` is an attribute. Every object has an arbitrary number of attributes and **exactly one** mode. The most basic mode is `set`, which means that every other mode is derived from (i.e. is a restriction of) `set`. Any combination of a number of attributes and a mode is called a *type*, e.g. `Function`, `one-to-one Function` and `countable infinite Subset of REAL` are types. Theoretically attributes and modes are interchangeable; for example `Function` is identical to `Function-like set` (actually, this is the very definition of `Function` in Mizar). In essence, types may be considered to be classes, e.g. `set` equals V. Types are identical if they (as classes) contain exactly the same elements. One may assign additional arguments to modes and attributes, e.g. `Function of A,B` denotes the class of all Functions with domain A and codomain B and `-extending`, as used below, is an attribute which requires an alphabet as its argument. Now, for every functor and predicate, one may specify the domains for all arguments and (in case of a functor) the codomain by naming their types. The intricacies of this concept are best seen by considering an example:

```
definition
  let Al be FO-alphabet, p be Element of CFO-WFF(Al),
   x be bound_FO-variable of Al;
  func Example_of(p,x) -> bound_FO-variable of FCEx(Al)
  equals :: GOEDCPOC:def 4
  [4,[the free_Symbol of Al,[x,p]]];
end;
```

`Example_of` is a *functor*; it has the domains (its domain is the cartesian product of the following classes) `Element of CFO-WFF(Al)` and `bound_FO-variable of Al` (which are both examples of modes with arguments, by the way) and its domain is `bound_FO-variable of FCEx(Al)`, the class of all literals in `FCEx(Al)` (`FCEx` itself is a functor with domain and codomain being

`FO-alphabet`). So, for some `p`, which is no well-formed formula of `Al`, and/or `x`, which is no literal of `Al`, the expression `Example_of(p,x)` is invalid, i.e. it would be no well-formed term in Mizar and it would give a `Unknown Functor` error. Also if `p` is a well-formed formula in `Al` and `x` is a literal of *another* alphabet `Al2` the term `Example_of(p,x)` is invalid, too, because the domain was defined in relation to *one common* alphabet. The next example is a crucial effect and a major cause of problems in the following discussion: If `p` is a well-formed formula of `Al` and `x` is a literal of `Al`, but the assigned type of `x` is something else, e.g. it was introduced as `let x be set such that x is bound_FO-variable of Al`, the term `Example_of(p,x)` is *still* invalid, since Mizar is not able to change the mode of any object by itself.

Therefore, if one would talk about Mizar as a programming language, one would say Mizar is *strongly typed*, i.e. types are not automatically converted as needed. This is not surprising, since Mizar is based on (and developed with) Pascal, which has a restricted (strong and static) typing system itself.

Definition 4.1 (FO_TRANS:def 1)**.** An Alphabet A_2 such that $A \subseteq A_2$ is an **extension** of A; A_2 is A**-extending** and A is a **subalphabet** of A_2.

Now Mizar presents us with an interesting problem. Take an A-extending Alphabet A_2 and $p, q \in \Omega(A)$. Then obviously $p, q \in \Omega(A_2)$, so consider $p_2, q_2 \in \Omega(A_2)$ such that $p_2 \equiv p$ and $q_2 \equiv q$ $(*)$, i.e. p_2, p and q_2, q are identical when considered as sets. One might assume that we now can trivially infer $p \wedge q \equiv p_2 \wedge q_2$. But this is not the case in Mizar. The problem lies in the definition of the junctors. In Mizar, they are defined as *functors*, i.e. $\wedge$ is the functor $\wedge : \Omega(A)^2 \to \Omega(A), p \wedge q := \wedge pq = (2,0)^\frown p^\frown q$. Therefore, due to typing, the conjunction on A, $\wedge_A$, and the conjunction on A_2, $\wedge_{A_2}$, are *entirely different objects*. As explained above, $\wedge_A$ and $\wedge_{A_2}$ may be considered classes, and since they are not *identical* ($\wedge_{A_2}$ is larger), Mizar considers them to be *different*, and therefore cannot directly infer their equality on p, q, p_2 and q_2. This is best understood by looking at the formal code. Mizar can directly prove the following statement:

```
for A,A2 being FO-alphabet, p,q being Element of CFO-WFF(A),
p2,q2 being Element of CFO-WFF(A2) st A=A2 & p=p2 & q=q2
holds p '&' q = p2 '&' q2;
```

Because here the `'&'`s, i.e. $\wedge_A$ and $\wedge_{A_2}$, represent the exact same class. But Mizar **cannot** prove this slightly different statement (that is, without the additional concepts constructed below):

```
for A,A2 being FO-alphabet, p,q being Element of CFO-WFF(A),
p2,q2 being Element of CFO-WFF(A2) st A c= A2 & p=p2 & q=q2
holds p '&' q = p2 '&' q2;
```

Since Mizar bears close resemblance to common strongly typed programming languages, I adapted a concept from object-oriented programming designed for this kind of problem: *Typecasting.*

Definition 4.2 (FO_TRANS:def 3, def 4, def 5, def 6)**.** Let A_2 be an A-extending alphabet. Let $p \in \Omega(A), x \in \Lambda(A), P \in \Pi(A), l \in \Lambda(A)^*$. Then define their respective A_2**-Casts**:

i. $(A_2) : \Omega(A) \to \Omega(A_2), (A_2)(p) := p.$

ii. $(A_2) : \Lambda(A) \to \Lambda(A_2), (A_2)(x) := x.$

iii. $(A_2) : \Pi(A) \to \Pi(A_2), (A_2)(P) := P.$

iv. $(A_2) : (\Lambda(A))^* \to (\Lambda(A_2))^*, (A_2)(l) := l.$

A typecast to A_2 is styled as `A2-Cast()` in Mizar. Then the following theorem is easily formalized.

Theorem 4.3 (Typecasting Lemma, FO_TRANS:8)**.** *Let A_2 be an A-extending alphabet. Let $k \in \mathbb{N}, p, q \in \Omega(A), x \in \Lambda(A), P \in \Pi(A, k), l \in (\Lambda(A))^k$. Then:*

i. $(A_2)(\top_A) = \top_{A_2}.$

ii. $(A_2)(P^\frown l) = (A_2)(P)^\frown (A_2)(l).$

iii. $(A_2)(\neg_A p_A) = \neg_{A_2}(A_2)(p_A).$

iv. $(A_2)(p \wedge_A q) = (A_2)(p) \wedge_{A_2} (A_2)(q).$

v. $(A_2)(\forall_A x p) = \forall_{A_2}(A_2)(x)(A_2)(p).$

Proof. The proof is conducted in all five cases by taking the argument of the typecast, considering them as their representation as "typeless" finite sequences of sets, then reassigning the respective types of the according subsequences in relation to A_2. □

Now a statement like the aforementioned $p \wedge q \equiv p_2 \wedge p_2$ may be directly inferred as follows:

$$\begin{aligned} p \wedge q &\equiv (A_2)(p \wedge q) \text{ by Definition 4.2} \\ &\equiv (A_2)(p) \wedge (A_2)(q) \text{ by Theorem 4.3} \\ &\equiv p_2 \wedge q_2. \end{aligned}$$

The last step is now valid since here the $\wedge$s in $p_2 \wedge q_2$ and $(A_2)(p) \wedge (A_2)(q)$ are the exact same function $\wedge_{A_2}$ and by $(*)$ and Definition 4.2 the arguments are the same, while equal functors on equal arguments yield equal results. The Typecasting Lemma, while trivial in natural-language mathematics, has far-reaching applications in the FO_TRANS and GOEDCPOC articles; it is mainly needed for structural inductions and retyping of complex formulae across multiple alphabets.

The main goal of the FO_TRANS article was the transfer of consistency and satisfiability along different alphabets. We will need to apply the countable Gödel Completeness Theorem, formalized by Braselmann and Koepke[8, Theorem 38]. Therefore the following auxiliary theorems are needed:

Theorem 4.4 (FO_TRANS:20)**.** *For $p \in \Omega(A)$, there is a countable alphabet A_1 such that A is A_1-extending and $p \in \Omega(A_1)$.*

Proof. Take the set $\mathrm{L}(p)$ of all letters in p. Consider the alphabet $A_1 = \mathbb{N} \times (\mathbb{N} \cup \mathrm{L}(p))$. Since p only contains finitely many letters, A_1 is countable, and because $\mathbb{N}^2 \subseteq A$ and $L(p) \subseteq \Sigma(A)$, A is A_1-extending. Obviously $p \in \Omega(A_1)$. $\square$

Formally (i.e. in Mizar) this can be proven via structural induction over $\Omega(A)$.

Corollary 4.5 (FO_TRANS:21)**.** *For finite $\Phi \subseteq \Omega(A)$, there is a countable alphabet A_1 such that A is A_1-extending and $\Phi \subseteq \Omega(A_1)$.*

Proof. By Theorem 4.4, for every $p \in \Phi$ there is a countable alphabet A_p with $A_p \subseteq A$ and $p \in \Omega(A_p)$. Then set $A_1 = \bigcup_{p\in\Phi} A_p$. Since every A_p is countable and Φ is finite, A_1 is countable. Obviously A_1 is an alphabet, $A_1 \subseteq A$ and $\Phi \subseteq \Omega(A_1)$. $\square$

This proof is best formalized by finite induction over the cardinality of Φ, starting with $\emptyset$ and letting A_1 grow incrementally.

Theorem 4.6 (FO_TRANS:22)**.** *For finite $\Phi \subseteq \Omega(A)$, the set of all unbound variables in Φ is finite.*

Proof. Trivial, since Φ is finite and every $p \in \Phi$ can only contain finitely many letters. $\square$

Now we may consider the four eponymous theorems of the FO_TRANS article:

Theorem 4.7 (Downward Semantic Language Transfer, slightly weaker than FO_TRANS:10)**.** *Let A_2 be an A-extending alphabet, $\Phi \subseteq \Omega(A)$ such that Φ is satisfiable in A_2. Then Φ is satisfiable in A.*

Theorem 4.8 (Upward Semantic Language Transfer, FO_TRANS:23)**.** *Let A_2 be an A-extending alphabet, $\Phi \subseteq \Omega(A)$ such that Φ is satisfiable in A. Then Φ is satisfiable in A_2.*

Theorem 4.9 (Downward Syntactic Language Transfer, FO_TRANS:19)**.** *Let A_2 be an A-extending alphabet, $\Phi \subseteq \Omega(A)$ such that Φ is consistent in A_2. Then Φ is consistent in A.*

Theorem 4.10 (Upward Syntactic Language Transfer, FO_TRANS:25)**.** *Let A_2 be an A-extending alphabet, $\Phi \subseteq \Omega(A)$ such that Φ is consistent in A. Then Φ is consistent in A_2.*

The proofs of Theorem 4.7, Theorem 4.8 and Theorem 4.9 are straightforward:

Proof (4.7). Let D_2 be a non empty set, I_2 an interpretation of A_2 in D_2 and v_2 a valuation of A_2 in D_2 such that J_2, v_2 satisfies Φ. Set $D := D_2$, $I := I_2 \restriction \Pi(A)$ and $v := v_2 \restriction \Lambda(A)$. Then I, v satisfies Φ. $\square$

Proof (4.8). Let D be a non empty set, I an interpretation of A in D and v a valuation of A in D such that J, v satisfies Φ. Take any $d \in D$. I and v are functions, i.e. sets of tuples. So set $D_2 := D$, $I_2 := I \cup \{(P, \emptyset) : P \in \Pi(A_2) \setminus \Pi(A)\}$, $v_2 := v \cup \{(x, d) : x \in \Lambda(A_2) \setminus \Lambda(A)\}$. Then I_2, v_2 satisfies Φ. $\square$

Proof (4.9). Assume Φ is inconsistent in A. Then there is a proof P of $\perp$ in A from Φ. Since every formula in every sequence of P may be formulated in A_2, P is a proof of $\perp$ in A_2 from Φ. But Φ was assumed to be consistent in A_2 $\lightning$. $\square$

All these seem easy enough so that in a natural-language discussion of the subject, they may just as well be left out entirely. However, for formalization in Mizar, a considerable amount of manual work is required. Firstly one has to assert the types of all used objects (e.g. that the v defined in the proof of Theorem 4.7 is a valuation of A; otherwise it would not be in the domain of $\models$ and the formula $I, v \models \Phi$ would not be well-formed in Mizar). That the modified models still satisfiy the given sets of formulae has to be proved in detail by structural induction, and for Theorem 4.9 every kind of correct step in a proof (nine in total[5, def 7]) has to be meticulously transformed into a step in the same proof, but with respect to A_2. A general pattern emerges: For every transformation between different alphabets one first has to get rid of all types and break the used objects down to their basic representation as sets. Then the type-information (only with respect to the other alphabet) has to be rebuilt from these sets. Basically a chain of definitions is "descended" until alphabet-dependent type information is lost and then "ascended" again for another alphabet. Since in some cases there is an extensive layer of abstractions between an object and its representation as set, this contributes a lot to the bulk of the formalized proofs.

In theory a proof of Theorem 4.10 would be possible with an argument similar to the one in Theorem 4.9, but one would have to choose the proof of $\perp$ carefully, so it can be fully transformed into the smaller language. Since it is in principle possible to construct proofs with a lot of non-conducive statements and sub-inferences, this would require a subtle deconstruction and reassembly of the proof. Luckily Theorem 4.10 can be inferred non-elementarily, but easily, from the work done thus far.

Proof (4.10). It is shown: Every finite $\Psi \subseteq \Phi$ is consistent in A_2. Then the consistency of Φ in A_2 follows from [14, Lemma 4.7.4] resp. [7, Theorem 7]. So let $\Psi \subseteq \Phi$ be finite. Ψ is consistent in A. By Corollary 4.5 and Theorem 4.6 there is a countable subalphabet A_1 of A such that $\Psi \subseteq \Omega(A_1)$, and Ψ contains only finitely many unbound variables. By Theorem 4.9, Ψ is consistent in A_1. Then, by the countable Gödel Completeness Theorem [8, Theorem 34], Ψ is satisfiable in A_1. Since $A_1 \subseteq A \subseteq A_2$, it follows from Theorem 4.8 that Ψ is satisfiable in A_2. Hence Ψ is consistent in A_2 by [14, Lemma 4.7.5] resp. [7, Theorem 12]. $\square$

Now we can obtain a more general result. This was not proven in Mizar, because it was not needed later on, but it is a nice application of the four language transfer theorems by its own right.

Corollary 4.11 (General Language Transfer). *Let A_2 be some arbitrary alphabet. Assume $\Phi \subseteq \Omega(A)$ and $\Phi \subseteq \Omega(A_2)$. Then Φ is satisfiable in A iff Φ is satisfiable in A_2, and Φ is consistent in A iff Φ is consistent in A_2.*

Proof. Since both A and A_2 are supersets of $\mathbb{N}^2$, the set $A_1 := A \cap A_2$ is an alphabet and $\Phi \subseteq \Omega(A_1)$. Both A and A_2 are A_1-extending. The equivalencies follow directly from Theorems 4.7, 4.8, 4.9 and 4.10 by "moving" Φ from A through A_1 to A_2 and back. $\square$

This conveniently allows us to neglect specifying which language is ment when talking about consistency or satisfiability. That is while talking abstractly, because in Mizar the theorems need to be referenced explicitly for every language transfer.

5 The Completeness Theorem

To conduct the proof of the Gödel Completeness Theorem as given by Ebbinghaus et al.[14], the alphabet needs to be extended by adding a previously unused constant symbol $c_{\exists xp}$ for every $\exists xp \in \Omega(A)$. Equivalently one may add $c_{x,p}$ for every $x \in \Lambda(A), p \in \Omega(A)$. Naively one might try to add the tuples (x, p) to $\Sigma(A)$, but it is not possible to assert that no tuple of this form is already in $\Sigma(A)$. Instead consider the following theorem:

Theorem 5.1 (GOEDCPOC:1). *For every alphabet A there is a set s such that $\forall x \in \Lambda(A), p \in \Omega(A) : (s, (x, p)) \notin \Sigma(A)$.*

Proof. Assume there is no such s. We shall prove that $\bigcup\bigcup\Sigma(A) = V$, which is a contradiction since $\Sigma(A)$ is a set.

Let s be an arbitrary set. By the assumption consider $x \in \Lambda(A), p \in \Omega(A)$ such that $\{\{s\}, \{s, (x, p)\}\} = (s, (x, p)) \in \Sigma(A)$. Then $\{s\} \in \bigcup\Sigma(A)$, i.e. $s \in \bigcup\bigcup\Sigma(A)$. Thus for every set s, $s \in \bigcup\bigcup\Sigma(A)$. $\lightning$ $\square$

The formalized version of this proof is remarkably concise when compared to most other formalizations, so this may be considered an elegant translation of the intuitively quite clear concept of “unused constant symbols”. Now, by Theorem 5.1, we can fix one c_A for every alphabet A such that $\forall x \in \Lambda(A), p \in \Omega(A) : (c_A, (x, p)) \notin \Sigma(A)$ and define the extensions we need.

Definition 5.2 (GOEDCPOC:def 3)**.** For an alphabet A define the **Formula Constant Extension** $\mathrm{FCEx}(A)$ as $\mathbb{N} \times (\Sigma(A) \cup \{(c_A, (x, p)) : x \in \Lambda(A), p \in \Omega(A)\})$.

Obviously $\mathrm{FCEx}(A)$ is an A-extending alphabet. Now Ebbinghaus et al. recursively construct a countable sequence of extensions:

$$S_0 := A \text{ and } S_{n+1} = \mathrm{FCEx}(S_n) \text{ [14, p. 89]}$$

Interestingly, the formalization of this simple recursion ended up being complex and convoluted, mostly due to issues with typing. Trying to convert above recursive formula directly into Mizar led to the following problem: FCEx takes an alphabet as its argument, therefore Mizar needs to know that S_n is an alphabet. It is, of course, but to accept the term `FCEx(S(n))`, Mizar would need to consider S as a function with codomain being the class of all alphabets, which is a proper class (every transfinite ordinal may be considered a symbol set of an alphabet) and therefore cannot be “used” in Mizar, i.e. $\{A : A \text{ is alphabet}\}$ is an invalid expression. While *functors* may be defined over proper classes, *Functions inside* the universe of the Mizar language may not, because the type of every object widens to `set`. And, to my knowledge, it is not possible to define *functors* recursively. The following workaround resolved this by firstly constructing *finite* sequences (so complete induction is available for the proof) and extracting a countably infinite sequence (i.e. a function on $\mathbb{N}$) from them. It should be mentioned that while I use “the” in the following definitions, indicating that the defined object is unique by its parameters, no such uniqueness was proven in Mizar. Instead I proved only their existence and then selected instances by an invocation of the Axiom of Global Choice, using the “the” operator built into Mizar.

Definition 5.3 (GOEDCPOC:def 7)**.** Let A be an alphabet and $k \in \mathbb{N}$. The **FCEx-Sequence of** A **and** k is a finite sequence $\mathrm{FCEx}_{A,k}$ with length $k + 1$ such that

i. for $n \in k + 1$, $\mathrm{FCEx}_{A,k}(n)$ is an alphabet,

ii. $\mathrm{FCEx}_{A,k}(0) = A$

iii. for every $n \in k$ there is an alphabet A' such that $\mathrm{FCEx}_{A,k}(n) = A'$ and $\mathrm{FCEx}_{A,k}(n + 1) = \mathrm{FCEx}(A')$.

Definition 5.4 (GOEDCPOC:def 8)**.** Let $k \in \mathbb{N}$. Define the **k-th Formula Constant Extension of** A by $\mathrm{FCEx}_A(k) := \mathrm{FCEx}_{A,k}(k)$.

The existence of FCEx-Sequences needs to be explicitly proven (again due to the issues with typing); this is done by complete induction over k for any fixed A.

Now set $c_{\exists xp} := (4, (c_A, (x, p)))$ for any alphabet A, $x \in \Lambda(A), p \in \Omega(A)$. Then we can iteratively extend any set of formulae to contain examples:

Definition 5.5 (GOEDCPOC:def 5)**.** Let A be an alphabet. For $x \in \Lambda(A), p \in \Omega(A)$ the **Example Formula (EF) of** p and x is the formula $\mathrm{EF}(x, p) := \neg\exists xp \vee p\frac{c_{\exists xp}}{x}$.

Definition 5.6 (GOEDCPOC:def 6)**.** Let A be an alphabet. The **Example Formulae of** A are the set $\mathrm{EF}(A) := \{\mathrm{EF}(x, p) : x \in \Lambda(A), p \in \Omega(A)\}$.

Evidently $\mathrm{EF}(x, p) \in \Omega(\mathrm{FCEx}(A))$ and hence $\mathrm{EF}(A) \subseteq \Omega(\mathrm{FCEx}(A))$ for any A, x, p. Note that in the formalization, typecasts were used extensively to move the formulae "up" to the Formula Constant Extension.

Definition 5.7 (GOEDCPOC:def 9)**.** Let A be an alphabet and $\Phi \subseteq \Omega(A)$. The **Example-Formulae-Sequence of** Φ is a countably infinite sequence EF_Φ such that

i. $\mathrm{EF}_\Phi(0) = \Phi$,

ii. for $n \in \mathbb{N}$ holds $\mathrm{EF}_\Phi(n+1) = \mathrm{EF}_\Phi(n) \cup \mathrm{EF}(\mathrm{FCEx}_A(n))$.

Definition 5.8. Let $k \in \mathbb{N}$. Define the **k-th Example-Formulae of** Φ as $\mathrm{EF}_\Phi(k)$.

Since for the recursion step no type-information was relevant, this definition is much easier and more straightforward than Definition 5.3 and Definition 5.4. Some basic results about the structure of the extensions can be obtained:

Theorem 5.9 (GOECPOC:4,5,6)**.** **i.** *For all $n \in \mathbb{N}$, $\mathrm{FCEx}_A(n)$ is an A-extending alphabet.*

Proof. trivial. □

ii. *For $k \in \mathbb{N}$, $\mathrm{FCEx}(\mathrm{FCEx}_A(k)) = \mathrm{FCEx}_A(k+1)$.*

Proof. Show by induction over n for any fixed k: $\mathrm{FCEx}_{A,k}(n) = \mathrm{FCEx}_{A,k+1}(n)$ for $n \leq k$. The result follows by Definition 5.3(iii). Some border cases may need separate attention. □

iii. *For $n, m \in \mathbb{N}, n \leq m$, $\mathrm{FCEx}_A(n) \subseteq \mathrm{FCEx}_A(m)$.*

Proof. Fix m. Show by induction that for $k \in \mathbb{N}, k \leq m$, $\mathrm{FCEx}_A(m - k) \subseteq \mathrm{FCEx}_A(m)$. The base case $k = 0$ is trivial; as for the inductive step:

$$\begin{aligned} \mathrm{FCEx}_A(m - (k+1)) &\subseteq \mathrm{FCEx}(\mathrm{FCEx}_A(m - (k+1))) \\ &= \mathrm{FCEx}_A(m - k) \text{ by (ii)} \\ &\subseteq \mathrm{FCEx}_A(m) \text{ by induction hypothesis.} \end{aligned}$$

The border case $k = m$ needs special consideration, since $(m-(k+1))$ would be negative. However, it follows trivially by (i). □

Now the proofs of [14, Lemma 5.3.4, Lemma 5.3.1, Lemma 5.3.2] can be directly formalized. As usual, a set Φ of well-formed formulae in a given alphabet A is said to **contain examples** if for any $x \in \Lambda(A), p \in \Omega(A)$, $\Phi \vdash \neg\exists xp \vee p\frac{c}{x}$ for some $c \in \Lambda(A)$, and to be **negation faithful** if for any $p \in \Omega(A)$, $\Phi \vdash p$ or $\Phi \vdash \neg p$.

Theorem 5.10 (GOEDCPOC:9)**.** *For an arbitrary alphabet A and consistent $\Phi \subseteq \Omega(A)$, $\Phi \cup \mathrm{EF}(A)$ is consistent.* [14, Lemma 5.3.4]

Proof. Set $\chi := \Phi \cup \mathrm{EF}(A)$. We show that every finite Subset of χ is consistent. Let $\chi' \subseteq \chi$ be finite. Set $\Phi' := \Phi \cap \chi'$. Then $\chi' = \Phi' \cup (\mathrm{EF}(A) \cap \chi')$. Because $\Phi' \subseteq \Phi$, Φ' is consistent. Then Φ' is satisfiable in some countable subalphabet of A by Corollary 4.5, Theorem 4.6 and the countable Gödel Completeness Theorem [8, Theorem 34]. So Φ' is satisfiable in $\mathrm{FCEx}(A)$ by Theorem 4.8. Take a model (D, J, v) of $\mathrm{FCEx}(A)$ that satisfies Φ'. Change (D, J, v) to a model of χ': Take $q \in \mathrm{EF}(A) \cap \chi'$. $q = \neg\exists xp \vee p\frac{c_{\exists xp}}{x}$ for some $x \in \Lambda(A), p \in \Omega(A)$. If $(D, J, v) \not\models \exists xp$ then $(D, J, v) \models q$. Otherwise take $d \in D$ such that $(D, J, v) \models p\frac{d}{x}$. Change v to a new interpretation v' by interpreting $c_{\exists xp}$ as d. Since $c_{\exists xp}$ is in no other formula than q, (D, J, v') satisfies everything (D, J, v) satisfied and also q. Iterating this process for each $q \in \mathrm{EF}(A) \cap \chi'$ gives a model that satisfies χ'. Therefore χ' is consistent by [14, Lemma 4.7.5] resp. [8, Theorem 12]. □

Using the whole strength of the previously obtained results and techniques, the formalization of this proof was remarkably direct and straightforward. However, it is not particularly short due to many case distinctions needed for changing the model.

Theorem 5.11 (GOEDCPOC:10,12)**.** *Let A be an alphabet. Let $\Phi \subseteq \Omega(A)$ be consistent.*

i. *There is an A-extending alphabet A_2 and a consistent $\Psi \subseteq \Omega(A_2)$ such that $\Phi \subseteq \Psi$ and Ψ contains examples.* [14, Lemma 5.3.1].

ii. *There is a consistent $\Theta \subseteq \Omega(A)$ such that $\Phi \subseteq \Theta$ and Θ is negation faithful.* [14, Lemma 5.3.2].

Proof (i). Set $A_2 := \bigcup\{\mathrm{FCEx}_A(k) : k \in \mathbb{N}\}$ and $\Psi := \bigcup\{\mathrm{EF}_\Phi(k) : k \in \mathbb{N}\}$. Then the proof can be broken down as follows:

i. $\Phi \subseteq \Psi$: Trivial.

ii. A_2 is an alphabet and $\Psi \subseteq \Omega(A_2)$: Every $p \in \Psi$ is in $\mathrm{EF}_\Phi(k)$ for some k and therefore in $\Omega(\mathrm{FCEx}_A(k))$. Hence $p \in \Omega(A_2)$.

iii. Ψ is consistent: By induction using Theorem 5.10 show that every $\mathrm{EF}_\Phi(k)$ is consistent. Hence, by Theorem 4.10, every $\mathrm{EF}_\Phi(k)$ is consistent in A_2. Thus Ψ is the union of an ascending set of consistent sets in A_2, therefore Ψ is consistent itself by [7, Theorem 11].

iv. Ψ contains examples: Take $x \in \Lambda(A_2), p \in \Omega(A_2)$. Then $x \in \Lambda(\mathrm{FCEx}_A(k))$, $p \in \Omega(\mathrm{FCEx}_A(k))$ for some k. Then $\mathrm{EF}(x,p) \in \mathrm{EF}_\Phi(k+1)$, therefore $\mathrm{EF}(x,p) \in \Psi$, hence $\Psi \vdash \mathrm{EF}(x,p)$.

$\square$

The application of Theorem 4.10 to [7, Theorem 11] to obtain the result of [14, 4.7.7], saying that a union of ascending consistent sets in an ascending series of alphabets is consistent in the union of these alphabets, was a very important step, since I expected the elementary formalized proof of [14, 4.7.7] to be very long and complicated.

Proof (ii). Set $U := \{\Psi : \Psi \subseteq \Omega(A), \Psi \text{ is consistent}, \Phi \subseteq \Psi\}$. U is partially ordered by $\subseteq$ and every chain $\mathcal{Z}$ in U has the upper bound $\bigcup \mathcal{Z}$: Obviously every $Z \in \mathcal{Z}$ is a subset of $\bigcup \mathcal{Z}$ and $\Phi \subseteq \bigcup \mathcal{Z}$. Assume $\bigcup \mathcal{Z}$ is not consistent. Then some finite subset Y of $\bigcup \mathcal{Z}$ is inconsistent. Since Y is finite there is a finite $\mathcal{R} \subseteq \mathcal{Z}$ such that $Y \subseteq \bigcup \mathcal{R}$. Now $\mathcal{R}$ can be ordered into an ascending chain, i.e. $R_1 \subseteq R_2 \subseteq \cdots \subseteq R_n$ for some enumeration of the $R_i \in \mathcal{R}$. Therefore $\bigcup \mathcal{R} \in \mathcal{R}$. But then $\bigcup \mathcal{R}$ is in U and is thus consistent. So $Y \subseteq \bigcup \mathcal{R}$ cannot be inconsistent $\lightning$.

Then by Zorn's lemma take a maximal element $\Theta \in U$. Θ is negation faithful, since if there were a $p \in \Omega(A)$ such that neither $\Theta \vdash p$ nor $\Theta \vdash \neg p$, either $\Theta \cup \{p\}$ or $\Theta \cup \{\neg p\}$ would be consistent. Then Θ would not be maximal in U $\lightning$. $\square$

In Mizar one cannot just "order a set" without significantly altering its internal structure (i.e. adding indices, thus making it a function or sequence), so the formalization had to work around this issue. So I inductively proved that $\bigcup \mathcal{R}' \in \mathcal{R}'$ for any $\emptyset \subset \mathcal{R}' \subseteq \mathcal{R}$ by starting with singletons and incrementally adding elements. This incremental approach seems to be an adequate translation of the concept of "ordering": In essence this is some kind of sorting algorithm, taking elements individually and putting them where they belong.

Now the Model Existence Theorem and then the Gödel Completeness Theorem are obtained immediately (though some transformations formally require the Language Transfer Theorems).

Theorem 5.12 (Model Existence Theorem, GOEDCPOC:14)**.** *Let A be an alphabet, $\Phi \subseteq \Omega(A)$. If Φ is consistent, then Φ is satisfiable.*

Proof. By Theorem 5.11 choose an A-extending alphabet A_2 and a $\Theta \subseteq \Omega(A_2)$ such that $\Phi \subseteq \Theta$, and Θ is negation faithful and contains examples. Then the Henkin Model of Θ in A_2 satisfies Θ by Henkins Theorem[8, Theorem 17]. Hence Φ is satisfiable. $\square$

Theorem 5.13 (Gödel Completeness Theorem, GOEDCPOC:15)**.** *Let A be an alphabet, $\Phi \subseteq \Omega(A)$, $p \in \Omega(A)$. $\Phi \models p \Rightarrow \Phi \vdash p$.*

Proof. Assume $\Phi \models p$ and not $\Phi \vdash p$. Then $\Phi \cup \{\neg p\}$ is consistent by [7, Theorem 9]. Then $\Phi \cup \{\neg p\}$ is satisfiable by Theorem 5.12. $\lightning$ by [8, Theorem 37]. $\square$

6 Remarks

As hinted above, I consider the proofs and formalizations of Theorem 4.10 and Theorem 5.1 to be of a particular elegance, due to the low formal bulk they require. Similarily, the Typecasting Lemma 4.3 has remarkable impact despite having a short and simple formalized proof. Indeed the conception and formalization of the typecasts and the Typecasting Lemma was a major breakthrough, since only then structural inductions (which I frequently used) across different languages became possible. Of similar importance was the systematic division of the Language Transfer Theorems into 4 cases and the simple proof of the last one from the 3 others. Their introduction significantly shorted some proofs in GOEDCPOC which are quite long even in their final version. Also the application of Theorem 4.10 in the proof of Theorem 5.11(i.) was invaluable. Nevertheless: The complete proof of the general case of the Gödel Completeness Theorem by Ebbinghaus et al.[14] takes about 4 pages in the book, whereas this *formalized* proof spans about 3800 lines which constitutes over 40 pages.

Originally the combined size of the FO_TRANS and GOEDCPOC articles was about 4500 lines, but I managed to remove a significant number of irrelevant inferences using tools provided by the Mizar System. However, this greatly reduced the readability of the proofs in some cases. When writing, I followed a "human" natural reasoning style, trying to get propositions as general as possible and deduce the desired results linearly from there, i.e. building a logical sound sequence of interconnected conclusions. Mizar uses a different approach, apparently. It prefered to prove every single statement in a sequence as efficiently as possible without regard to a greater context, which sometimes led to strange results. For instance, if I proved a general statement beforehand, sometimes Mizar decided that wherever I used it, it could infer this conclusion directly from the premises of the general statement. So, if I refered multiple times to the general statement, Mizar would prove it anew every time (in the background, though). This way some proofs became quite confusing; they give the impression that I introduce some objects, amass ostensibly completely unrelated statements about these objects and finish with a large number of references from which the Mizar Checker can somehow derive the desired result. This is akin to what I believe human mathematicians do when they have no idea in which direction the proof is going to go, so they aimlessly collect propositions until these are powerful enough to infer whatever statement they are trying to prove. I guess this is fitting, since Mizar surely has no idea how a proof is *supposed* to go as well.

All in all, the difficulties of working with different languages and most notably the typing-related issues this caused were among the biggest hurdles to overcome and the most severe cause of inflation in the proofs. The existing language definition was largely ill-equipped for this, which is somewhat understandable, considering that working with multiple languages was never intended. If this had been foreseen, there could have been some methods available to mend this problems. For instance, any object in Mizar can only have one *mode* but multiple *attributes*. Right now "being a well-formed formula of A" is a mode, so any object can only be well-formed in relation to a single alphabet at any given time. Were it formalized as an attribute of a wider mode (`set` might suffice) one might consider the well-formed property in relation to two or more alphabets. Also, using the `cluster` registration in Mizar, attributes can be assigned automatically under certain conditions, whereas mode-changes are always explicity invoked. To my understanding, recent work by Marco Caminati[11] on the formalization of first order languages does just that. Since Caminati constructed the whole theory anew from scratch, without the aforementioned historical issues, his framework seems to be generally better suited for future work in this area. Nevertheless, I hope that my work alleviates some of the most severe issues.

Considering that I needed to develop the (formalized) theory of language transfers completely anew, without any previous work to build on, the inflation of the proof of the Gödel Completeness Theorem may be not as severe as it seems. Using a framework more fitted to the intended use might have halved the total number of lines. It would maybe be interesting to construct the same proof using Caminatis framework. I suspect that it might be much shorter.

Another major cause of inflation were the techniques for variable substitution, formalized by Braselmann and Koepke[2, 3, 4]. The framework seems to be overly convoluted, reinventing the whole language anew as a substitution-language with wide generality. Again limited by the static typing, results about standard formulae do not easily carry over to substituted formulae. The analogon to the Typecasting Lemma for substituted formulae, Theorem 18 in FO_TRANS, saying that equal substitutions on equal formulae in different languages yield the same resulting formulae, took multiple sublemmata, a series of inductions and all in all a lot of time and space to prove, despite being seemingly trivial. Quantified formulae were the most problematic, since some higher level theorems about them did not exist. So I had to do some work with the bare definitions. As I cannot oversee the full extent of the problems attached to formalizing variable substitution, I am however not able to guess whether this could be substantially improved.

On the non-technical side, certain natural-language mathematical idioms caused the proofs to grow. Apparently mathematicians tend to assure that one "can do" certain transformations, but never give them explicitly. Probably because they are long, require tedious case-by-case-analysis and are obvious, anyway. However, Mizar knows no such thing, so one always has to walk it through such a transformation. An example can be found in the proof of Lemma 5.3.4 by Ebbinghaus et al.[14] (Theorem 9 in GOEDCPOC) where the construction of a satisfying model takes about one line in natural language, but is a large effort formally, requiring multiple nested distinctions of cases and some induction. It actually takes the greatest part of the proof, by far. Also, some definitions I have written may seem unnecessary in their contents and would be regarded useless in natural-language mathematics, but many of them are "syntactic sugar" in Mizar to actually shorten the proofs.

7 Automatic Proof Checking

As mentioned in the introduction, a number of preliminary results from the Mizar Mathematical Library needed to be modified to accomodate arbitrarily large languages. So to check the proof with the Mizar system, one needs to add the modified versions of QC_LANG1 [15] (FO_LANG1), QC_LANG2 [1] (FO_LANG2), QC_LANG3 [9] (FO_LANG3), CQC_LANG [10] (CFO_LANG), CQC_THE1 [12] (CFO_THE1), CQC_SIM1 [13] (CFO_SIM1), VALUAT_1 [16] (FO_VALUA), SUBSTUT1 [2](SUBST1_2), SUBLEMMA [3] (SUBLEM_2), SUBSTUT2 [4] (SUBST2_2), CALCUL_1 [5] (CALCL1_2), CALCUL_2 [6] (CALCL2_2), HENMODEL [7] (HENMOD_2) and GOEDELCP [8] (GOEDCP_2) in this order to the local library using the `miz2prel` utility. Afterwards the article FO_TRANS can be successfully checked by `mizf` and added to the local library as well. Now one may check the GOEDCPOC article. The complete text of the FO_TRANS and GOEDCPOC articles, as well as `miz2abs`-generated abstracts (containing definitions and theorems) of the modified articles are in the appendix. The FO_TRANS and GOEDCPOC articles as attached are proof checked by `mizf`, and their proofs have been optimized by all available tools in the Mizar System distribution: `relprem`, `relinfer`, `reliters`, `trivdemo`, `chklab`, `inacc`, `irrvoc` and `irrths`. As mentioned before, this may cause some proofs to be less readable than actually intended by me. I used Mizar version 7.12.01 (Linux/FPC) with MML version 4.166.1132. These are the latest versions as of July 5, 2011.

References

[1] Grzegorz Bancerek, Connectives and Subformulae of the First Order Language. *Formalized Mathematics* 1 (1990).

[2] Patrick Braselmann, Peter Koepke, Substitution in first-order formulas: elementary properties. *Formalized Mathematics* 13 (2005).

[3] Patrick Braselmann, Peter Koepke, Coincidence lemma and substitution lemma. *Formalized Mathematics* 13 (2005).

[4] Patrick Braselmann, Peter Koepke, Substitution in first-order formulas. Part II. The construction of first-order formulas. *Formalized Mathematics* 13 (2005).

[5] Patrick Braselmann, Peter Koepke, A sequent calculus for first order logic. *Formalized Mathematics* 13 (2005).

[6] Patrick Braselmann, Peter Koepke, Consequences of the sequent calculus. *Formalized Mathematics* 13 (2005).

[7] Patrick Braselmann, Peter Koepke, Equivalences of inconsistency and Henkin models. *Formalized Mathematics* 13 (2005).

[8] Patrick Braselmann, Peter Koepke, Gödel's completeness theorem. *Formalized Mathematics* 13 (2005).

[9] Czesław Byliński, Grzegorz Bancerek, Variables in Formulae of the First Order Language. *Formalized Mathematics* 1 (1990).

[10] Czesław Byliński, A Classical First Order Language. *Formalized Mathematics* 1 (1990).

[11] Marco Caminati, First order languages: syntax, part two: semantics. *Formalized Mathematics* (in print).

[12] Agata Darmochwał, A First–Order Predicate Calculus. *Formalized Mathematics* 1 (1990).

[13] Agata Darmochwał, Andrzej Trybulec, Similarity of Formulae. *Formalized Mathematics* 2 (1991).

[14] Heinz-Dieter Ebbinghaus, Jörg Flum, Wolfgang Thomas, Einführung in die mathematische Logik. *Spektrum Akademischer Verlag* 5. Edition (2007).

[15] Piotr Rudnicki, Andrzej Trybulec, A First Order Language. *Formalized Mathematics* 1 (1990).

[16] Edmund Woronowicz, Interpretation and Satisfiability in the First Order Logic. *Formalized Mathematics* 1 (1990).

8 Appendix - FO_TRANS and GOEDCPOC

8.1 FO_TRANS

```
environ

 vocabularies NUMBERS, SUBSET_1, FO_LANG1, CFO_LANG, XBOOLE_0, FO_VALUA,
      FINSEQ_1, HENMOD_2, CFO_THE1, XBOOLEAN, BVFUNC_2, FUNCT_1, ORDINAL4,
      CALCL1_2, ARYTM_3, RELAT_1, CARD_1, XXREAL_0, TARSKI, ZF_MODEL, REALSET1,
      SUBST1_2, SUBST2_2, ZF_LANG, ARYTM_1, CARD_3, ZFMISC_1, FINSET_1,
      MCART_1, NAT_1, MARGREL1,FUNCT_2, FUNCOP_1, FO_TRANS, ZF_LANG1, FUNCT_4,
      CLASSES2, SUBLEM_2;
 notations TARSKI, XBOOLE_0, ZFMISC_1, SUBSET_1, XCMPLX_0, XXREAL_0, NAT_1,
      CARD_1, CARD_3,FINSEQ_1, RELAT_1, FO_LANG1, FO_LANG2, NUMBERS, CFO_THE1,
      CFO_LANG, FUNCT_1, FINSET_1, FO_VALUA, RELSET_1, FUNCT_2, CFO_SIM1,
      DOMAIN_1, MCART_1, SUBST1_2, SUBLEM_2, SUBST2_2, CALCL1_2, HENMOD_2,
      ORDINAL1, GOEDCP_2, MARGREL1, FO_LANG3, FUNCT_4, FUNCOP_1;
 constructors SETFAM_1, DOMAIN_1, XXREAL_0, NAT_1, NAT_D, FINSEQ_2, FO_LANG1,
      CFO_THE1, CFO_SIM1, SUBST1_2, SUBLEM_2, SUBST2_2, CALCL1_2, HENMOD_2,
      CARD_3, RELSET_1, CARD_1, WELLORD2, GOEDCP_2, FO_VALUA, MARGREL1,
      CFO_LANG, FO_LANG3, FUNCT_4, FUNCOP_1;
 registrations SUBSET_1, RELAT_1, FUNCT_1, ORDINAL1, XXREAL_0, XREAL_0,
      HENMOD_2, FINSEQ_1, FINSET_1, CARD_3, XBOOLE_0, FO_LANG1, CFO_LANG,
      MARGREL1, FO_VALUA, CARD_1, GOEDCP_2, FUNCT_4, FUNCOP_1, SUBLEM_2,
      SUBST1_2;
 requirements REAL, NUMERALS, SUBSET, BOOLE, ARITHM;
 definitions TARSKI, XBOOLE_0, GOEDCP_2;
 theorems TARSKI, FUNCT_1, MCART_1, XBOOLE_0, XBOOLE_1, CFO_LANG, FO_LANG1,
     ZFMISC_1, RELAT_1, FO_LANG3, FO_LANG2, HENMOD_2, CALCL1_2, SUBLEM_2,
     NAT_1, FINSEQ_1, FO_VALUA, FUNCT_2, SUBST2_2, CFO_SIM1, CARD_2, ORDINAL1,
     CARD_1, GOEDCP_2, FUNCOP_1, FINSEQ_2, FINSET_1, SUBST1_2, FUNCT_4, CARD_4;
 schemes CFO_LANG, FINSET_1, FRAENKEL;

begin

reserve Al for FO-alphabet;

reserve PHI for Consistent Subset of CFO-WFF(Al),
        p,q,r,s for Element of CFO-WFF(Al),
        A for non empty set,
        J for interpretation of Al,A,
        v for Element of Valuations_in(Al,A),
        m,n,i,j,k for Element of NAT,
        l for CFO-variable_list of k,Al,
        P for FO-pred_symbol of k,Al,
        x,y,z for bound_FO-variable of Al,
        b for FO-symbol of Al,
        PR for FinSequence of [:set_of_CFO-WFF-seq(Al),Proof_Step_Kinds:];

definition
  let Al;
  let Al2 be FO-alphabet;
  attr Al2 is Al-extending means
  :Def1:
  Al c= Al2;
end;

registration
  let Al;
  cluster Al-extending for FO-alphabet;
  existence
  proof
    Al is Al-extending by Def1;
    hence thesis;
  end;
end;

registration
  let Al1,Al2 be countable FO-alphabet;
  cluster countable Al1-extending Al2-extending for FO-alphabet;
  existence
  proof
    set Al3 = Al1 \/ Al2;
    Al1 = [:NAT,FO-symbols(Al1):] & Al2 =[:NAT,FO-symbols(Al2):] by FO_LANG1:5;
    then
A1: Al3 = [:NAT, FO-symbols(Al1) \/ FO-symbols(Al2):] by ZFMISC_1:97;
    NAT c= FO-symbols(Al1) \/ FO-symbols(Al2) by XBOOLE_1:10,FO_LANG1:3;
    then reconsider Al3 as FO-alphabet by A1,FO_LANG1:def 1;
A2: Al1 c= Al3 & Al2 c= Al3 by XBOOLE_1:7;
    take Al3;
    thus thesis by A2,Def1,CARD_2:85;
  end;
end;

definition
  let Al,Al2 be FO-alphabet;
  let P be Subset of CFO-WFF(Al);
  attr P is Al2-Consistent means :Def2:
   for S being Subset of CFO-WFF(Al2) st P=S holds S is Consistent;
end;

registration
   let Al;
   cluster non empty Consistent for Subset of CFO-WFF(Al);
   existence
   proof
     {VERUM(Al)} is Consistent by HENMOD_2:13;
     hence thesis;
   end;
end;

registration
   let Al;
   cluster Consistent -> Al-Consistent for Subset of CFO-WFF(Al);
   coherence
   proof
     let P be Subset of CFO-WFF(Al) such that
A1:   P is Consistent;
     for S being Subset of CFO-WFF(Al) st S=P holds
     S is Consistent by A1;
     hence P is Al-Consistent by Def2;
   end;
   cluster Al-Consistent -> Consistent for Subset of CFO-WFF(Al);
   coherence by Def2;
end;

reserve Al2 for Al-extending FO-alphabet,
        J2 for interpretation of Al2,A,
        Jp for interpretation of Al,A,
        v2 for Element of Valuations_in(Al2,A),
        vp for Element of Valuations_in(Al,A);

theorem Th1:
  len l = k & the_arity_of P = len l
proof
  thus len l = k by CARD_1:def 7;
  thus len l = k by CARD_1:def 7 .= the_arity_of P by FO_LANG1:11;
end;

theorem Th2:
  FO-symbols(Al) c= FO-symbols(Al2)
proof
  Al c= Al2 & Al = [: NAT, FO-symbols(Al) :] &
  Al2 = [:NAT,FO-symbols(Al2) :] by Def1, FO_LANG1:5;
  hence FO-symbols(Al) c= FO-symbols(Al2) by ZFMISC_1:115;
end;

theorem Th3:
   FO-pred_symbols(Al) c= FO-pred_symbols(Al2)
proof
  for Q being set st Q in FO-pred_symbols(Al) holds Q in FO-pred_symbols(Al2)
  proof
    let Q be set such that
A1: Q in FO-pred_symbols(Al);
    set preds = { [k,b] : 7 <= k };
    set preds2 = { [k,b2] where b2 is FO-symbol of Al2 : 7 <= k };
    Q in preds by A1, FO_LANG1:def 7;
    then consider k,b such that
```

```
A2: Q=[k,b] & 7 <= k;
    FO-symbols(Al) c= FO-symbols(Al2) by Th2;
    then b in FO-symbols(Al2) by TARSKI:def 3;
    then Q in preds2 by A2;
    hence Q in FO-pred_symbols(Al2) by FO_LANG1:def 7;
  end;
  hence thesis by TARSKI:def 3;
end;

theorem Th4:
  bound_FO-variables(Al) c= bound_FO-variables(Al2)
proof
A1: FO-symbols(Al) c= FO-symbols(Al2) by Th2;
  bound_FO-variables(Al) = [: {4}, FO-symbols(Al) :] &
  bound_FO-variables(Al2) = [: {4}, FO-symbols(Al2) :]  by FO_LANG1:def 4;
  hence thesis by A1, ZFMISC_1:96;
end;

theorem Th5:
  for k,l holds l is CFO-variable_list of k,Al2
proof
  let k,l;
  rng l c= bound_FO-variables(Al) &
     bound_FO-variables(Al) c= bound_FO-variables(Al2) by Th4;
  then
A1: rng l c= bound_FO-variables(Al2) by XBOOLE_1:1;
  then rng l c= FO-variables(Al2) by XBOOLE_1:1;
  hence thesis by A1,FINSEQ_1:def 4;
end;

theorem Th6:
  for Al2, k,P holds P is FO-pred_symbol of k,Al2
proof
  let Al2,k,P;
A1: P is FO-pred_symbol of Al &
  FO-pred_symbols(Al) c= FO-pred_symbols(Al2) by Th3;
  the_arity_of P = k by FO_LANG1:11;
  then
A2: P`1 = 7+k by FO_LANG1:def 8;
  reconsider P as FO-pred_symbol of Al2 by A1,TARSKI:def 3;
  the_arity_of P = k by FO_LANG1:def 8, A2;
  hence thesis by FO_LANG3:1;
end;

theorem Th7:
  for Al2 being Al-extending FO-alphabet
  for p holds p is Element of CFO-WFF(Al2)
proof
  let Al2 be Al-extending FO-alphabet;
  defpred P[Element of CFO-WFF(Al)] means $1 is Element of CFO-WFF(Al2);
A0: for p for q being Element of CFO-WFF(Al2) st p=q holds @p = @q
  proof
    let p;
    let q be Element of CFO-WFF(Al2) such that
A1: p=q;
    thus @p = p by FO_LANG1:def 13
           .= @q by A1,FO_LANG1:def 13;
  end;
T1: P[VERUM(Al)]
  proof
    VERUM(Al) = <*[0,0]*> by FO_LANG1:def 14
             .= VERUM(Al2) by FO_LANG1:def 14;
    hence thesis;
  end;
T2: for k,P,l holds P[P!l]
  proof
    let k,P,l;
A2: the_arity_of P = len l by Th1;
    P is FO-pred_symbol of k,Al2 & l is CFO-variable_list of k,Al2 by Th5,Th6;
    then consider P2 being FO-pred_symbol of k,Al2,
    l2 being CFO-variable_list of k,Al2 such that
A3: P=P2 & l=l2;
    the_arity_of P2 = len l2 by Th1;
    then P2!l2 = <*P2*>^l2 by FO_LANG1:def 12;
    hence thesis by A2,A3,FO_LANG1:def 12;
  end;
T3: P[p] implies P['not' p]
  proof
    assume P[p];
    then consider q being Element of CFO-WFF(Al2) such that
A7: p = q;
    'not' p = <*[1,0]*>^@p by FO_LANG1:def 15
           .= <*[1,0]*>^@q by A7,A0
           .= 'not' q by FO_LANG1:def 15;
    hence thesis;
  end;
T4: P[p] & P[q] implies P[p '&' q]
  proof
    assume P[p] & P[q];
    then consider t,u being Element of CFO-WFF(Al2) such that
A8: p = t & q = u;
A9: @p = @t & @q = @u by A8,A0;
    p '&' q = <*[2, 0]*>^@p^@q by FO_LANG1:def 16
           .= t '&' u by A9,FO_LANG1:def 16;
    hence thesis;
  end;
T5: for x holds P[p] implies P[All(x,p)]
  proof
    let x;
    assume P[p];
    then consider q being Element of CFO-WFF(Al2) such that
A10: p = q;
    x is bound_FO-variable of Al &
    bound_FO-variables(Al) c= bound_FO-variables(Al2) by Th4;
    then x is bound_FO-variable of Al2 by TARSKI:def 3;
    then consider y being bound_FO-variable of Al2 such that
A11: x = y;
    All(x,p) = <*[3,0]*>^<*x*>^@p by FO_LANG1:def 17
            .= <*[3,0]*>^<*x*>^@q by A0,A10
            .= All(y,q) by FO_LANG1:def 17, A11;
    hence thesis;
  end;
T6: for r,s being Element of CFO-WFF(Al)
  for x being bound_FO-variable of Al for k
  for l being CFO-variable_list of k, Al for P being
  FO-pred_symbol of k,Al holds P[VERUM(Al)] & P[P!l] &
  (P[r] implies P['not' r]) & (P[r] & P[s] implies P[r '&' s]) &
  (P[r] implies P[All(x, r)]) by T1,T2,T3,T4,T5;
  for p holds P[p] from CFO_LANG:sch 1(T6);
  hence for p holds p is Element of CFO-WFF(Al2);
end;

definition
  let Al;
  let Al2 be Al-extending FO-alphabet;
  let p be Element of CFO-WFF(Al);
  func Al2-Cast(p) -> Element of CFO-WFF(Al2) equals p;
  coherence by Th7;
end;

definition
  let Al;
  let Al2 be Al-extending FO-alphabet;
  let x be bound_FO-variable of Al;
  func Al2-Cast(x) -> bound_FO-variable of Al2 equals x;
  coherence
  proof
    bound_FO-variables(Al) c= bound_FO-variables(Al2) by Th4;
    hence thesis by TARSKI:def 3;
  end;
end;

definition
  let Al;
  let Al2 be Al-extending FO-alphabet;
  let k;
  let P be FO-pred_symbol of k,Al;
  func Al2-Cast(P) -> FO-pred_symbol of k,Al2 equals P;
  coherence by Th6;
end;

definition
  let Al;
  let Al2 be Al-extending FO-alphabet;
  let k;
  let l be CFO-variable_list of k,Al;
  func Al2-Cast(l) -> CFO-variable_list of k,Al2 equals l;
  coherence by Th5;
end;


theorem Th8:
 for p,r,x,P,l for Al2 being Al-extending FO-alphabet
 holds Al2-Cast(VERUM(Al)) = VERUM(Al2) &
 Al2-Cast(P!l) = Al2-Cast(P)!Al2-Cast(l) &
 Al2-Cast('not' p) = 'not' (Al2-Cast(p)) &
 Al2-Cast(p '&' r) = (Al2-Cast(p)) '&' (Al2-Cast(r)) &
 Al2-Cast(All(x,p)) = All(Al2-Cast(x),Al2-Cast(p))
```

```
proof
  let p,r,x,P,l;
  let Al2 be Al-extending FO-alphabet;

A1: @p = p by FO_LANG1:def 13
      .= @(Al2-Cast(p)) by FO_LANG1:def 13;
A2: @r = r by FO_LANG1:def 13
      .= @(Al2-Cast(r)) by FO_LANG1:def 13;
A3: the_arity_of P = len l by Th1;
A4: the_arity_of Al2-Cast(P) = len (Al2-Cast(l)) by Th1;

  thus Al2-Cast(VERUM(Al)) = <*[0,0]*> by FO_LANG1:def 14
                          .= VERUM(Al2) by FO_LANG1:def 14;
  thus Al2-Cast(P!l) = <*P*>^l by A3,FO_LANG1:def 12
           .= Al2-Cast(P)!Al2-Cast(l) by A4,FO_LANG1:def 12;
  thus Al2-Cast('not' p) = <*[1,0]*>^@p by FO_LANG1:def 15
            .= 'not' (Al2-Cast(p)) by A1,FO_LANG1:def 15;
  thus Al2-Cast(p '&' r)
            = <*[2, 0]*>^@(Al2-Cast(p))^@(Al2-Cast(r)) by A1,A2,FO_LANG1:def 16
            .= (Al2-Cast(p)) '&' (Al2-Cast(r)) by FO_LANG1:def 16;
  thus Al2-Cast(All(x,p)) = <*[3, 0]*>^<*x*>^@p by FO_LANG1:def 17
            .= All(Al2-Cast(x),Al2-Cast(p)) by A1,FO_LANG1:def 17;
end;

theorem Th9:
 Jp = J2|FO-pred_symbols(Al) & vp = v2|bound_FO-variables(Al)
 implies (J2,v2 |= Al2-Cast(r) iff Jp,vp |= r)
proof
  defpred T[Element of CFO-WFF(Al)] means
      for J2,Jp,v2,vp holds
       Jp = J2|FO-pred_symbols(Al) & vp = v2|bound_FO-variables(Al) implies
    (( J2,v2 |= Al2-Cast($1) ) iff Jp,vp |= $1);
T1: T[VERUM(Al)]
  proof
    let J2, Jp, v2, vp;
    J2,v2 |= VERUM(Al2) by FO_VALUA:32;
    hence thesis by Th8,FO_VALUA:32;
  end;
T2: for k,P,l holds T[P!l]
  proof
    let k,P,l;
    let J2, Jp, v2, vp;
    assume
A0: Jp = J2|FO-pred_symbols(Al) & vp = v2|bound_FO-variables(Al);
    set p = P!l;
    the_arity_of P = len l by Th1;
    then
A3: P!l = <*P*>^l by FO_LANG1:def 12;
    P is FO-pred_symbol of k,Al2  & l is CFO-variable_list of k,Al2
    by Th5, Th6;
    then consider P2 being FO-pred_symbol of k,Al2,
    l2 being CFO-variable_list of k,Al2 such that
A4: P=P2 & l=l2;
A6: the_arity_of P2 = len l2 by Th1;
A7: v2*'l2 = vp*'l
    proof
A8:   bound_FO-variables(Al) c= bound_FO-variables(Al2) by Th4;
A9:   for j st 1 <= j & j <= len l holds l.j in bound_FO-variables(Al) iff
       l.j in bound_FO-variables(Al2)
      proof
        let j such that
A10:    1 <= j & j <= len l;
        thus l.j in bound_FO-variables(Al) implies
         l.j in bound_FO-variables(Al2) by A8;
        thus now
          assume l.j in bound_FO-variables(Al2);
          len l = k by Th1;
          then j in Seg k by A10, FINSEQ_1:1;
          then j in dom l by FINSEQ_1:89;
          hence l.j in bound_FO-variables(Al) by FUNCT_1:102;
        end;
      end;
      set t1 ={l.i: 1 <= i & i <= len l & l.i in bound_FO-variables(Al)};
      set t2 ={l.i: 1 <= i & i <= len l & l.i in bound_FO-variables(Al2)};
A11:  t1=t2
      proof
        thus t1 c= t2
        proof
         let x be set;
         assume x in t1;
         then consider i such that
A12:     x = l.i & 1 <= i & i <= len l & l.i in bound_FO-variables(Al);
         x = l.i & 1 <= i & i <= len l & l.i in bound_FO-variables(Al2)
           by A9,A12;
         hence x in t2;
        end;
        thus t2 c= t1
        proof
         let x be set;
         assume x in t2;
         then consider i such that
A13:     x = l.i & 1 <= i & i <= len l & l.i in bound_FO-variables(Al2);
         x = l.i & 1 <= i & i <= len l & l.i in bound_FO-variables(Al)
           by A9,A13;
         hence x in t1;
        end;
      end;
A14:  still_not-bound_in l = variables_in(l,bound_FO-variables(Al))
        by FO_LANG3:2
      .= { l.j : 1 <= j & j <= len l & l.j in bound_FO-variables(Al2) }
        by A11,FO_LANG3:def 1
      .= variables_in(l2,bound_FO-variables(Al2)) by A4,FO_LANG3:def 1
      .= still_not-bound_in l2 by FO_LANG3:2;
A15:  vp|still_not-bound_in l
       = v2|(bound_FO-variables(Al) /\ still_not-bound_in l) by A0,RELAT_1:71
      .= v2|still_not-bound_in l by XBOOLE_1:28;
      v2*'l2 = l*(vp|still_not-bound_in l) by A4,A14,A15,SUBLEM_2:59
      .= vp*'l by SUBLEM_2:59;
      hence thesis;
    end;
A16:J2,v2 |= Al2-Cast(P!l) implies Jp,vp |= P!l
    proof
      assume J2,v2 |= Al2-Cast(P!l);
      then J2,v2 |= P2!l2 by A3,A4,A6,FO_LANG1:def 12;
      then Valid(P2!l2,J2).v2 = TRUE by FO_VALUA:def 7;
      then (l2 'in' (J2.P2)).v2 = TRUE by FO_VALUA:6;
      then
A17:  vp*'l in (J2.P2) by A7,FO_VALUA:def 4;
      vp*'l in (Jp.P) by FUNCT_1:49,A0,A4,A17;
      then (l 'in' (Jp.P)).vp = TRUE by FO_VALUA:def 4;
      then Valid(P!l,Jp).vp = TRUE by FO_VALUA:6;
      hence thesis by FO_VALUA:def 7;
    end;
    (not J2,v2 |= Al2-Cast(P!l)) implies (not Jp,vp |= P!l)
    proof
      assume not J2,v2 |= Al2-Cast(P!l);
      then not J2,v2 |= P2!l2 by A3,A4,A6,FO_LANG1:def 12;
      then not Valid(P2!l2,J2).v2 = TRUE by FO_VALUA:def 7;
      then not (l2 'in' (J2.P2)).v2 = TRUE by FO_VALUA:6;
      then
A18:  not vp*'l in (J2.P2) by A7,FO_VALUA:def 4;
      not vp*'l in (Jp.P) by FUNCT_1:49,A0,A4,A18;
      then not (l 'in' (Jp.P)).vp = TRUE by FO_VALUA:def 4;
      then not Valid(P!l,Jp).vp = TRUE by FO_VALUA:6;
      hence thesis by FO_VALUA:def 7;
    end;
    hence thesis by A16;
  end;
T3: for p holds T[p] implies T['not' p]
  proof
    let p;
    assume
A19:T[p];
    let J2, Jp, v2, vp;
    assume
A20:Jp = J2|FO-pred_symbols(Al) & vp = v2|bound_FO-variables(Al);
    per cases;
    suppose
A21:  not J2,v2 |= Al2-Cast(p);
      then
A22:  not Jp,vp |= p by A19,A20;
      J2,v2 |= 'not' (Al2-Cast(p)) by A21, FO_VALUA:17;
      hence thesis by A22,Th8,FO_VALUA:17;
    end;
    suppose
A23:  J2,v2 |= Al2-Cast(p);
      then
A24:  Jp,vp |= p by A19,A20;
      not J2,v2 |= 'not' (Al2-Cast(p)) by FO_VALUA:17, A23;
      hence thesis by A24,FO_VALUA:17,Th8;
    end;
  end;
T4: for p,r holds (T[p] & T[r]) implies T[p '&' r]
  proof
    let p,r;
    assume
A25:T[p] & T[r];
    let J2, Jp, v2, vp;
    assume
A26: Jp = J2|FO-pred_symbols(Al) & vp = v2|bound_FO-variables(Al);
A27:J2,v2 |= (Al2-Cast(p '&' r)) implies Jp,vp |= p '&' r
    proof
      assume J2,v2 |= Al2-Cast(p '&' r);
      then J2,v2 |= (Al2-Cast(p)) '&' (Al2-Cast(r)) by Th8;
      then J2,v2 |= Al2-Cast(p) & J2,v2 |= Al2-Cast(r) by FO_VALUA:18;
      then Jp,vp |= p & Jp,vp |= r by A25, A26;
      hence Jp,vp |= p '&' r by FO_VALUA:18;
    end;
    Jp,vp |= p '&' r implies J2,v2 |= (Al2-Cast(p '&' r))
    proof
      assume Jp,vp |= p '&' r;
      then Jp,vp |= p & Jp,vp |= r by FO_VALUA:18;
      then J2,v2 |= Al2-Cast(p) & J2,v2 |= Al2-Cast(r) by A25,A26;
      then J2,v2 |= (Al2-Cast(p)) '&' (Al2-Cast(r)) by FO_VALUA:18;
      hence J2,v2 |= Al2-Cast(p '&' r) by Th8;
    end;
    hence thesis by A27;
  end;
T5: for x,r holds T[r] implies T[All(x,r)]
  proof
    let x,r;
    assume
A28:T[r];
    let J2, Jp, v2, vp;
    assume
A29: Jp = J2|FO-pred_symbols(Al) & vp = v2|bound_FO-variables(Al);
A30:J2,v2 |= Al2-Cast(All(x,r)) implies Jp,vp |= All(x,r)
    proof
      assume J2,v2 |= Al2-Cast(All(x,r));
      then
A31:  J2,v2 |= All(Al2-Cast(x),Al2-Cast(r)) by Th8;
      for vp1 being Element of Valuations_in(Al,A) st
           for y being bound_FO-variable of Al st x <> y holds vp1.y = vp.y
           holds Jp,vp1 |= r
      proof
        let vp1 be Element of Valuations_in(Al,A) such that
A32:    for y being bound_FO-variable of Al st x <> y holds vp1.y = vp.y;
        set s = Al2-Cast(x) .--> vp1.x;
A33:    s = {Al2-Cast(x)} --> vp1.x by FUNCOP_1:def 9;
        set v21 = v2 +* s;
        v2 is Element of Funcs(bound_FO-variables(Al2),A) by FO_VALUA:def 1;
        then
A34:    dom v2 = bound_FO-variables(Al2) & rng v2 c= A by FUNCT_2:92;
        dom s = {Al2-Cast(x)} by A33,FUNCOP_1:13;
        then dom v21 = dom v2 \/ {Al2-Cast(x)} by FUNCT_4:def 1;
        then
A36:    dom v21 = bound_FO-variables(Al2) by A34,XBOOLE_1:12;
A37:    rng v2 \/ {vp1.x} c= A by A34, XBOOLE_1:8;
        rng s = {vp1.x} by A33, FUNCOP_1:8;
        then rng v21 c= rng v2 \/ {vp1.x} by FUNCT_4:17;
        then rng v21 c= A by A37,XBOOLE_1:1;
        then v21 is Element of Funcs(bound_FO-variables(Al2),A)
             by A36, FUNCT_2:def 2;
        then reconsider v21 as Element of Valuations_in(Al2,A)
         by FO_VALUA:def 1;
        for y being bound_FO-variable of Al2 st Al2-Cast(x) <> y
           holds v21.y = v2.y by FUNCT_4:83;
        then
A39:    J2,v21 |= Al2-Cast(r) by A31,FO_VALUA:29;
        vp1 is Element of Funcs(bound_FO-variables(Al),A) by FO_VALUA:def 1;
        then
A40:    dom vp1 = bound_FO-variables(Al) by FUNCT_2:92
              .= (dom v21) /\ bound_FO-variables(Al) by A36, Th4, XBOOLE_1:28;
        for c being set st c in dom vp1 holds vp1.c = v21.c
        proof
          let c be set such that
A41:      c in dom vp1;
          per cases;
          suppose
A42:        c = Al2-Cast(x);
            then c in dom s by FUNCOP_1:74;
            hence v21.c = s.c by FUNCT_4:13
                 .= vp1.c by A42,FUNCOP_1:72;
          end;
          suppose
A43:        c <> Al2-Cast(x);
            reconsider c as bound_FO-variable of Al by A40,A41,XBOOLE_0:def 4;
            v21.c = v2.c by A43,FUNCT_4:83
                     .= (v2|bound_FO-variables(Al)).c by FUNCT_1:49
                     .= vp1.c by A29,A32,A43;
            hence thesis;
          end;
        end;
        then v21|bound_FO-variables(Al) = vp1 by FUNCT_1:46, A40;
        hence Jp,vp1 |= r by A29,A28,A39;
      end;
      hence Jp,vp |= All(x,r) by FO_VALUA:29;
    end;
    Jp,vp |= All(x,r) implies J2,v2 |= Al2-Cast(All(x,r))
    proof
      assume
A45:   Jp,vp |= All(x,r);
      for v21 being Element of Valuations_in(Al2,A) st
           for y being bound_FO-variable of Al2 st Al2-Cast(x) <> y holds
           v21.y = v2.y holds J2,v21 |= Al2-Cast(r)
      proof
        let v21 be Element of Valuations_in(Al2,A) such that
A46:    for y being bound_FO-variable of Al2 st Al2-Cast(x) <> y holds
              v21.y = v2.y;
        set s = x .--> v21.(Al2-Cast(x));
A47:    s = {x} --> v21.(Al2-Cast(x)) by FUNCOP_1:def 9;
        set vp1 = vp +* s;
        vp is Element of Funcs(bound_FO-variables(Al),A) by FO_VALUA:def 1;
        then
A48:    dom vp = bound_FO-variables(Al) & rng vp c= A by FUNCT_2:92;
        dom s = {x} by A47,FUNCOP_1:13;
        then dom vp1 = dom vp \/ {x} by FUNCT_4:def 1;
        then
A51:    dom vp1 = bound_FO-variables(Al) by A48,XBOOLE_1:12;
A52:    rng vp \/ {v21.(Al2-Cast(x))} c= A by A48, XBOOLE_1:8;
        rng s = {v21.(Al2-Cast(x))} by A47, FUNCOP_1:8;
        then rng vp1 c= rng vp \/ {v21.(Al2-Cast(x))} by FUNCT_4:17;
        then rng vp1 c= A by A52, XBOOLE_1:1;
        then vp1 is Element of Funcs(bound_FO-variables(Al),A)
             by A51, FUNCT_2:def 2;
        then reconsider vp1 as Element of Valuations_in(Al,A)
         by FO_VALUA:def 1;
        for y being bound_FO-variable of Al st x <> y
           holds vp1.y = vp.y by FUNCT_4:83;
        then
A54:    Jp,vp1 |= r by A45,FO_VALUA:29;
        v21 is Element of Funcs(bound_FO-variables(Al2),A) by FO_VALUA:def 1;
        then
A55:    dom v21 = bound_FO-variables(Al2) by FUNCT_2:92;
        vp1 is Element of Funcs(bound_FO-variables(Al),A) by FO_VALUA:def 1;
        then
A56:    dom vp1 = bound_FO-variables(Al) by FUNCT_2:92
            .= (dom v21) /\ bound_FO-variables(Al) by A55, Th4, XBOOLE_1:28;
        for c being set st c in dom vp1 holds vp1.c = v21.c
        proof
          let c be set such that
A57:      c in dom vp1;
          per cases;
          suppose
A58:        c = x;
            then c in dom s by FUNCOP_1:74;
            then vp1.c = s.x by A58,FUNCT_4:13 .= v21.c by A58,FUNCOP_1:72;
            hence vp1.c = v21.c;
          end;
          suppose
A59:        c <> x;
A60:        c in bound_FO-variables(Al) by A56,A57,XBOOLE_0:def 4;
            vp1 is Element of Funcs(bound_FO-variables(Al),A)
             by FO_VALUA:def 1;
            then dom vp1 = bound_FO-variables(Al) by FUNCT_2:92;
            then
A61:         dom vp1 c= bound_FO-variables(Al2) by Th4;
            vp1.c = vp.c by A59,FUNCT_4:83
                     .= v2.c by A29,A60,FUNCT_1:49
                     .= v21.c by A59,A46,A57,A61;
            hence thesis;
          end;
        end;
        then v21|bound_FO-variables(Al) = vp1 by FUNCT_1:46, A56;
        hence J2,v21 |= Al2-Cast(r) by A29,A28,A54;
      end;
      then J2,v2 |= All(Al2-Cast(x),Al2-Cast(r)) by FO_VALUA:29;
      hence J2,v2 |= Al2-Cast(All(x,r)) by Th8;
    end;
    hence thesis by A30;
  end;
T6: for r,s being Element of CFO-WFF(Al)
  for x being bound_FO-variable of Al for k
  for l being CFO-variable_list of k, Al for P being
  FO-pred_symbol of k,Al holds T[VERUM(Al)] & T[P!l] &
  (T[r] implies T['not' r]) & (T[r] & T[s] implies T[r '&' s]) &
  (T[r] implies T[All(x, r)]) by T1,T2,T3,T4,T5;
  for r being Element of CFO-WFF(Al) holds T[r] from CFO_LANG:sch 1(T6);
  hence thesis;
end;

theorem
  for Al2 being Al-extending FO-alphabet,
  THETA being Subset of CFO-WFF(Al2) st PHI c= THETA holds
   for A2 being non empty set, J2 being interpretation of Al2,A2,
   v2 being Element of Valuations_in (Al2,A2) st J2,v2 |= THETA holds
   ex A,J,v st J,v |= PHI
proof
  let Al2 be Al-extending FO-alphabet,
  THETA be Subset of CFO-WFF(Al2) such that
A0: PHI c= THETA;
  let A2 be non empty set, J2 be interpretation of Al2,A2,
      v2 be Element of Valuations_in(Al2,A2) such that
A1:  J2,v2 |= THETA;
  set A = A2;
A2: FO-pred_symbols(Al) c= FO-pred_symbols(Al2) by Th3;
  set Jp = J2|FO-pred_symbols(Al);
  reconsider Jp as Function of FO-pred_symbols(Al),relations_on A
   by A2,FUNCT_2:32;
  for P being Element of FO-pred_symbols(Al),
  r being Element of relations_on A st Jp.P = r holds
  r = empty_rel(A) or the_arity_of P = the_arity_of r
  proof
    let P be Element of FO-pred_symbols(Al),
        r be Element of relations_on A such that
A6:     Jp.P = r;
    P is Element of FO-pred_symbols(Al2) by A2,TARSKI:def 3;
    then consider Q being Element of FO-pred_symbols(Al2) such that
A7: P = Q;
A9: P`1 = 7+the_arity_of P & Q`1 = 7+the_arity_of Q by FO_LANG1:def 8;
    Jp.P = J2.Q by A7,FUNCT_1:49;
    hence thesis by A6,A7,A9,FO_VALUA:def 5;
  end;
  then reconsider Jp as interpretation of Al,A by FO_VALUA:def 5;
A10: bound_FO-variables(Al) c= bound_FO-variables(Al2) by Th4;
  set vp = v2|bound_FO-variables(Al);
  reconsider vp as Function of bound_FO-variables(Al),A
   by A10, FUNCT_2:32;
A12: Funcs(bound_FO-variables(Al),A) = Valuations_in(Al,A) by FO_VALUA:def 1;
  reconsider vp as Element of Valuations_in(Al,A) by A12,FUNCT_2:8;
  for r being Element of CFO-WFF(Al) holds
  r in PHI implies Jp,vp |= r
  proof
    let r be Element of CFO-WFF(Al) such that
A15: r in PHI;
    J2,v2 |= Al2-Cast(r) by A0,A1,A15,CALCL1_2:def 11;
    hence thesis by Th9;
  end;
  then Jp,vp |= PHI by CALCL1_2:def 11;
  hence thesis;
end;

theorem Th11:
 for f being FinSequence of CFO-WFF(Al2),g being FinSequence of CFO-WFF(Al)
 st f=g holds Ant f = Ant g & Suc f = Suc g
proof
 let f be FinSequence of CFO-WFF(Al2),g be FinSequence of CFO-WFF(Al) such that
A1: f = g;
  per cases;
  suppose
A3: len f > 0;
    then consider k being Nat such that
A4:  len f = k + 1 by NAT_1:6;
    reconsider k as Element of NAT by ORDINAL1:def 12;
    thus Ant f = g|(Seg k) by A1,A3,A4,CALCL1_2:def 1
              .= Ant g by A1,A3,A4,CALCL1_2:def 1;
    Suc f = g.(len g) by A1,A3,CALCL1_2:def 2
         .= Suc g by A1,A3,CALCL1_2:def 2;
    hence thesis;
  end;
  suppose
A5:  not len f > 0;
    thus Ant f = {} by A5,CALCL1_2:def 1
              .= Ant g by A1,A5,CALCL1_2:def 1;
    thus Suc f = VERUM(Al2) by A5, CALCL1_2:def 2
              .= <*[0,0]*> by FO_LANG1:def 14
              .= VERUM(Al) by FO_LANG1:def 14
              .= Suc g by A1,A5, CALCL1_2:def 2;
  end;
end;
```

```
theorem Th12:
 for i,j being Element of NAT, n being Nat st i < j & j < n holds i < n
proof
  let i,j be Element of NAT, n be Nat such that
A1:  i < j & j < n;
  i < n+1
  proof
    set k = n - j;
    reconsider k as Element of NAT by A1,NAT_1:21;
    n = j + k;
    then i <= n by A1,NAT_1:12;
    hence thesis by NAT_1:13;
  end;
  hence thesis by A1,NAT_1:22;
end;

theorem Th13:
 for p holds still_not-bound_in p = still_not-bound_in (Al2-Cast(p))
proof
  defpred A[Element of CFO-WFF(Al)] means still_not-bound_in $1 =
   still_not-bound_in (Al2-Cast($1));
A1: A[VERUM(Al)]
  proof
    thus still_not-bound_in VERUM(Al) = {} by FO_LANG3:3
        .= still_not-bound_in VERUM(Al2) by FO_LANG3:3
        .= still_not-bound_in (Al2-Cast(VERUM(Al))) by Th8;
  end;
A2: for k,P,l holds A[P!l]
  proof
   let k,P,l;
A3: still_not-bound_in l = still_not-bound_in Al2-Cast(l)
   proof
     for x being set st x in still_not-bound_in l holds
      x in still_not-bound_in Al2-Cast(l)
     proof
       let x be set such that
A4:     x in still_not-bound_in l;
       x in variables_in(l,bound_FO-variables(Al)) by A4,FO_LANG3:2;
       then x in {l.n: 1 <= n & n <= len l & l.n in bound_FO-variables(Al)}
         by FO_LANG3:def 1;
       then consider n such that
A5:     x = l.n & 1 <= n & n <= len l & l.n in bound_FO-variables(Al);
       set y = l.n;
       reconsider y as bound_FO-variable of Al by A5;
       y = Al2-Cast(y);
       then y in {Al2-Cast(l).j: 1 <= j & j <= len (Al2-Cast(l)) &
        Al2-Cast(l).j in bound_FO-variables(Al2)} by A5;
       then x in variables_in(Al2-Cast(l),bound_FO-variables(Al2))
        by A5,FO_LANG3:def 1;
       hence thesis by FO_LANG3:2;
     end;
     hence still_not-bound_in l c= still_not-bound_in Al2-Cast(l)
       by TARSKI:def 3;
     for x being set st x in still_not-bound_in Al2-Cast(l) holds
      x in still_not-bound_in l
     proof
       let x be set such that
A6:     x in still_not-bound_in Al2-Cast(l);
       x in variables_in(Al2-Cast(l),bound_FO-variables(Al2)) by A6,FO_LANG3:2;
       then x in {Al2-Cast(l).n: 1 <= n & n <= len (Al2-Cast(l)) &
                   Al2-Cast(l).n in bound_FO-variables(Al2)} by FO_LANG3:def 1;
       then consider n such that
A7:     x = Al2-Cast(l).n & 1 <= n & n <= len (Al2-Cast(l)) &
        Al2-Cast(l).n in bound_FO-variables(Al2);
       set y = Al2-Cast(l).n;
       rng l c= bound_FO-variables(Al) & dom l = Seg (len l) by FINSEQ_1:def 3;
       then y = l.n & n in dom l & l is FinSequence of bound_FO-variables(Al)
        by A7,FINSEQ_1:1,def 4;
       then y in bound_FO-variables(Al) by FINSEQ_2:11;
       then y in  {l.j: 1 <= j & j <= len l & l.j in bound_FO-variables(Al)}
         by A7;
       then x in variables_in(l,bound_FO-variables(Al)) by A7,FO_LANG3:def 1;
       hence thesis by FO_LANG3:2;
     end;
     hence still_not-bound_in Al2-Cast(l) c= still_not-bound_in l
        by TARSKI:def 3;
   end;
   thus still_not-bound_in (P!l) = still_not-bound_in l by FO_LANG3:5
     .= still_not-bound_in (Al2-Cast(P)!Al2-Cast(l)) by A3,FO_LANG3:5
     .= still_not-bound_in Al2-Cast(P!l) by Th8;
  end;
A8: for p holds A[p] implies A['not' p]
    proof
```

```
       let p such that
A9:     A[p];
       thus still_not-bound_in 'not' p
         = still_not-bound_in p by FO_LANG3:7
        .= still_not-bound_in 'not' Al2-Cast(p) by A9,FO_LANG3:7
        .= still_not-bound_in Al2-Cast('not' p) by Th8;
    end;
A12: for p,q holds A[p] & A[q] implies A[p '&' q]
    proof
       let p,q such that
A13:     A[p] & A[q];
       set p2 = Al2-Cast(p);
       set q2 = Al2-Cast(q);
       reconsider p2,q2 as Element of CFO-WFF(Al2);
A14:   p '&' q is conjunctive & p2 '&' q2 is conjunctive by FO_LANG1:def 20;
       then
A15:   p = the_left_argument_of p '&' q & q = the_right_argument_of p '&' q &
        p2 = the_left_argument_of p2 '&' q2 &
        q2 = the_right_argument_of p2 '&' q2 by FO_LANG1:def 25,def 26;
       hence still_not-bound_in p '&' q
         = still_not-bound_in p \/ still_not-bound_in q by A14,FO_LANG3:9
        .= still_not-bound_in p2 '&' q2 by A13,A14,A15,FO_LANG3:9
        .= still_not-bound_in Al2-Cast(p '&' q) by Th8;
    end;
  for x,p holds A[p] implies A[All(x,p)]
  proof
    let x,p such that
A16: A[p];
    set x2 = Al2-Cast(x);
    set p2 = Al2-Cast(p);
    reconsider p2 as Element of CFO-WFF(Al2);
    reconsider x2 as bound_FO-variable of Al2;
A17: All(x,p) is universal & All(x2,p2) is universal by FO_LANG1:def 21;
    then
A18: p = the_scope_of All(x,p) & x = bound_in All(x,p) & p2 = the_scope_of
      All(x2,p2) & x2 = bound_in All(x2,p2) by FO_LANG1:def 27, def 28;
    hence still_not-bound_in All(x,p)
      = still_not-bound_in p \ {x} by A17,FO_LANG3:11
     .= still_not-bound_in All(x2,p2) by A16,A17,A18,FO_LANG3:11
     .= still_not-bound_in Al2-Cast(All(x,p)) by Th8;
  end;
  then
A19: for p, q, x, k, l, P holds A[VERUM(Al)] & A[P!l] &
     (A[p] implies A['not' p]) & (A[p] & A[q] implies A[p '&' q]) &
     (A[p] implies A[All(x, p)]) by A1,A2,A8,A12;
  for p holds A[p] from CFO_LANG:sch 1(A19);
  hence thesis;
end;

theorem Th14:
 for p2 being Element of CFO-WFF(Al2), S being CFO_Substitution of Al,
 S2 being CFO_Substitution of Al2, x2 being bound_FO-variable of Al2, x, p
 st p = p2 & S = S2 & x = x2 holds RestrictSub(x,p,S) = RestrictSub(x2,p2,S2)
proof
  let p2 be Element of CFO-WFF(Al2), S be CFO_Substitution of Al, S2 be
   CFO_Substitution of Al2, x2 be bound_FO-variable of Al2, x, p such that
A1: p = p2 & S = S2 & x = x2;
  set rset = {y where y is bound_FO-variable of Al : y in still_not-bound_in p
   & y is Element of dom S & y <> x & y <> S.y};
  set rset2 = {y2 where y2 is bound_FO-variable of Al2 : y2 in
   still_not-bound_in p2 & y2 is Element of dom S2 & y2 <> x2 & y2 <> S2.y2};
  rset = rset2
  proof
    for s being set st s in rset holds s in rset2
    proof
      let s be set such that
A2:    s in rset;
      consider y being bound_FO-variable of Al such that
A3:    s = y & y in still_not-bound_in p & y is Element of dom S & y <> x &
       y <> S.y by A2;
      bound_FO-variables(Al) c= bound_FO-variables(Al2) by Th4;
      then reconsider y as bound_FO-variable of Al2 by TARSKI:def 3;
      p2 = Al2-Cast(p) by A1;
      then y in still_not-bound_in p2 & y is Element of dom S2 & y <> x2 &
       y <> S2.y by A1,A3,Th13;
      hence s in rset2 by  A3;
    end;
    hence rset c= rset2 by TARSKI:def 3;
    for s2 being set st s2 in rset2 holds s2 in rset
    proof
      let s2 be set such that
A4:    s2 in rset2;
      consider y2 being bound_FO-variable of Al2 such that
A5:    s2 = y2 & y2 in still_not-bound_in p2 & y2 is Element of dom S2 &
```

```
       y2 <> x2 & y2 <> S2.y2 by A4;
      p2 = Al2-Cast(p) by A1;
      then
A6:   y2 in still_not-bound_in p by A5,Th13;
      then reconsider y2 as bound_FO-variable of Al;
      thus s2 in rset by A1,A5,A6;
    end;
    hence rset2 c= rset by TARSKI:def 3;
  end;
  then S|rset = S2|rset2 & S|rset = RestrictSub(x,p,S) &
   S2|rset2 = RestrictSub(x2,p2,S2) by A1,SUBST1_2:def 6;
  hence thesis;
end;

theorem Th15:
 for p2 being Element of CFO-WFF(Al2), S being finite CFO_Substitution of Al,
 S2 being finite CFO_Substitution of Al2, p st S = S2 & p = p2 holds
 upVar(S,p) = upVar(S2,p2)
proof
  let p2 be Element of CFO-WFF(Al2), S be finite CFO_Substitution of
   Al, S2 being finite CFO_Substitution of Al2, p such that
A1: S = S2 & p = p2;
A2: Sub_Var(S) = Sub_Var(S2)
  proof
    for s being set st s in Sub_Var(S) holds s in Sub_Var(S2)
    proof
      let s be set such that
A3:    s in Sub_Var(S);
      s in {t where t is FO-symbol of Al : x.t in rng S} by A3,SUBST1_2:def 10;
      then consider s2 being FO-symbol of Al such that
A5:    s = s2 & x.s2 in rng S;
      s2 in FO-symbols(Al) & FO-symbols(Al) c= FO-symbols(Al2) by Th2;
      then consider s3 being FO-symbol of Al2 such that
A6:    s3 = s2;
      x.s2 =[4,s2] by FO_LANG3:def 2 .= x.s3 by A6,FO_LANG3:def 2;
      then s3 in {t where t is FO-symbol of Al2 : x.t in rng S2} by A1,A5;
      hence thesis by A5,A6,SUBST1_2:def 10;
    end;
    hence Sub_Var(S) c= Sub_Var(S2) by TARSKI:def 3;
    for s being set st s in Sub_Var(S2) holds s in Sub_Var(S)
    proof
      let s be set such that
A8:    s in Sub_Var(S2);
      s in {t where t is FO-symbol of Al2:x.t in rng S2} by A8,SUBST1_2:def 10;
      then consider s2 being FO-symbol of Al2 such that
A10:   s = s2 & x.s2 in rng S2;
A0:   rng @S c= bound_FO-variables(Al) by SUBST1_2:39;
      x.s2 in rng @S by A1,A10,SUBST1_2:def 2;
      then x.s2 in bound_FO-variables(Al) by A0;
      then [4,s2] in bound_FO-variables(Al) by FO_LANG3:def 2;
      then [4,s2] in [:{4},FO-symbols(Al):] by FO_LANG1:def 4;
      then s2 in FO-symbols(Al) by ZFMISC_1:87;
      then consider s3 being FO-symbol of Al such that
A11:   s3 = s2;
      x.s2 =[4,s2] by FO_LANG3:def 2.= x.s3 by A11,FO_LANG3:def 2;
      then s3 in {t where t is FO-symbol of Al : x.t in rng S} by A1,A10;
      hence thesis by A10,A11,SUBST1_2:def 10;
    end;
    hence Sub_Var(S2) c= Sub_Var(S) by TARSKI:def 3;
  end;
  defpred P[Element of FO-WFF(Al)] means for q2 being Element of CFO-WFF(Al2)
   st $1 = q2 holds Bound_Vars($1) = Bound_Vars(q2);
A13: P[VERUM(Al)]
  proof
    let q2 be Element of CFO-WFF(Al2) such that
A14: q2 = VERUM(Al);
    q2 = Al2-Cast(VERUM(Al)) by A14 .= VERUM(Al2) by Th8;
    hence Bound_Vars(q2)={} by SUBST1_2:2.=Bound_Vars(VERUM(Al)) by SUBST1_2:2;
  end;
A15: for k,P,l holds P[P!l]
  proof
    let k,P,l;
    set P2 = Al2-Cast(P);
    set l2 = Al2-Cast(l);
    let q2 be Element of CFO-WFF(Al2) such that
A16: P!l = q2;
A17: q2 = Al2-Cast(P!l) by A16 .= P2!l2 by Th8;
    thus Bound_Vars(P!l) = still_not-bound_in (P!l) by SUBLEM_2:43
     .= still_not-bound_in (Al2-Cast(P!l)) by Th13
     .= still_not-bound_in (P2!l2) by Th8
     .= Bound_Vars(q2) by A17,SUBLEM_2:43;
  end;
A18: for r,s st P[r] & P[s] holds P[r '&' s]
  proof
```

```
    let r,s such that
A19: P[r] & P[s];
    set q = r '&' s;
    set r2 = Al2-Cast(r);
    set s2 = Al2-Cast(s);
    let q2 be Element of CFO-WFF(Al2) such that
A20: r '&' s = q2;
A21: q2 = Al2-Cast(r '&' s) by A20 .= r2 '&' s2 by Th8;
    then
A22: q is conjunctive & q2 is conjunctive by FO_LANG1:def 20;
    then
A23: the_left_argument_of q = r & the_right_argument_of q = s &
    the_left_argument_of q2 = r2 & the_right_argument_of q2 = s2
    by A21,FO_LANG1:def 25, def 26;
A24: Bound_Vars(r) = Bound_Vars(r2) & Bound_Vars(s) = Bound_Vars(s2) by A19;
    thus Bound_Vars(q) = Bound_Vars(r) \/ Bound_Vars(s) by A22,A23,SUBST1_2:5
     .= Bound_Vars(q2) by A22,A23,A24,SUBST1_2:5;
  end;
A25: for r st P[r] holds P['not' r]
  proof
    let r such that
A26: P[r];
    set q = 'not' r;
    set r2 = Al2-Cast(r);
    let q2 be Element of CFO-WFF(Al2) such that
A27: q = q2;
A28: q2 = Al2-Cast('not' r) by A27 .= 'not' r2 by Th8;
    then
A29: q is negative & q2 is negative by FO_LANG1:def 19;
    then
A30: the_argument_of q = r & the_argument_of q2 = r2 by A28,FO_LANG1:def 24;
    thus Bound_Vars(q) = Bound_Vars(r) by A29,A30,SUBST1_2:4
     .= Bound_Vars(r2) by A26 .= Bound_Vars(q2) by A29,A30,SUBST1_2:4;
  end;
A31: for x,r st P[r] holds P[All(x,r)]
  proof
    let x,r such that
A32: P[r];
    set q = All(x,r);
    set r2 = Al2-Cast(r);
    set x2 = Al2-Cast(x);
    let q2 be Element of CFO-WFF(Al2) such that
A33: q = q2;
A34: q2 = Al2-Cast(All(x,r)) by A33 .= All(x2,r2) by Th8;
    then
A35: q is universal & q2 is universal by FO_LANG1:def 21;
    then
A36: the_scope_of q = r & bound_in q = x & the_scope_of q2 = r2 &
     bound_in q2 = x2 by A34,FO_LANG1:def 27,def 28;
    thus Bound_Vars(q) = Bound_Vars(r) \/ {x} by A35,A36,SUBST1_2:6
     .= Bound_Vars(r2) \/ {x2} by A32 .= Bound_Vars(q2) by A35,A36,SUBST1_2:6;
  end;
A37: for r,s,x,k,l,P holds P[VERUM(Al)] & P[P!l] & (P[r] implies P['not' r]) &
  (P[r] & P[s] implies P[r '&' s]) & (P[r] implies P[All(x, r)])
   by A13,A15,A18,A25,A31;
A38: for q being Element of CFO-WFF(Al) holds P[q] from CFO_LANG:sch 1(A37);
A39: Dom_Bound_Vars p = Dom_Bound_Vars p2
  proof
    for s being set st s in Dom_Bound_Vars p holds s in Dom_Bound_Vars p2
    proof
      let s be set such that
A40:   s in Dom_Bound_Vars p;
      s in {b where b is FO-symbol of Al : x.b in Bound_Vars p}
       by A40,SUBST1_2:def 9;
      then consider s2 being FO-symbol of Al such that
A41:   s = s2 & x.s2 in Bound_Vars p;
      x.s2 in Bound_Vars p2 by A1,A38,A41;
      then x.s2 in bound_FO-variables(Al2);
      then [4,s2] in bound_FO-variables(Al2) by FO_LANG3:def 2;
      then [4,s2] in [:{4},FO-symbols(Al2):] by FO_LANG1:def 4;
      then s2 in FO-symbols(Al2) by ZFMISC_1:87;
      then consider s3 being FO-symbol of Al2 such that
A43:   s3 = s2;
      x.s2 =[4,s2] by FO_LANG3:def 2 .= x.s3 by A43,FO_LANG3:def 2;
      then x.s3 in Bound_Vars p2 by A1,A38,A41;
      then s3 in {b where b is FO-symbol of Al2 : x.b in Bound_Vars p2};
      hence thesis by A41,A43,SUBST1_2:def 9;
    end;
    hence Dom_Bound_Vars p c= Dom_Bound_Vars p2 by TARSKI:def 3;
    for s being set st s in Dom_Bound_Vars p2 holds s in Dom_Bound_Vars p
    proof
      let s be set such that
A44:   s in Dom_Bound_Vars p2;
      s in {b where b is FO-symbol of Al2 : x.b in Bound_Vars p2}
```

```
       by A44,SUBST1_2:def 9;
      then consider s2 being FO-symbol of Al2 such that
A45:   s = s2 & x.s2 in Bound_Vars p2;
      x.s2 in Bound_Vars p by A1,A38,A45;
      then x.s2 in bound_FO-variables(Al);
      then [4,s2] in bound_FO-variables(Al) by FO_LANG3:def 2;
      then [4,s2] in [:{4},FO-symbols(Al):] by FO_LANG1:def 4;
      then s2 in FO-symbols(Al) by ZFMISC_1:87;
      then consider s3 being FO-symbol of Al such that
A47:   s3 = s2;
      x.s2 =[4,s2] by FO_LANG3:def 2 .= x.s3 by A47,FO_LANG3:def 2;
      then x.s3 in Bound_Vars p by A1,A38,A45;
      then s3 in {b where b is FO-symbol of Al : x.b in Bound_Vars p};
      hence thesis by A45,A47,SUBST1_2:def 9;
    end;
    hence Dom_Bound_Vars p2 c= Dom_Bound_Vars p by TARSKI:def 3;
  end;
A49: NSub(p,S) = NAT\(Dom_Bound_Vars(p)\/Sub_Var(S)) by SUBST1_2:def 11
   .= NSub(p2,S2) by A2,A39,SUBST1_2:def 11;
  thus upVar(S,p) = the Element of NSub(p,S) by SUBST1_2:def 12
   .= upVar(S2,p2) by A49,SUBST1_2:def 12;
end;

theorem Th16:
 for p2 being Element of CFO-WFF(Al2), S being CFO_Substitution of Al,
 S2 being CFO_Substitution of Al2, x2 being bound_FO-variable of Al2, x, p
 st p = p2 & S = S2 & x = x2 holds ExpandSub(x,p,RestrictSub(x,All(x,p),S))
 = ExpandSub(x2,p2,RestrictSub(x2,All(x2,p2),S2))
proof
  let p2 being Element of CFO-WFF(Al2), S being CFO_Substitution of Al,
   S2 being CFO_Substitution of Al2, x2 being bound_FO-variable of Al2, x, p
   such that
A1: p = p2 & S = S2 & x = x2;
  set rsub = RestrictSub(x,All(x,p),S);
  set rsub2 = RestrictSub(x2,All(x2,p2),S2);
  set esub = ExpandSub(x,p,rsub);
  set esub2 = ExpandSub(x2,p2,rsub2);
  set uv = upVar(rsub,p);
  set uv2 = upVar(rsub2,p2);
A2: All(x,p) = Al2-Cast(All(x,p)) .= All(Al2-Cast(x),Al2-Cast(p)) by Th8
    .= All(x2,p2) by A1;
A5: x.uv =[4,uv] by FO_LANG3:def 2 .= [4,uv2] by A1,A2,Th14,Th15
   .= x.uv2 by FO_LANG3:def 2;
  per cases;
  suppose
A6: not x in rng rsub;
    then not x2 in rng rsub2 by A1,A2,Th14;
    hence esub2 = rsub2 \/ {[x2,x2]} by SUBST1_2:def 13
     .= rsub \/ {[x,x]} by A1,A2,Th14 .= esub by A6,SUBST1_2:def 13;
  end;
  suppose
A7: x in rng rsub;
    then x2 in rng rsub2 by A1,A2,Th14;
    hence esub2 = rsub2 \/ {[x2,(x.uv2)]} by SUBST1_2:def 13
    .= rsub \/ {[x,(x.uv)]} by A1,A2,Th14,A5
    .= esub by A7,SUBST1_2:def 13;
  end;
end;

theorem Th17:
 for Z being Element of CFO-Sub-WFF(Al), Z2 being Element of CFO-Sub-WFF(Al2)
 st Z`1 is universal & Z2`1 is universal & bound_in Z`1 = bound_in Z2`1 &
 the_scope_of Z`1 = the_scope_of Z2`1 & Z = Z2 holds S_Bound(@Z) = S_Bound(@Z2)
proof
  let Z be Element of CFO-Sub-WFF(Al), Z2 be Element of CFO-Sub-WFF(Al2)
   such that
A1: Z`1 is universal & Z2`1 is universal & bound_in Z`1 = bound_in Z2`1 &
    the_scope_of Z`1 = the_scope_of Z2`1 & Z = Z2;
  set p = (@Z)`1;
  set p2 = (@Z2)`1;
  set S = (@Z)`2;
  set S2 = (@Z2)`2;
  set x = bound_in (@Z)`1;
  set x2 = bound_in (@Z2)`1;
  set q = the_scope_of (@Z)`1;
  set q2 = the_scope_of (@Z2)`1;
  reconsider p as Element of CFO-WFF(Al) by A1,SUBST1_2:def 35;
  reconsider p2 as Element of CFO-WFF(Al2) by A1,SUBST1_2:def 35;
  (@Z)`1 is universal by A1,SUBST1_2:def 35;
  then p = All(x,q) by FO_LANG2:6;
  then reconsider q as Element of CFO-WFF(Al) by CFO_LANG:13;
  (@Z2)`1 is universal by A1,SUBST1_2:def 35;
  then p2 = All(x2,q2) by FO_LANG2:6;
  then reconsider q2 as Element of CFO-WFF(Al2) by CFO_LANG:13;
  reconsider x as bound_FO-variable of Al;
  reconsider x2 as bound_FO-variable of Al2;
A2: p =Z`1 by SUBST1_2:def 35 .= p2 by A1,SUBST1_2:def 35;
A3: S =Z`2 by SUBST1_2:def 35 .= S2 by A1,SUBST1_2:def 35;
A4: x = bound_in Z`1 by SUBST1_2:def 35 .= x2 by A1,SUBST1_2:def 35;
A5: q = the_scope_of Z`1 by SUBST1_2:def 35 .= q2 by A1,SUBST1_2:def 35;
  set rsub = RestrictSub(x,p,S);
  set rsub2 = RestrictSub(x2,p2,S2);
  per cases;
  suppose
A8: not x in rng rsub;
    then not x2 in rng rsub2 by A2,A3,A4,Th14;
    hence S_Bound @Z2 = x2 by SUBST1_2:def 36
     .= S_Bound @Z by A4,A8,SUBST1_2:def 36;
  end;
  suppose
A9: x in rng rsub;
    then x2 in rng rsub2 by A2,A3,A4,Th14;
    then S_Bound @Z2 = x.(upVar(rsub2,q2)) by SUBST1_2:def 36
     .= [4,upVar(rsub2,q2)] by FO_LANG3:def 2
     .= [4,upVar(rsub,q)] by A5,A2,A3,A4,Th14,Th15
     .= x.(upVar(rsub,q)) by FO_LANG3:def 2
     .= S_Bound @Z by A9,SUBST1_2:def 36;
    hence thesis;
  end;
end;

theorem Th18:
  for p2 being Element of CFO-WFF(Al2),x2,y2 being bound_FO-variable of Al2,
  p,x,y st p=p2 & x=x2 & y=y2 holds p.(x,y) = p2.(x2,y2)
proof
  defpred P[Element of CFO-WFF(Al)] means for p2 being Element of CFO-WFF(Al2),
   S being CFO_Substitution of Al, S2 being CFO_Substitution of Al2
   st $1 = p2 & S = S2 holds CFO_Sub [$1,S] = CFO_Sub [p2,S2];
A1: P[VERUM(Al)]
  proof
    set p = VERUM(Al);
    let p2 be Element of CFO-WFF(Al2), S be CFO_Substitution of Al,
     S2 be CFO_Substitution of Al2 such that
A2:  p = p2 & S = S2;
A3: VERUM(Al2) = Al2-Cast(p) by Th8 .= p2 by A2;
A4: [p,S] is Al-Sub_VERUM & [p2,S2] is Al2-Sub_VERUM by A3,SUBST1_2:def 19;
    thus CFO_Sub [p,S] = Al2-Cast(p) by A4,SUBLEM_2:3
     .= VERUM(Al2) by Th8 .= CFO_Sub [p2,S2] by A4,SUBLEM_2:3;
  end;
A4: for k,P,l holds P[P!l]
  proof
    let k,P,l;
    set P2 = Al2-Cast(P);
    set l2 = Al2-Cast(l);
    reconsider P2 as FO-pred_symbol of k,Al2;
    reconsider l2 as CFO-variable_list of k,Al2;
    let p2 be Element of CFO-WFF(Al2), S be CFO_Substitution of Al,
     S2 be CFO_Substitution of Al2 such that
A5:  P!l = p2 & S = S2;
A6: p2 = Al2-Cast(P!l) by A5 .= P2!l2 by Th8;
    set p = P!l;
    reconsider p as Element of CFO-WFF(Al);
    set sub = CFO_Subst(l,S);
A7: sub = CFO_Subst(Al2-Cast(l),S2)
    proof
A8:   len sub = len l by SUBST1_2:def 3
       .= len CFO_Subst(Al2-Cast(l),S2) by SUBST1_2:def 3;
      for n being Nat st n in dom sub holds
       sub.n = (CFO_Subst(Al2-Cast(l),S2)).n
      proof
        let n be Nat such that
A9:     n in dom sub;
        n in Seg len sub by A9,FINSEQ_1:def 3;
        then 1 <= n & n <= len sub by FINSEQ_1:1;
        then
A10:    1 <= n & n <= len l by SUBST1_2:def 3;
        reconsider n as Element of NAT by ORDINAL1:def 12;
        per cases;
        suppose
A11:      l.n in dom S;
          sub.n = S.(l.n) by A10,A11,SUBST1_2:def 3
           .= CFO_Subst(Al2-Cast(l),S2).n by A5,A10,A11,SUBST1_2:def 3;
          hence thesis;
        end;
        suppose
A13:      not l.n in dom S;
          sub.n = l.n by A10,A13,SUBST1_2:def 3
           .= CFO_Subst(Al2-Cast(l),S2).n by A5,A10,A13,SUBST1_2:def 3;
          hence thesis;
        end;
      end;
      hence thesis by A8,FINSEQ_2:9;
    end;
    the_arity_of P = len l & the_arity_of P2 = len l2 by Th1;
    then
A15: [P!l,S] = Sub_P(P,l,S) & [P2!l2,S2] = Sub_P(P2,l2,S2)
     by SUBST1_2:def 18;
    P!l is atomic & P2!l2 is atomic by FO_LANG1:def 18;
    then
A17: P = the_pred_symbol_of P!l & P2 = the_pred_symbol_of P2!l2
     by FO_LANG1:def 22;
A18: [P!l,S]`1 = P!l & [P!l,S]`2 = S & [P2!l2,S2]`1 = P2!l2 & [P2!l2,S2]`2 = S2
     & Sub_the_arguments_of [P!l,S] = l & Sub_the_arguments_of [P2!l2,S2] = l2
     by A15,SUBST1_2:def 29,MCART_1:def 1,def 2;
    thus CFO_Sub [P!l,S] = Al2-Cast(P!CFO_Subst(l,S)) by A15,A17,A18,SUBLEM_2:6
     .= (Al2-Cast(P))!(Al2-Cast(CFO_Subst(l,S))) by Th8
     .= CFO_Sub [p2,S2] by A6,A7,A15,A17,A18,SUBLEM_2:6;
  end;
A19: for q st P[q] holds P['not' q]
  proof
    let q such that
A20: P[q];
    set p = 'not' q;
    reconsider p as Element of CFO-WFF(Al);
    set q2 = Al2-Cast(q);
    reconsider q2 as Element of CFO-WFF(Al2);
    let p2 be Element of CFO-WFF(Al2), S being CFO_Substitution of Al,
     S2 being CFO_Substitution of Al2 such that
A21:  'not' q = p2 & S = S2;
A22: [q,S]`1 = q & [q,S]`2 = S & [q2,S2]`1 = q2 & [q2,S2]`2 = S2
     by MCART_1:def 1,def 2;
    thus CFO_Sub ['not' q, S] = CFO_Sub Sub_not [q,S] by A22,SUBST1_2:def 20
    .= Al2-Cast('not' CFO_Sub [q,S]) by SUBST1_2:29
    .= 'not' Al2-Cast(CFO_Sub[q,S]) by Th8 .= 'not' CFO_Sub [q2,S2] by A20,A21
    .= CFO_Sub Sub_not [q2,S2] by SUBST1_2:29 .= CFO_Sub ['not' q2,S2]
     by A22,SUBST1_2:def 20
    .= CFO_Sub[Al2-Cast('not' q),S2] by Th8 .= CFO_Sub [p2,S2] by A21;
  end;
A23: for r,s st P[r] & P[s] holds P[r '&' s]
  proof
    let r,s such that
A24: P[r] & P[s];
    set r2 = Al2-Cast(r);
    set s2 = Al2-Cast(s);
    reconsider r2,s2 as Element of CFO-WFF(Al2);
    let p2 be Element of CFO-WFF(Al2), S be CFO_Substitution of Al,
    S2 be CFO_Substitution of Al2 such that
A25: r '&' s = p2 & S = S2;
A26: CFO_Sub[r,S] = CFO_Sub[r2,S2] & CFO_Sub[s,S] = CFO_Sub[s2,S2] by A24,A25;
A27: [r,S]`1 = r & [r,S]`2 = S & [s,S]`1 = s & [s,S]`2 = S & [r2,S2]`1 = r2 &
     [r2,S2]`2 = S2 & [s2,S2]`1 = s2 & [s2,S2]`2 = S2 by MCART_1:def 1,def 2;
    thus CFO_Sub[r '&' s,S] = CFO_Sub CFOSub_&([r,S],[s,S]) by SUBST2_2:19
     .= Al2-Cast((CFO_Sub [r,S]) '&' (CFO_Sub[s,S])) by A27,SUBLEM_2:23
     .= Al2-Cast(CFO_Sub [r,S]) '&' Al2-Cast(CFO_Sub[s,S]) by Th8
     .= CFO_Sub CFOSub_&([r2,S2],[s2,S2]) by A26,A27,SUBLEM_2:23
     .= CFO_Sub [r2 '&' s2,S2] by SUBST2_2:19
     .= CFO_Sub [Al2-Cast(r '&' s),S2] by Th8 .= CFO_Sub [p2,S2] by A25;
  end;
  for z being bound_FO-variable of Al,q st P[q] holds P[All(z,q)]
  proof
    let z be bound_FO-variable of Al,q such that
A28: P[q];
    set p = All(z,q);
    set q2 = Al2-Cast(q);
    set z2 = Al2-Cast(z);
    reconsider p as Element of CFO-WFF(Al);
    let p2 be Element of CFO-WFF(Al2), S be CFO_Substitution of Al,
     S2 be CFO_Substitution of Al2 such that
A29: All(z,q) = p2 & S = S2;
    set qsc = Qsc(q,z,S);
    set qsc2 = Qsc(q2,z2,S2);
    set sub = [All(z,q),S];
    set sub2 = [All(z2,q2),S2];
    set qscope = [q,(CFQ(Al)).sub];
    set qscope2 = [q2,(CFQ(Al2)).sub2];
A30: QScope(q,z,S) = [qscope,z] by SUBST2_2:def 3;
    reconsider qscope as Element of CFO-Sub-WFF(Al);
    reconsider qsc as second_Q_comp of [qscope,z] by SUBST2_2:def 3;
A31: QScope(q2,z2,S2) = [qscope2,z2] by SUBST2_2:def 3;
    reconsider qscope2 as Element of CFO-Sub-WFF(Al2);
    reconsider qsc2 as second_Q_comp of [qscope2,z2] by SUBST2_2:def 3;
A32: sub = CFOSub_All([qscope,z],qsc) & [qscope,z] is quantifiable &
```

```
     sub2 = CFOSub_All([qscope2,z2],qsc2) & [qscope2,z2] is quantifiable
     by A30,A31,SUBST2_2:22;
    set expandsub = ExpandSub(z,q,RestrictSub(z,All(z,q),S));
    set expandsub2 = ExpandSub(z2,q2,RestrictSub(z2,All(z2,q2),S2));
A35: All(z,q) is universal & All(z2,q2) is universal by FO_LANG1:def 21;
     then z = bound_in All(z,q) & q = the_scope_of All(z,q) &
      z2 = bound_in All(z2,q2) & q2 = the_scope_of All(z2,q2)
     by FO_LANG1:def 27, def 28;
    then All(z,q),S PQSub expandsub &
     All(z2,q2),S2 PQSub expandsub2 by A35,SUBST1_2:def 14;
    then [sub,expandsub] in QSub(Al) & [sub2,expandsub2] in QSub(Al2)
     by SUBST1_2:def 15;
    then [sub,expandsub] in (QSub(Al))|(CFO-Sub-WFF(Al)) &
     [sub2,expandsub2] in (QSub(Al2))|(CFO-Sub-WFF(Al2)) by RELAT_1:def 11;
    then
A36:[sub,expandsub] in CFQ(Al) & [sub2,expandsub2] in CFQ(Al2)
     by SUBST2_2:def 2;
    set scope = CFOSub_the_scope_of sub;
    set scope2 = CFOSub_the_scope_of sub2;
A38: bound_in sub`1 = bound_in All(z,q) by MCART_1:def 1
      .= z by A35,FO_LANG1:def 27
      .= bound_in All(z2,q2) by A35,FO_LANG1:def 27
      .= bound_in sub2`1 by MCART_1:def 1;
A39: the_scope_of sub`1 = the_scope_of All(z,q) by MCART_1:def 1
      .= q by A35,FO_LANG1:def 28
      .= the_scope_of All(z2,q2) by A35,FO_LANG1:def 28
      .= the_scope_of sub2`1 by MCART_1:def 1;
A40: sub`1 is universal & sub2`1 is universal by A35,MCART_1:def 1;
A41: expandsub = expandsub2 by A29,Th16;
A42: CFO_Sub qscope = CFO_Sub[q,expandsub] by A36,FUNCT_1:1
      .= CFO_Sub[q2,expandsub2] by A28,A41
      .= CFO_Sub qscope2 by A36,FUNCT_1:1;
A43: All(z,q) = Al2-Cast(All(z,q)) .= All(z2,q2) by Th8;
A44: sub2 = [Al2-Cast(All(z,q)),S2] by Th8 .= [p2,S2] by A29;
    CFO_Sub [p,S] = CFOQuant(sub, CFO_Sub scope) by A32,SUBLEM_2:27,28
     .= Quant(sub,CFO_Sub scope) by A32,SUBLEM_2:27,def 7
     .= All(S_Bound(@sub),CFO_Sub scope) by SUBST1_2:def 37
     .= Al2-Cast(All(S_Bound(@sub),CFO_Sub qscope)) by A32,SUBLEM_2:30
     .= All(Al2-Cast(S_Bound(@sub)),Al2-Cast(CFO_Sub qscope)) by Th8
     .= All(S_Bound(@sub2),CFO_Sub qscope2) by A42,A43,A29,A38,A39,A40,Th17
     .= All(S_Bound(@sub2),CFO_Sub scope2) by A32,SUBLEM_2:30
     .= Quant(sub2,CFO_Sub scope2) by SUBST1_2:def 37
     .= CFOQuant(sub2, CFO_Sub scope2) by A32,SUBLEM_2:27,def 7
     .= CFO_Sub [p2,S2] by A32,A44,SUBLEM_2:27,28;
    hence thesis;
  end;
then
A45: for r,s,x,k,l,P holds P[VERUM(Al)] & P[P!l] & (P[r] implies P['not' r]) &
 (P[r] & P[s] implies P[r '&' s]) & (P[r] implies P[All(x, r)])
   by A1,A4,A19,A23;
A46: for p being Element of CFO-WFF(Al) holds P[p] from CFO_LANG:sch 1(A45);
  let p2 be Element of CFO-WFF(Al2),x2,y2 be bound_FO-variable of Al2 ,p,x,y
   such that
A47: p=p2 & x=x2 & y=y2;
  thus p.(x,y) = CFO_Sub [p,Sbst(x,y)] by SUBST2_2:def 1
   .= CFO_Sub [p2,Sbst(x2,y2)] by A46,A47 .= p2.(x2,y2) by SUBST2_2:def 1;
end;

theorem Th19:
 for PHI being Consistent Subset of CFO-WFF(Al2) st
 PHI is Subset of CFO-WFF(Al) holds PHI is Al-Consistent
proof
  let PHI be Consistent Subset of CFO-WFF(Al2) such that
    PHI is Subset of CFO-WFF(Al);
  for S being Subset of CFO-WFF(Al) st PHI=S holds S is Consistent
  proof
    let S be Subset of CFO-WFF(Al) such that
A1:  PHI=S;
    assume S is Inconsistent;
     then
A2:  S |- 'not' VERUM(Al) by GOEDCP_2:24;
     PHI |- 'not' VERUM(Al2)
     proof
       consider f being FinSequence of CFO-WFF(Al) such that
A3:     rng f c= S & |- f^<*'not' VERUM(Al)*> by A2,HENMOD_2:def 1;
       set f2 = f;
       for x being set st x in rng f2 holds x in CFO-WFF(Al2)
       proof
         let x be set such that
A5:       x in rng f2;
         x in PHI by A1,A3,A5;
         hence x in CFO-WFF(Al2);
       end;
       then rng f2 c= CFO-WFF(Al2) by TARSKI:def 3;
       then reconsider f2 as FinSequence of CFO-WFF(Al2) by FINSEQ_1:def 4;
       consider PR such that
A6:     PR is a_proof & f^<*'not' VERUM(Al)*> = (PR.(len PR))`1
       by A3,CALCL1_2:def 9;
A7:    PR <> {} & for n being Nat st 1 <= n & n <= len PR holds
        PR,n is_a_correct_step by A6,CALCL1_2:def 8;
      set PR2 = PR;
      PR2 is FinSequence of [:set_of_CFO-WFF-seq(Al2),Proof_Step_Kinds:]
      proof
        for p being set holds p in CFO-WFF(Al) implies p in CFO-WFF(Al2)
        proof
          let p be set;
          assume p in CFO-WFF(Al);
          then p is Element of CFO-WFF(Al2) by Th7;
          hence thesis;
        end;
        then
A8:     CFO-WFF(Al) c= CFO-WFF(Al2) &
        rng PR2 c= [:set_of_CFO-WFF-seq(Al),Proof_Step_Kinds:] by TARSKI:def 3;
        for x being set holds x in set_of_CFO-WFF-seq(Al) implies
         x in set_of_CFO-WFF-seq(Al2)
        proof
          let x be set;
          assume x in set_of_CFO-WFF-seq(Al);
          then reconsider x as FinSequence of CFO-WFF(Al) by CALCL1_2:def 6;
          rng x c= CFO-WFF(Al2) by A8,XBOOLE_1:1;
          then x is FinSequence of CFO-WFF(Al2) by FINSEQ_1:def 4;
          hence thesis by CALCL1_2:def 6;
        end;
        then set_of_CFO-WFF-seq(Al) c= set_of_CFO-WFF-seq(Al2) by TARSKI:def 3;
        then [:set_of_CFO-WFF-seq(Al),Proof_Step_Kinds:] c=
         [:set_of_CFO-WFF-seq(Al2),Proof_Step_Kinds:] by ZFMISC_1:95;
        then rng PR2 c= [:set_of_CFO-WFF-seq(Al2),Proof_Step_Kinds:]
         by XBOOLE_1:1;
        hence thesis by FINSEQ_1:def 4;
      end;
      then reconsider PR2 as FinSequence of
       [:set_of_CFO-WFF-seq(Al2),Proof_Step_Kinds:];
A9:   PR2 is a_proof
      proof
        for n being Nat st 1 <= n & n <= len PR2 holds PR2,n is_a_correct_step
        proof
          let n be Nat such that
A10:       1 <= n & n <= len PR2;
A11:      (for i st 1 <= i & i<n holds (PR2.i)`1 in set_of_CFO-WFF-seq(Al2)) &
           ((PR2.n)`1 in set_of_CFO-WFF-seq(Al2))
          proof
            thus for i st 1 <= i & i < n holds
             (PR2.i)`1 in set_of_CFO-WFF-seq(Al2)
            proof
              let i such that
A12:           1 <= i & i < n;
              set k = len PR2 - n;
              reconsider k as Element of NAT by A10,NAT_1:21;
              len PR2 = n + k;
              then
A13:          1 <= i & i <= len PR2 by A12, NAT_1:12;
              dom PR2 = Seg (len PR2) by FINSEQ_1:def 3;
              then i in dom PR2 by A13,FINSEQ_1:1;
              then PR2.i in rng PR2 by FUNCT_1:def 3;
              hence (PR2.i)`1 in set_of_CFO-WFF-seq(Al2) by MCART_1:10;
            end;
            dom PR2 = Seg (len PR2) by FINSEQ_1:def 3;
            then n in dom PR2 by A10,FINSEQ_1:1;
            then PR2.n in rng PR2 by FUNCT_1:def 3;
            hence (PR2.n)`1 in set_of_CFO-WFF-seq(Al2) by MCART_1:10;
          end;
A14:      PR,n is_a_correct_step by A6,CALCL1_2:def 8,A10;
          per cases by A10,CALCL1_2:31;
          suppose
A15:        (PR2.n)`2 = 0;
            then consider g2 being FinSequence of CFO-WFF(Al) such that
A16:         (Suc g2 is_tail_of Ant g2 & (PR2.n)`1 = g2) by A14,CALCL1_2:def 7;
            g2 is FinSequence of CFO-WFF(Al2) by A11,A16,CALCL1_2:def 6;
            then consider g being FinSequence of CFO-WFF(Al2) such that
A17:         g=g2;
A18:        Suc g = Suc g2 & Ant g = Ant g2 by A17,Th11;
            thus thesis by A15,A16,A17,A18,CALCL1_2:def 7;
          end;
          suppose
A19:        (PR2.n)`2 = 1;
            then consider g2 being FinSequence of CFO-WFF(Al) such that
A20:         ((PR.n)`1 = g2^<*VERUM(Al)*>) by A14,CALCL1_2:def 7;
            g2^<*VERUM(Al)*> is FinSequence of CFO-WFF(Al2)
```

```
               by A11,A20,CALCL1_2:def 6;
              then consider gp being FinSequence of CFO-WFF(Al2) such that
A21:           gp=g2^<*VERUM(Al)*>;
              len gp <> 0 by A21;
              then consider g being FinSequence of CFO-WFF(Al2),
               v being Element of CFO-WFF(Al2) such that
A22:           gp = g^<*v*> by FINSEQ_2:19;
              g = g2 & v = VERUM(Al) by A21,A22,FINSEQ_2:17;
              then v = <*[0,0]*> by FO_LANG1:def 14
                     .= VERUM(Al2) by FO_LANG1:def 14;
              hence thesis by A19,A20,A21,A22,CALCL1_2:def 7;
            end;
            suppose
A23:          (PR2.n)`2 = 2;
              then consider i being Element of NAT, g2,h2 being
               FinSequence of CFO-WFF(Al) such that
A24:           (1 <= i & i<n & Ant g2 is_Subsequence_of Ant h2 & Suc g2 = Suc h2
               & (PR2.i)`1 = g2 & (PR2.n)`1 = h2) by A14,CALCL1_2:def 7;
              g2 in set_of_CFO-WFF-seq(Al2) & h2 in set_of_CFO-WFF-seq(Al2)
               by A11,A24;
              then h2 is FinSequence of CFO-WFF(Al2) &
                   g2 is FinSequence of CFO-WFF(Al2) by CALCL1_2:def 6;
              then consider g,h being FinSequence of CFO-WFF(Al2) such that
A25:           g=g2 & h=h2;
A26:          Suc g = Suc g2 by A25,Th11
                   .= Suc h by A24,A25,Th11;
              consider N being Subset of NAT such that
A27:           (Ant g2) c= Seq((Ant h2)|N) by A24,CALCL1_2:def 4;
              (Ant h2)|N = (Ant h)|N by A25,Th11;
              then (Ant g) c= Seq((Ant h)|N) by A25,A27,Th11;
              then
A28:          Ant g is_Subsequence_of Ant h by CALCL1_2:def 4;
              thus thesis by A23,A24,A25,A26,A28,CALCL1_2:def 7;
            end;
            suppose
A30:          (PR2.n)`2 = 3;
              then consider i,j being Element of NAT, g,h being
               FinSequence of CFO-WFF(Al) such that
A31:           (1 <= i & i < n & 1 <= j & j < i & len g > 1 & len h > 1 &
               Ant (Ant g) = Ant (Ant h) & 'not' (Suc (Ant g)) = Suc (Ant h)
               & Suc g = Suc h & g = (PR2.j) `1 & h = (PR2.i)`1 &
               (Ant (Ant g))^<*(Suc g)*> = (PR2.n)`1) by A14,CALCL1_2:def 7;
              (PR2.j)`1 = g & (PR2.i)`1 = h & j < n by A31,Th12;
              then g in set_of_CFO-WFF-seq(Al2) & h in set_of_CFO-WFF-seq(Al2)
               by A11,A31;
              then h is FinSequence of CFO-WFF(Al2) &
                   g is FinSequence of CFO-WFF(Al2) by CALCL1_2:def 6;
              then consider g2,h2 being FinSequence of CFO-WFF(Al2) such that
A32:           g2=g & h2=h;
A34:          Ant g2 = Ant g & Ant h2 = Ant h by A32,Th11;
              then
A35:          Ant (Ant g2) = Ant (Ant g) by Th11
                          .= Ant (Ant h2) by A31,A34,Th11;
A36:          'not' (Suc (Ant g2)) = 'not' (Al2-Cast(Suc (Ant g))) by A34,Th11
                                  .= Al2-Cast('not' (Suc (Ant g))) by Th8
                                  .= Suc (Ant h2) by A31,A34,Th11;
A37:          Suc g2 = Suc g by A32,Th11
                   .= Suc h2 by A31,A32,Th11;
A39:          (PR2.n)`1 = (Ant (Ant g))^<*(Suc g2)*> by A31,A32,Th11
                       .= (Ant (Ant g2))^<*Suc g2*> by A34,Th11;
              thus thesis by A30,A31,A32,A35,A36,A37,A39,CALCL1_2:def 7;
            end;
            suppose
A40:          (PR2.n)`2 = 4;
              then consider i,j being Element of NAT, g,h being FinSequence of
               CFO-WFF(Al), p being Element of CFO-WFF(Al) such that
A41:           (1 <= i & i < n & 1 <= j & j < i & len g > 1 & Ant g = Ant h &
                Suc (Ant g) = 'not' p & 'not' (Suc g) = Suc h & g = (PR2.j)`1
                & h = (PR2.i)`1 & (Ant (Ant g))^<*p*> = (PR2.n)`1)
                by A14,CALCL1_2:def 7;
              (PR2.j)`1 = g & (PR2.i)`1 = h & j < n by A41,Th12;
              then g in set_of_CFO-WFF-seq(Al2) & h in set_of_CFO-WFF-seq(Al2)
               by A11,A41;
              then h is FinSequence of CFO-WFF(Al2) &
                   g is FinSequence of CFO-WFF(Al2) by CALCL1_2:def 6;
              then consider g2,h2 being FinSequence of CFO-WFF(Al2) such that
A42:           g2=g & h2=h;
A44:          Ant g2 = Ant g by A42,Th11 .= Ant h2 by A41,A42,Th11;
              Ant g2 = Ant g by A42,Th11;
              then
A45:          Suc (Ant g2) = Al2-Cast('not' p) by A41,Th11
                          .= 'not' (Al2-Cast(p)) by Th8;
A46:          'not' (Suc g2) = 'not' (Al2-Cast((Suc g))) by A42,Th11
                            .= Al2-Cast('not' (Suc g)) by Th8
                            .= Suc h2 by A41,A42,Th11;
              Ant g2 = Ant g by A42,Th11;
              then (Ant (Ant g2))^<*Al2-Cast(p)*> = (PR2.n)`1 by Th11,A41;
              hence thesis by A40,A41,A42,A44,A45,A46,CALCL1_2:def 7;
            end;
            suppose
A48:         (PR2.n)`2 = 5;
              then consider i,j being Element of NAT, g,h being FinSequence of
               CFO-WFF(Al) such that
A49:           (1 <= i & i < n & 1<=j & j<i & Ant g = Ant h & g = (PR2.j)`1 &
                h = (PR2.i)`1 & (Ant g)^<*((Suc g) '&' (Suc h))*> =(PR2.n)`1)
                by A14,CALCL1_2:def 7;
              (PR2.j)`1 = g & (PR2.i)`1 = h & j < n by A49,Th12;
              then g in set_of_CFO-WFF-seq(Al2) & h in set_of_CFO-WFF-seq(Al2)
               by A11,A49;
              then h is FinSequence of CFO-WFF(Al2) &
                   g is FinSequence of CFO-WFF(Al2) by CALCL1_2:def 6;
              then consider g2,h2 being FinSequence of CFO-WFF(Al2) such that
A50:          g=g2 & h=h2;
              Al2-Cast(Suc g) = Suc g2 & Al2-Cast(Suc h) = Suc h2 by A50,Th11;
              then (Suc g2)'&'(Suc h2) = Al2-Cast((Suc g) '&' (Suc h)) by Th8
                          .=((Suc g) '&' (Suc h));
              then
A51:         (Ant g2)^<*((Suc g2) '&' (Suc h2))*> = (PR2.n)`1 by A49,A50,Th11;
              Ant g2 = Ant g by A50,Th11 .= Ant h2 by A49,A50,Th11;
              hence thesis by A48,A49,A50,A51,CALCL1_2:def 7;
            end;
            suppose
A52:         (PR2.n)`2 = 6;
              then consider i being Element of NAT, g being FinSequence of
               CFO-WFF(Al), p,q being Element of CFO-WFF(Al) such that
A53:          (1 <= i & i < n & p '&' q = Suc g & g = (PR2.i)`1 &
               (Ant g)^<*p*> = (PR2.n)`1) by A14,CALCL1_2:def 7;
              g in set_of_CFO-WFF-seq(Al2) by A11,A53;
              then g is FinSequence of CFO-WFF(Al2) by CALCL1_2:def 6;
              then consider g2 being FinSequence of CFO-WFF(Al2) such that
A54:          g=g2;
A55:         Suc g2 = Al2-Cast(p '&' q) by A53,A54,Th11
                   .= (Al2-Cast(p)) '&' (Al2-Cast(q)) by Th8;
              (Ant g2)^<*p*> = (PR2.n)`1 by A53,A54,Th11;
              hence thesis by A52,A53,A54,A55,CALCL1_2:def 7;
            end;
            suppose
A56:         (PR2.n)`2 = 7;
              then consider i being Element of NAT, g being FinSequence of
               CFO-WFF(Al), p,q being Element of CFO-WFF(Al) such that
A57:          (1 <= i & i < n & p '&' q = Suc g & g = (PR2.i)`1 &
               (Ant g)^<*q*> = (PR2.n)`1) by A14,CALCL1_2:def 7;
              g in set_of_CFO-WFF-seq(Al2) by A11,A57;
              then g is FinSequence of CFO-WFF(Al2) by CALCL1_2:def 6;
              then consider g2 being FinSequence of CFO-WFF(Al2) such that
A58:          g=g2;
A59:         Suc g2 = Al2-Cast(p '&' q) by A57,A58,Th11
                   .= (Al2-Cast(p)) '&' (Al2-Cast(q)) by Th8;
              (Ant g2)^<*Al2-Cast(q)*> = (PR2.n)`1 by A57,A58,Th11;
              hence thesis by A56,A57,A58,A59,CALCL1_2:def 7;
            end;
            suppose
A60:         (PR2.n)`2 = 8;
              then consider i being Element of NAT, g being FinSequence of
               CFO-WFF(Al), p being Element of CFO-WFF(Al), x,y being
               bound_FO-variable of Al such that
A61:          (1 <= i & i < n & Suc g = All(x,p) & g = (PR2.i)`1 &
               (Ant g)^<*(p.(x,y))*> = (PR2.n)`1 ) by A14,CALCL1_2:def 7;
              g in set_of_CFO-WFF-seq(Al2) by A11,A61;
              then g is FinSequence of CFO-WFF(Al2) by CALCL1_2:def 6;
              then consider g2 being FinSequence of CFO-WFF(Al2) such that
A62:          g=g2;
              bound_FO-variables(Al) c= bound_FO-variables(Al2) by Th4;
              then p is Element of CFO-WFF(Al2) & x is bound_FO-variable of Al2 &
               y is bound_FO-variable of Al2 by Th7,TARSKI:def 3;
              then consider q being Element of CFO-WFF(Al2),
               a,b being bound_FO-variable of Al2 such that
A63:          a = x & b = y & q = p;
A64:         (PR2.n)`1 = (Ant g)^<*(q.(a,b))*> by A61,A63,Th18
                            .= (Ant g2)^<*(q.(a,b))*> by A62,Th11;
               Suc g2 = Al2-Cast(All(x,p)) by A61,A62,Th11
                    .= All(Al2-Cast(x),Al2-Cast(p)) by Th8
                    .= All(a,q) by A63;
              hence thesis by A60,A61,A62,A64,CALCL1_2:def 7;
            end;
            suppose
A65:         (PR2.n)`2 = 9;
              then consider i being Element of NAT, g being FinSequence of
               CFO-WFF(Al), p being Element of CFO-WFF(Al), x,y being
               bound_FO-variable of Al such that
A66:         (1 <= i & i < n & Suc g = p.(x,y) & not y in still_not-bound_in
               (Ant g) & not y in still_not-bound_in (All(x,p)) & g=(PR2.i)`1 &
               (Ant g)^<*(All(x,p))*> = (PR2.n)`1) by A14,CALCL1_2:def 7;
              g in set_of_CFO-WFF-seq(Al2) by A11,A66;
              then g is FinSequence of CFO-WFF(Al2) by CALCL1_2:def 6;
              then consider g2 being FinSequence of CFO-WFF(Al2) such that
A67:         g=g2;
              bound_FO-variables(Al) c= bound_FO-variables(Al2) by Th4;
              then p is Element of CFO-WFF(Al2) & x is bound_FO-variable of Al2 &
               y is bound_FO-variable of Al2 by Th7,TARSKI:def 3;
              then consider q being Element of CFO-WFF(Al2),
               a,b being bound_FO-variable of Al2 such that
A68:         q = p & a = x & b = y;
A69:        Suc g2 = Suc g by A67,Th11 .= q.(a,b) by A66,A68,Th18;
A70:         still_not-bound_in All(x,p) =
               still_not-bound_in (Al2-Cast(All(x,p))) by Th13
               .= still_not-bound_in All(Al2-Cast(x),Al2-Cast(p)) by Th8
               .= still_not-bound_in All(a,q) by A68;
A71:        not b in still_not-bound_in (Ant g2)
              proof
                assume b in still_not-bound_in (Ant g2);
                then consider i being Element of NAT,
                 r being Element of CFO-WFF(Al2) such that
A72:           i in dom (Ant g2) & r = (Ant g2).i & b in still_not-bound_in r
                 by CALCL1_2:def 5;
A73:         dom (Ant g2) = dom (Ant g) by A67,Th11;
               r = (Ant g).i by A67,A72,Th11;
               then reconsider r as Element of CFO-WFF(Al)
                by A72,A73,FINSEQ_2:11;
                i in dom (Ant g) & Al2-Cast(r) = (Ant g).i &
                 b in still_not-bound_in (Al2-Cast(r)) by A67,A72,Th11;
                then i in dom (Ant g) & r = (Ant g).i &
                 y in still_not-bound_in r by A68,Th13;
                hence contradiction by A66,CALCL1_2:def 5;
              end;
              All(x,p) = Al2-Cast(All(x,p))
                            .= All(Al2-Cast(x),Al2-Cast(p)) by Th8
                            .= All(a,q) by A68;
              then (Ant g2)^<*(All(a,q))*> = (PR2.n)`1 by A66,A67,Th11;
              hence thesis by A65,A66,A67,A68,A69,A70,A71,CALCL1_2:def 7;
            end;
          end;
          hence thesis by A7,CALCL1_2:def 8;
        end;
        Al2-Cast('not' VERUM(Al)) = 'not' (Al2-Cast(VERUM(Al))) by Th8
                                .= 'not' VERUM(Al2) by Th8;
        then |- f2^<*('not' VERUM(Al2))*> by A6,A9,CALCL1_2:def 9;
        hence thesis by A1,A3,HENMOD_2:def 1;
       end;
       hence contradiction by GOEDCP_2:24;
    end;
    hence thesis by Def2;
end;

theorem Th20:
 for p ex Al1 being countable FO-alphabet st p is Element of CFO-WFF(Al1)
 & Al is Al1-extending
proof
    defpred P[Element of CFO-WFF(Al)] means ex Al1 being countable FO-alphabet
     st $1 is Element of CFO-WFF(Al1) & Al is Al1-extending;
A1: P[VERUM(Al)]
    proof
      set Al1 = [:NAT, NAT:];
      reconsider Al1 as countable FO-alphabet by FO_LANG1:def 1,CARD_4:7;
      NAT c= FO-symbols(Al) & Al = [:NAT,FO-symbols(Al):] by FO_LANG1:3,5;
      then Al1 c= Al by ZFMISC_1:95;
      then reconsider Al as Al1-extending FO-alphabet by Def1;
      VERUM(Al1) = Al-Cast(VERUM(Al1)) .= VERUM(Al) by Th8;
      hence thesis;
    end;
A2: for k,P,l holds P[P!l]
    proof
      let k,P,l;
      bound_FO-variables(Al) c= FO-variables(Al) &
       FO-variables(Al) c= [:NAT, FO-symbols(Al):] by FO_LANG1:4;
      then bound_FO-variables(Al) c= [:NAT, FO-symbols(Al):] by XBOOLE_1:1;
      then rng l c= [:NAT, FO-symbols(Al):] by XBOOLE_1:1;
      then consider A,B being set such that
A4:  A is finite & A c= NAT & B is finite & B c= FO-symbols(Al) &
       rng l c= [:A,B:] by FINSET_1:13;
      [:A,B:] c= [:NAT,B:] by A4,ZFMISC_1:95;
      then
```

```
A5: rng l c= [:NAT,B:] by A4,XBOOLE_1:1;
    set Al1 = [:NAT, NAT:] \/ [:NAT, {P`2}:] \/ [:NAT,B:];
    [:NAT, NAT:] is countable & [:NAT, {P`2}:] is countable by CARD_4:7;
    then
A6: [:NAT, NAT:] \/ [:NAT, {P`2}:] is countable & [:NAT,B:] is countable
     by A4,CARD_2:85,CARD_4:7;
A7: Al1 = [:NAT, NAT \/ {P`2}:] \/ [:NAT,B:] by ZFMISC_1:97
       .= [:NAT, NAT \/ {P`2} \/ B:] by ZFMISC_1:97;
    NAT c= NAT \/ {P`2} & NAT \/ {P`2} c= NAT \/ {P`2} \/ B by XBOOLE_1:7;
    then NAT c= NAT \/ {P`2} \/ B by XBOOLE_1:1;
    then reconsider Al1 as countable FO-alphabet
     by A6,A7,FO_LANG1:def 1,CARD_2:85;
    P is Element of FO-pred_symbols(Al) & FO-pred_symbols(Al)
     c= [:NAT,FO-symbols(Al):] by FO_LANG1:6;
    then P in [:NAT,FO-symbols(Al):] by TARSKI:def 3;
    then consider a,b being set such that
A8:  a in NAT & b in FO-symbols(Al) & P = [a,b] by ZFMISC_1:def 2;
    P`2 in FO-symbols(Al) by A8,MCART_1:def 2;
    then {P`2} c= FO-symbols(Al) & NAT c= NAT & NAT c= FO-symbols(Al)
     by FO_LANG1:3,ZFMISC_1:31;
    then [:NAT,{P`2}:] c= [:NAT,FO-symbols(Al):] &
     [:NAT,NAT:] c= [:NAT, FO-symbols(Al):] by ZFMISC_1:95;
    then [:NAT,NAT:] \/ [:NAT,{P`2}:] c= [:NAT,FO-symbols(Al):]
      & [:NAT,B:] c= [:NAT,FO-symbols(Al):] by A4,ZFMISC_1:95, XBOOLE_1:8;
    then Al1 c= [:NAT,FO-symbols(Al):] by XBOOLE_1:8;
    then Al1 c= Al by FO_LANG1:5;
    then reconsider Al as Al1-extending FO-alphabet by Def1;
    [:NAT, NAT \/ {P`2} \/ B:] = [:NAT, FO-symbols(Al1):] by A7,FO_LANG1:5;
    then
A9: FO-symbols(Al1) = NAT \/ {P`2} \/ B by ZFMISC_1:110;
    set P2 = [a,b];
    b = P`2 by A8,MCART_1:def 2;
    then b in {P`2} by TARSKI:def 1;
    then b in NAT \/ {P`2} by XBOOLE_0:def 3;
    then reconsider b as FO-symbol of Al1 by A9,XBOOLE_0:def 3;
    reconsider a as Element of NAT by A8;
A10: P`1 = 7 + the_arity_of P & P`1 = a by A8,FO_LANG1:def 8,MCART_1:def 1;
    then 7 <= a by NAT_1:11;
    then [a,b] in {[n,x] where x is FO-symbol of Al1 : 7 <= n};
    then reconsider P2 as FO-pred_symbol of Al1 by FO_LANG1:def 7;
    P2`1 = 7 + k by A8,A10,FO_LANG1:11;
    then the_arity_of P2 = k by FO_LANG1:def 8;
    then P2 in {Q where Q is FO-pred_symbol of Al1: the_arity_of Q = k};
    then reconsider P2 as FO-pred_symbol of k,Al1 by FO_LANG1:def 9;
    set l2 = l;
    for s being set st s in rng l2 holds s in bound_FO-variables(Al1)
    proof
      let s be set such that
A11:   s in rng l2;
       s in bound_FO-variables(Al) by A11;
       then s in [:{4}, FO-symbols(Al):] by FO_LANG1:def 4;
       then consider s1,s2 being set such that
A12:    s1 in {4} & s2 in FO-symbols(Al) & s = [s1,s2] by ZFMISC_1:def 2;
       B c= FO-symbols(Al1) by A9,XBOOLE_1:7;
       then
A13:   [:NAT,B:] c= [:NAT,FO-symbols(Al1):] by ZFMISC_1:95;
       s in [:NAT,B:] by A5,A11;
       then consider s3,s4 being set such that
A14:    s3 in NAT & s4 in FO-symbols(Al1) & s = [s3,s4] by A13,ZFMISC_1:def 2;
       s = [s1,s4] by A12,A14,ZFMISC_1:27;
       then s in [:{4},FO-symbols(Al1):] by A12,A14,ZFMISC_1:def 2;
       hence thesis by FO_LANG1:def 4;
    end;
    then
A15: rng l2 c= bound_FO-variables(Al1) by TARSKI:def 3;
    then rng l2 c= FO-variables(Al1) by XBOOLE_1:1;
    then reconsider l2 as CFO-variable_list of k,Al1 by A15,FINSEQ_1:def 4;
    P2!l2 = Al-Cast(P2!l2) .= Al-Cast(P2)!Al-Cast(l2) by Th8 .= P!l by A8;
    then P!l is Element of CFO-WFF(Al1) & Al is Al1-extending;
    hence thesis;
  end;
A17: for r st P[r] holds P['not' r]
  proof
    let r such that
A18: P[r];
    consider Al1 being countable FO-alphabet such that
A19: r is Element of CFO-WFF(Al1) & Al is Al1-extending by A18;
    reconsider Al as Al1-extending FO-alphabet by A19;
    consider r2 being Element of CFO-WFF(Al1) such that
A20: r = r2 by A19;
    'not' r2 = Al-Cast('not' r2) .= 'not' Al-Cast(r2) by Th8
     .= 'not' r by A20;
    hence thesis by A19;
  end;
```

```
A21: for r,s st P[r] & P[s] holds P[r '&' s]
  proof
    let r,s such that
A22: P[r] & P[s];
    consider Al1, Al2 being countable FO-alphabet such that
A23: r is Element of CFO-WFF(Al1) & s is Element of CFO-WFF(Al2) &
     Al is Al1-extending & Al is Al2-extending by A22;
    set Al3 = Al1 \/ Al2;
    Al1 = [:NAT,FO-symbols(Al1):] & Al2 =[:NAT,FO-symbols(Al2):] by FO_LANG1:5;
    then
A24: Al3 = [:NAT, FO-symbols(Al1) \/ FO-symbols(Al2):] by ZFMISC_1:97;
    NAT c= FO-symbols(Al1) \/ FO-symbols(Al2) by XBOOLE_1:10,FO_LANG1:3;
    then reconsider Al3 as FO-alphabet by A24,FO_LANG1:def 1;
    Al1 c= Al3 & Al2 c= Al3 by XBOOLE_1:7;
    then reconsider Al3 as countable Al1-extending Al2-extending FO-alphabet
     by Def1,CARD_2:85;
    consider r2 being Element of CFO-WFF(Al1), s2 being Element of CFO-WFF(Al2)
     such that
A25: r2 = r & s2 = s by A23;
    reconsider r2 as Element of CFO-WFF(Al3) by Th7;
    reconsider s2 as Element of CFO-WFF(Al3) by Th7;
    Al1 c= Al & Al2 c= Al by A23,Def1;
    then Al3 c= Al by XBOOLE_1:8;
    then reconsider Al as Al3-extending FO-alphabet by Def1;
    r2 '&' s2 = Al-Cast(r2 '&' s2) .= Al-Cast(r2) '&' Al-Cast(s2) by Th8
     .= r '&' s by A25;
    then r '&' s is Element of CFO-WFF(Al3) & Al is Al3-extending;
    hence thesis;
  end;
for x,r st P[r] holds P[All(x,r)]
  proof
    let x,r such that
A26: P[r];
    consider Al1 being countable FO-alphabet such that
A27: r is Element of CFO-WFF(Al1) & Al is Al1-extending by A26;
    x in bound_FO-variables(Al);
    then x in [:{4}, FO-symbols(Al):] by FO_LANG1:def 4;
    then consider s1,s2 being set such that
A28:  s1 in {4} & s2 in FO-symbols(Al) & x = [s1,s2] by ZFMISC_1:def 2;
    set Al2 = [:NAT, FO-symbols(Al1) \/ {s2}:];
A29: Al1 = [:NAT, FO-symbols(Al1):] & FO-symbols(Al1) c= FO-symbols(Al1)\/{s2}
     & NAT c= FO-symbols(Al1) by FO_LANG1:3,5, XBOOLE_1:7;
    then Al1 c= Al2 & NAT c= FO-symbols(Al1)\/{s2} by XBOOLE_1:1,ZFMISC_1:95;
    then reconsider Al2 as Al1-extending FO-alphabet by Def1,FO_LANG1:def 1;
A30: Al2 = [:NAT, FO-symbols(Al1):] \/ [:NAT, {s2}:]
     & [:NAT, FO-symbols(Al1):] c= Al by A27,A29,Def1,ZFMISC_1:97;
    [:NAT, FO-symbols(Al1):] is countable & [:NAT, {s2}:] is countable
     by FO_LANG1:5,CARD_4:7;
    then reconsider Al2 as countable Al1-extending FO-alphabet
     by A30,CARD_2:85;
    {s2} c= FO-symbols(Al) by A28,ZFMISC_1:31;
    then [:NAT, {s2}:] c= [:NAT,FO-symbols(Al):] & Al = [:NAT,FO-symbols(Al):]
     by FO_LANG1:5,ZFMISC_1:96;
    then Al2 c= Al by A30,XBOOLE_1:8;
    then reconsider Al as Al2-extending FO-alphabet by Def1;
    consider r2 being Element of CFO-WFF(Al1) such that
A31: r = r2 by A27;
    reconsider r2 as Element of CFO-WFF(Al2) by Th7;
A32: x = [4,s2] by A28,TARSKI:def 1;
    Al2 = [:NAT,FO-symbols(Al2):] by FO_LANG1:5;
    then FO-symbols(Al2) = FO-symbols(Al1) \/ {s2} & s2 in {s2}
     by TARSKI:def 1,ZFMISC_1:110;
    then s2 in FO-symbols(Al2) by XBOOLE_0:def 3;
    then x in [:{4},FO-symbols(Al2):] by A32,ZFMISC_1:105;
    then x is bound_FO-variable of Al2 by FO_LANG1:def 4;
    then consider x2 being bound_FO-variable of Al2 such that
A33: x = x2;
    All(x2,r2) = Al-Cast(All(x2,r2)) .= All(Al-Cast(x2),Al-Cast(r2)) by Th8
     .= All(x,r) by A31,A33;
    then All(x,r) is Element of CFO-WFF(Al2) & Al is Al2-extending;
    hence thesis;
  end;
  then
A34: for r,s,x,k,l,P holds P[VERUM(Al)] & P[P!l] & (P[r] implies P['not' r]) &
  (P[r] & P[s] implies P[r '&' s]) & (P[r] implies P[All(x,r)])
  by A1,A2,A17,A21;
  for p holds P[p] from CFO_LANG:sch 1(A34);
  hence thesis;
end;

theorem Th21:
 for PHI being finite Subset of CFO-WFF(Al) ex Al1 being countable FO-alphabet
 st PHI is finite Subset of CFO-WFF(Al1) & Al is Al1-extending
proof
```

```
  let PHI be finite Subset of CFO-WFF(Al);
  defpred P[set] means $1 is finite Subset of CFO-WFF(Al) implies
   ex Al1 being countable FO-alphabet st $1 is finite Subset of CFO-WFF(Al1) &
   Al is Al1-extending;
A1: PHI is finite;
A2: P[{}]
  proof
    set Al1 = [:NAT,NAT:];
    reconsider Al1 as countable FO-alphabet by FO_LANG1:def 1,CARD_4:7;
    Al = [:NAT,FO-symbols(Al):] & NAT c= FO-symbols(Al) by FO_LANG1:3,5;
    then Al1 c= Al by ZFMISC_1:96;
    then Al is Al1-extending & {} is finite Subset of CFO-WFF(Al1) by
     Def1, XBOOLE_1:2;
    hence thesis;
  end;
A3: for x,B being set st x in PHI & B c= PHI & P[B] holds P[B\/{x}]
  proof
    let x,B be set such that
A4:  x in PHI & B c= PHI & P[B];
    reconsider x as Element of CFO-WFF(Al) by A4;
    reconsider B as finite Subset of CFO-WFF(Al) by A4,XBOOLE_1:1;
    consider Al1 being countable FO-alphabet such that
A5:  x is Element of CFO-WFF(Al1) & Al is Al1-extending by Th20;
    consider Al2 being countable FO-alphabet such that
A6:  B is finite Subset of CFO-WFF(Al2) & Al is Al2-extending by A4;
    set Al3 = Al1 \/ Al2;
    Al1 = [:NAT,FO-symbols(Al1):] & Al2 =[:NAT,FO-symbols(Al2):] by FO_LANG1:5;
    then
A7:  Al3 = [:NAT, FO-symbols(Al1) \/ FO-symbols(Al2):] by ZFMISC_1:97;
    NAT c= FO-symbols(Al1) \/ FO-symbols(Al2) by FO_LANG1:3,XBOOLE_1:10;
    then reconsider Al3 as FO-alphabet by A7,FO_LANG1:def 1;
    Al1 c= Al3 & Al2 c= Al3 by XBOOLE_1:7;
    then reconsider Al3 as countable Al1-extending Al2-extending FO-alphabet
     by Def1,CARD_2:85;
    consider x2 being Element of CFO-WFF(Al1) such that
A9:  x = x2 by A5;
    for s being set st s in B holds s in CFO-WFF(Al3)
    proof
      let s be set such that
A10:   s in B;
      consider s2 being Element of CFO-WFF(Al2) such that
A11:   s = s2 by A6,A10;
      s2 is Element of CFO-WFF(Al3) by Th7;
      hence s in CFO-WFF(Al3) by A11;
    end;
    then x2 is Element of CFO-WFF(Al3) & B c= CFO-WFF(Al3) by Th7,TARSKI:def 3;
    then {x2} c= CFO-WFF(Al3) & B c= CFO-WFF(Al3) by ZFMISC_1:31;
    then
A12: B \/ {x} c= CFO-WFF(Al3) by A9,XBOOLE_1:8;
    Al1 c= Al & Al2 c= Al by A5,A6,Def1;
    then Al3 c= Al by XBOOLE_1:8;
    then Al is Al3-extending FO-alphabet by Def1;
    hence thesis by A12;
  end;
  P[PHI] from FINSET_1:sch 2(A1,A2,A3);
  hence thesis;
end;

theorem Th22:
 for PHI being finite Subset of CFO-WFF(Al) holds
 still_not-bound_in PHI is finite
proof
  let PHI be finite Subset of CFO-WFF(Al);
  deffunc snb(Element of CFO-WFF(Al)) = still_not-bound_in $1;
A1: for x being set st x in {snb(p) : p in PHI} holds x is finite
  proof
    let x be set such that
A2:  x in {(still_not-bound_in p) : p in PHI};
    consider p such that
A3:  x = still_not-bound_in p & p in PHI by A2;
    thus x is finite by A3,CFO_SIM1:19;
  end;
A4: PHI is finite;
  {snb(p) : p in PHI} is finite from FRAENKEL:sch 21(A4);
  hence still_not-bound_in PHI is finite by A1,FINSET_1:7;
end;

theorem Th23:
 for THETA being Subset of CFO-WFF(Al2) st PHI = THETA holds
  for A,J,v st J,v |= PHI ex A2 being non empty set, J2 being interpretation
  of Al2,A2, v2 being Element of Valuations_in (Al2,A2) st J2,v2 |= THETA
proof
  let THETA be Subset of CFO-WFF(Al2) such that
A1: PHI = THETA;
```

```
  let A,J,v such that
A2: J,v |= PHI;
  set J2 = (FO-pred_symbols(Al2) --> empty_rel(A)) +* J;
A3: dom J = FO-pred_symbols(Al) & dom (FO-pred_symbols(Al2) --> empty_rel(A)) =
   FO-pred_symbols(Al2) by FUNCT_2:def 1;
  then dom J2 = FO-pred_symbols(Al) \/ FO-pred_symbols(Al2) by FUNCT_4:def 1;
  then
A4: dom J2 = FO-pred_symbols(Al2) by Th3,XBOOLE_1:12;
  J in Funcs(FO-pred_symbols(Al),relations_on A) by FUNCT_2:8;
  then rng J c= relations_on A by FUNCT_2:92;
  then (rng(FO-pred_symbols(Al2) --> empty_rel(A))) \/
   (rng J) c= relations_on A & rng J2 c=(rng(FO-pred_symbols(Al2) -->
    empty_rel(A)))\/(rng J) by FUNCT_4:17,XBOOLE_1:8;
  then reconsider J2 as Function of FO-pred_symbols(Al2),relations_on A
   by A4,FUNCT_2:2,XBOOLE_1:1;
A5: J = J2|FO-pred_symbols(Al) by A3,FUNCT_4:23;
  for P2 being Element of FO-pred_symbols(Al2), r being Element of relations_on
   A st J2.P2 = r holds r = empty_rel(A) or the_arity_of P2 = the_arity_of r
  proof
    let P2 be Element of FO-pred_symbols(Al2), r be Element of relations_on A
    such that
A6:  J2.P2 = r;
    per cases;
    suppose P2 in FO-pred_symbols(Al);
      then consider P being FO-pred_symbol of Al such that
A7:    P = P2;
A8:   J.P = r by A3,A6,A7,FUNCT_4:13;
      7 + the_arity_of P2 = P`1 by A7,FO_LANG1:def 8 .= 7 + the_arity_of P
       by FO_LANG1:def 8;
      hence thesis by A8,FO_VALUA:def 5;
    end;
    suppose not P2 in FO-pred_symbols(Al);
      then not P2 in dom J by FUNCT_2:def 1;
      then J2.P2 = (FO-pred_symbols(Al2) --> empty_rel(A)).P2 by FUNCT_4:11
       .= empty_rel(A) by FUNCOP_1:7;
      hence thesis by A6;
    end;
  end;
  then reconsider J2 as interpretation of Al2,A by FO_VALUA:def 5;
  consider a being Element of A such that not contradiction;
  set v2 = (bound_FO-variables(Al2) --> a) +* v;
  v in Valuations_in(Al,A);
  then v in Funcs(bound_FO-variables(Al),A) by FO_VALUA:def 1;
  then
A9: dom v = bound_FO-variables(Al) & dom (bound_FO-variables(Al2) --> a) =
   bound_FO-variables(Al2) by FUNCOP_1:13,FUNCT_2:92;
  then dom v2=bound_FO-variables(Al)\/bound_FO-variables(Al2) by FUNCT_4:def 1;
  then
A10: dom v2 = bound_FO-variables(Al2) by Th4,XBOOLE_1:12;

  v in Valuations_in(Al,A);
  then v in Funcs(bound_FO-variables(Al),A) by FO_VALUA:def 1;
  then rng v c= A by FUNCT_2:92;
  then (rng (bound_FO-variables(Al2) --> a)) \/ (rng v) c= A &
   rng v2 c= (rng (bound_FO-variables(Al2) --> a)) \/ (rng v)
   by FUNCT_4:17, XBOOLE_1:8;
  then reconsider v2 as Function of bound_FO-variables(Al2),A
   by A10,FUNCT_2:2,XBOOLE_1:1;
A11: v = v2|bound_FO-variables(Al) by A9,FUNCT_4:23;
  v2 in Funcs(bound_FO-variables(Al2),A) by FUNCT_2:8;
  then reconsider v2 as Element of Valuations_in(Al2,A) by FO_VALUA:def 1;
  for p2 being Element of CFO-WFF(Al2) st p2 in THETA holds J2,v2 |= p2
  proof
    let p2 be Element of CFO-WFF(Al2) such that
A12:  p2 in THETA;
     consider p such that
A13:  p = p2 & p in PHI by A1,A12;
     J,v |= p by A2,A13,CALCL1_2:def 11;
     then J2,v2 |= Al2-Cast(p) by A5,A11,Th9;
     hence thesis by A13;
  end;
  then J2,v2 |= THETA by CALCL1_2:def 11;
  hence thesis;
end;

theorem Th24:
 for CHI being Subset of CFO-WFF(Al) st CHI c= PHI holds CHI is Consistent
proof
 let CHI be Subset of CFO-WFF(Al) such that
A1: CHI c= PHI;
 assume CHI is Inconsistent;
 then CHI |- ('not' VERUM(Al)) by GOEDCP_2:24;
 then ex f being FinSequence of CFO-WFF(Al) st rng f c= CHI &
  |- f^<*'not' VERUM(Al)*> by HENMOD_2:def 1;
 then ex f being FinSequence of CFO-WFF(Al) st rng f c= PHI &
  |- f^<*'not' VERUM(Al)*> by A1,XBOOLE_1:1;
 then PHI |- ('not' VERUM(Al)) by HENMOD_2:def 1;
 hence contradiction by GOEDCP_2:24;
end;

theorem
 PHI is Al2-Consistent
proof
  for PSI being Subset of CFO-WFF(Al2) st PHI = PSI holds PSI is Consistent
  proof
    let PSI be Subset of CFO-WFF(Al2) such that
A1:  PHI = PSI;
    for CHI being Subset of CFO-WFF(Al2) st CHI c= PSI & CHI is finite
     holds CHI is Consistent
    proof
      let CHI be Subset of CFO-WFF(Al2) such that
A2:    CHI c= PSI & CHI is finite;
      reconsider CHI as finite Subset of CFO-WFF(Al) by A1,A2,XBOOLE_1:1;
      consider Al1 being countable FO-alphabet such that
A4:    CHI is finite Subset of CFO-WFF(Al1) & Al is Al1-extending by Th21;
      reconsider Al as Al1-extending FO-alphabet by A4;
      reconsider CHI as finite Subset of CFO-WFF(Al);
      reconsider PHI as Consistent Subset of CFO-WFF(Al);
      reconsider CHI as Consistent Subset of CFO-WFF(Al) by A1,A2,Th24;
      CHI is Al1-Consistent by A4,Th19;
      then reconsider CHI as Consistent Subset of CFO-WFF(Al1) by A4,Def2;
      still_not-bound_in CHI is finite by Th22;
      then consider CZ being Consistent Subset of CFO-WFF(Al1), JH being
       Henkin_interpretation of CZ such that
A5:    JH,valH(Al1) |= CHI by GOEDCP_2:34;
      Al1 c= Al & Al c= Al2 by Def1;
      then Al1 c= Al2 by XBOOLE_1:1;
      then reconsider Al2 as Al1-extending FO-alphabet by Def1;
      consider CHI2 being Subset of CFO-WFF(Al2) such that
A6:    CHI = CHI2;
      consider A being non empty set, J2 being interpretation of Al2,A, v2
       being Element of Valuations_in(Al2,A) such that
A7:    J2,v2 |= CHI2 by A5,A6,Th23;
      thus thesis by A6,A7,HENMOD_2:12;
    end;
    hence thesis by HENMOD_2:7;
  end;
  hence thesis by Def2;
end;
```

8.2 GOEDCPOC

```
environ

 vocabularies NUMBERS, SUBSET_1, FO_LANG1, CFO_LANG, XBOOLE_0,
      FO_VALUA, FINSEQ_1, HENMOD_2, CFO_THE1, XBOOLEAN, BVFUNC_2, FUNCT_1,
      ORDINAL4, ARYTM_3, RELAT_1, CARD_1, XXREAL_0, TARSKI, ZF_MODEL, REALSET1,
      ARYTM_1, CARD_3, ZFMISC_1, FINSET_1, MCART_1, NAT_1, ORDINAL1, GOEDCP_2,
      MARGREL1, FUNCT_2, FO_TRANS, GOEDCPOC;
 notations TARSKI, XBOOLE_0, ZFMISC_1, SUBSET_1, XCMPLX_0, XXREAL_0, NAT_1,
      CARD_1, CARD_3,FINSEQ_1, RELAT_1, FO_LANG1, FO_LANG2, NUMBERS, CFO_THE1,
      CFO_LANG, FUNCT_1, FINSET_1, FO_VALUA, RELSET_1, FUNCT_2, CFO_SIM1,
      DOMAIN_1, MCART_1, SUBST1_2, SUBLEM_2, SUBST2_2, CALCL1_2, HENMOD_2,
      ORDINAL1, GOEDCP_2, MARGREL1, FO_LANG3, FUNCT_4, FUNCOP_1, COHSP_1,
      FO_TRANS;
 constructors SETFAM_1, DOMAIN_1, XXREAL_0, NAT_1, NAT_D, FINSEQ_2, FO_LANG1,
      CFO_THE1, CFO_SIM1, SUBST2_2, CALCL1_2, HENMOD_2, CARD_3, RELSET_1,
      CARD_1, WELLORD2, GOEDCP_2, FO_VALUA, MARGREL1, CFO_LANG, FO_LANG3,
      FUNCT_4, FUNCOP_1, COHSP_1, FO_TRANS;
 registrations SUBSET_1, RELAT_1, FUNCT_1, ORDINAL1, XXREAL_0, XREAL_0,
      HENMOD_2, FINSEQ_1, FINSET_1, CARD_3, XBOOLE_0, FO_LANG1, CFO_LANG,
      MARGREL1, FO_VALUA, CARD_1, GOEDCP_2, FUNCT_4, FUNCOP_1, FO_TRANS;
 requirements REAL, NUMERALS, SUBSET, BOOLE, ARITHM;
 definitions TARSKI, XBOOLE_0, GOEDCP_2;
 theorems TARSKI, FUNCT_1, MCART_1, XBOOLE_0, XBOOLE_1, FO_LANG1, ZFMISC_1,
      FO_LANG2, HENMOD_2, CALCL1_2, SUBLEM_2, NAT_1, FINSEQ_1, FO_VALUA,
      FUNCT_2, XXREAL_0, ORDINAL1, CARD_1, GOEDCP_2, ORDERS_1, COHSP_1,
      FO_TRANS;
 schemes NAT_1, FUNCT_1, CFO_LANG, FINSET_1;

begin

reserve Al for FO-alphabet,
     PHI for Consistent Subset of CFO-WFF(Al),
     PSI for Subset of CFO-WFF(Al),
     p,q,r,s for Element of CFO-WFF(Al),
     A for non empty set,
     J for interpretation of Al,A,
     v for Element of Valuations_in(Al,A),
     m,n,i,j,k for Element of NAT,
     l for CFO-variable_list of k,Al,
     P for FO-pred_symbol of k,Al,
     x,y for bound_FO-variable of Al,
     z for FO-symbol of Al,
     Al2 for Al-extending FO-alphabet;

definition
  let Al;
  let PHI be Subset of CFO-WFF(Al);
  attr PHI is satisfiable means :Def1:
  ex A,J,v st J,v |= PHI;
end;

reserve J2 for interpretation of Al2,A,
        Jp for interpretation of Al,A,
        v2 for Element of Valuations_in(Al2,A),
        vp for Element of Valuations_in(Al,A);

theorem Th1:
 ex s being set st for p,x holds not [s,[x,p]] in FO-symbols(Al)
proof
  assume
A1: for s being set holds ex p,x st [s,[x,p]] in FO-symbols(Al);
  for s being set holds s in union union FO-symbols(Al)
  proof
    let s be set;
    consider p,x such that
A2:  [s,[x,p]] in FO-symbols(Al) by A1;
A3: {s} in {{s,[x,p]},{s}} by TARSKI:def 2;
A4: s in {s} by TARSKI:def 1;
    {{s,[x,p]},{s}} c= union FO-symbols(Al) by A2,ZFMISC_1:74;
    then {s} c= union union FO-symbols(Al) by A3,ZFMISC_1:74;
    hence thesis by A4;
  end;
  then union union FO-symbols(Al) in union union FO-symbols(Al) &
  for X being set holds not X in X;
  hence contradiction;
end;

definition
  let Al;
  mode free_symbol of Al -> set means :Def2:
  it is set & for p,x holds not [it,[x,p]] in FO-symbols(Al);
  existence
  proof
    consider s being set such that
A1:  for p,x holds not [s,[x,p]] in FO-symbols(Al) by Th1;
    take s;
    thus thesis by A1;
  end;
end;

definition
  let Al;
  func FCEx(Al) -> Al-extending FO-alphabet equals
    [:NAT, FO-symbols(Al) \/
     { [the free_symbol of Al,[x,p]] : not contradiction} :];
  coherence
  proof
    set Al2 = [:NAT, FO-symbols(Al) \/
     { [the free_symbol of Al,[x,p]] : not contradiction} :];
    set X = FO-symbols(Al)\/
     { [the free_symbol of Al,[x,p]] : not contradiction};
    Al2 is non empty set & NAT c= X & Al2 = [: NAT, X :]
     by FO_LANG1:3,XBOOLE_1:10;
    then reconsider Al2 as FO-alphabet by FO_LANG1:def 1;
    [: NAT, FO-symbols(Al) :] c= [:NAT, FO-symbols(Al):] \/
     [: NAT, { [the free_symbol of Al,[x,p]] : not contradiction} :]
    by XBOOLE_1:7;
    then [: NAT, FO-symbols(Al) :] c= [:NAT, X :] by ZFMISC_1:97;
    then Al c= Al2 by FO_LANG1:5;
    hence thesis by FO_TRANS:def 1;
  end;
end;

definition
  let Al;
  let p;
  let x;
  func Example_of(p,x) ->
```

```
    bound_FO-variable of FCEx(Al) equals
   [4,[the free_symbol of Al,[x,p]]];
   coherence
   proof
     set Al2 = FCEx(Al);
     [: NAT, FO-symbols(Al2) :] = [:NAT, FO-symbols(Al) \/
      { [the free_symbol of Al,[y,q]] : not contradiction} :] by FO_LANG1:5;
     then
A1:  FO-symbols(Al2) = FO-symbols(Al) \/
      { [the free_symbol of Al,[y,q]] : not contradiction} by ZFMISC_1:110;
     [the free_symbol of Al,[x,p]] in
      { [the free_symbol of Al,[y,q]] : not contradiction };
     then
A2: [the free_symbol of Al,[x,p]] in FO-symbols(Al2) by A1, XBOOLE_0:def 3;
     4 in {4} by TARSKI:def 1;
     then [4, [the free_symbol of Al,[x,p]]] in [:{4},FO-symbols(Al2):]
       by A2,ZFMISC_1:87;
     hence thesis by FO_LANG1:def 4;
   end;
end;

definition
  let Al;
  let p;
  let x;
  func Example_Formula_of(p,x) -> Element of CFO-WFF(FCEx(Al))
   equals
  ('not' Ex((FCEx(Al))-Cast(x),(FCEx(Al))-Cast(p))) 'or'
  (((FCEx(Al))-Cast(p)).((FCEx(Al))-Cast(x),Example_of(p,x)));
  coherence;
end;

definition
  let Al;
  func Example_Formulae_of(Al) -> Subset of CFO-WFF(FCEx(Al)) equals
  { Example_Formula_of(p,x) : not contradiction };
  coherence
  proof
    for z being set st z in { Example_Formula_of(p,x) : not contradiction }
     holds z in CFO-WFF(FCEx(Al))
    proof
      let z be set such that
A1:   z in { Example_Formula_of(p,x) : not contradiction };
      ex p,x st z = Example_Formula_of(p,x) by A1;
      hence thesis;
    end;
    hence thesis by TARSKI:def 3;
  end;
end;

theorem Th2:
 for k being Element of NAT st k > 0 holds ex F being k-element FinSequence  st
 (for n being Nat st n <= k & 1 <= n holds F.n is FO-alphabet) &
 (F.1  = Al) & (for n being Nat st n < k & 1 <= n holds
 ex Al2 being FO-alphabet st F.n = Al2 & F.(n+1) = FCEx(Al2))
proof
  defpred A[Element of NAT] means $1 > 0 implies (
   ex F being $1-element FinSequence st
   ( for n being Nat st n <= $1 & 1 <= n holds F.n is FO-alphabet ) &
   ( F.1 = Al ) & ( for n being Nat st n < $1 & 1 <= n holds
    ex Al2 being FO-alphabet st F.n = Al2 & F.(n+1) = FCEx(Al2)) );
A1: for k being Element of NAT st A[k] holds A[k+1]
  proof
    let k be Element of NAT;
    assume
A2: A[k];
    per cases;
    suppose
A3:   k = 0;
A4:   <*Al*> is 1-element FinSequence & <*Al*>.1 = Al by FINSEQ_1:40;
A5:   for n being Nat st
      n < 1 & 1 <= n holds ex Al2 being FO-alphabet
      st <*Al*>.n = Al2  &  <*Al*>.(n+1) = FCEx(Al2);
      for n being Nat st n<=1 & 1<=n holds <*Al*>.n is FO-alphabet
       by A4,XXREAL_0:1;
      hence thesis by A3,A4,A5;
    end;
    suppose
A6:   k <> 0;
      then consider F being k-element FinSequence such that
A7:    ( for n being Nat st n <= k & 1 <= n holds F.n is FO-alphabet ) &
       ( F.1  = Al ) & ( for n being Nat st n < k & 1 <= n holds ex Al2 being
       FO-alphabet st F.n = Al2 & F.(n+1) = FCEx(Al2)) by A2,NAT_1:2;
      set K = F.k;
      K is FO-alphabet
      proof
        per cases;
        suppose k = 1;
          hence thesis by A7;
        end;
        suppose
A8:       k <> 1;
          consider j being Nat such that
A9:        k = j+1 by NAT_1:6, A6;
          j <> 0 by A8,A9;
          then j >= 1 & j < k by A9,NAT_1:25,19;
          then ex Al2 being FO-alphabet st F.j = Al2 & F.(k) = FCEx(Al2)
            by A7,A9;
          hence thesis;
        end;
      end;
      then reconsider K as FO-alphabet;
      set K2 = <*FCEx(K)*>;
      set F2 = F^K2;
      reconsider F2 as (k+1)-element FinSequence;
A10:  1 <= k & len F = k by A6,NAT_1:25,CARD_1:def 7;
A11:  for n being Nat st n < k & 1 <= n holds
       ex Al2 being FO-alphabet st F2.n = Al2 & F2.(n+1) = FCEx(Al2)
      proof
        let n be Nat such that
A12:      n < k & 1 <= n;
        consider Al2 being FO-alphabet such that
A13:     F.n = Al2 & F.(n+1) = FCEx(Al2) by A7,A12;
        1 <= n+1 & n+1 <= k & k = len F by A12,NAT_1:13,CARD_1:def 7;
        then F2.n = F.n & F2.(n+1) = F.(n+1) by A12,FINSEQ_1:64;
        hence thesis by A13;
      end;
A15:  K is FO-alphabet &  F2.k = K by A10,FINSEQ_1:64;
A16:  for n being Nat st n < k+1 & 1 <= n holds
       ex Al2 being FO-alphabet st F2.n = Al2 & F2.(n+1) = FCEx(Al2)
      proof
        let n be Nat such that
A17:      n < k+1 & 1 <= n;
        per cases;
        suppose n <> k;
          hence thesis by A11,A17,NAT_1:22;
        end;
        suppose n = k;
          hence thesis by A10,A15,FINSEQ_1:42;
        end;
      end;
A18:  for n being Nat st n <= k+1 & 1 <= n holds F2.n is FO-alphabet
      proof
        let n be Nat such that
A19:      n <= k+1 & 1 <= n;
        per cases;
        suppose n <> k+1;
          then n <= k by A19,NAT_1:8;
          then F2.n = F.n & F.n is FO-alphabet by A7,A10,A19,FINSEQ_1:64;
          hence thesis;
        end;
        suppose n = k +1;
          hence thesis by A10,FINSEQ_1:42;
        end;
      end;
      F2.1 = Al by A7,A10,FINSEQ_1:64;
      hence thesis by A16,A18;
    end;
  end;
A20: A[0];
  for n being Element of NAT holds A[n] from NAT_1:sch 1(A20,A1);
  hence thesis;
end;

definition
  let Al;
  let k be Nat;
  mode FCEx-Sequence of Al,k -> (k+1)-element FinSequence means :Def7:
    ( for n being Nat st n <= (k+1) & 1 <= n holds it.n is FO-alphabet ) &
    ( it.1 = Al ) & ( for n being Nat st n < (k+1) & 1 <= n holds
      ex Al2 being FO-alphabet st it.n = Al2 & it.(n+1) = FCEx(Al2) );
  existence by NAT_1:5,Th2;
end;

theorem Th3:
  for k being Nat for S being FCEx-Sequence of Al,k holds
  S.(k+1) is FO-alphabet
proof
```

```
  let k be Nat;
  let S be FCEx-Sequence of Al,k;
  0 < 0 + (k + 1) by NAT_1:5;
  then 1 <= k + 1 & k + 1 <= k + 1 by NAT_1:19;
  hence thesis by Def7;
end;

theorem Th4:
  for k being Nat for S being FCEx-Sequence of Al,k holds
   S.(k+1) is Al-extending FO-alphabet
proof
  let k be Nat;
  let S be FCEx-Sequence of Al,k;
  set Al2 = S.(k+1);
  reconsider Al2 as FO-alphabet by Th3;
  Al c= Al2
  proof
    let x be set;
    assume
A2: x in Al;
    defpred A[Element of NAT] means $1 < k+1 implies x in S.($1+1);
A3: A[0] by A2,Def7;
A4: for l being Element of NAT st A[l] holds A[l+1]
    proof
      let l be Element of NAT;
      assume
A5:   A[l];
      assume
A6:   l+1 < k+1;
A8:   for n being Nat st n+1 < (k+1) & 1 <= n+1 holds
      ex Al2 being FO-alphabet st S.(n+1) = Al2 & S.(n+2) = FCEx(Al2)
      proof
        let n be Nat such that
A9:     n + 1 < k + 1 &  1 <= n+1;
        set m = n +1;
        ex Al2 being FO-alphabet st S.(m) = Al2 & S.(m+1) = FCEx(Al2)
         by Def7,A9;
        hence thesis;
      end;
      0 < 0 + (l + 1) by NAT_1:5;
      then 1 <= l + 1 & l + 1 < k + 1 by NAT_1:19,A6;
      then consider Al2 being FO-alphabet such that
A10:   S.(l+1) = Al2 & S.(l+2) = FCEx(Al2) by A8;
      S.(l+1) c= S.(l+2) by A10,FO_TRANS:def 1;
      hence thesis by A5,A6,NAT_1:16,XXREAL_0:2;
    end;
    reconsider k as Element of NAT by ORDINAL1:def 12;
    for n being Element of NAT holds A[n] from NAT_1:sch 1(A3,A4);
    then k < k+1 implies x in S.(k+1);
    hence thesis by NAT_1:13;
  end;
  hence thesis by FO_TRANS:def 1;
end;

definition
  let Al;
  let k be Nat;
  func k-th_FCEx(Al) -> Al-extending FO-alphabet equals
    (the FCEx-Sequence of Al,k).(k+1);
  coherence by Th4;
end;

definition
  let Al;
  let PHI;
  mode EF-Sequence of Al,PHI -> Function means :Def9:
    dom it = NAT & it.0 = PHI & for n being Nat holds
     it.(n+1) = it.n \/ Example_Formulae_of(n-th_FCEx(Al));
  existence
  proof
    deffunc F1() = PHI;
    deffunc F2(Nat,set) = $2 \/ Example_Formulae_of($1-th_FCEx(Al));
    ex f being Function st dom f = NAT & f.0 = F1() &
        for n being Nat holds f.(n+1) = F2(n,f.n)
     from NAT_1:sch 11;
    hence thesis;
  end;
end;

theorem Th5:
 FCEx(k-th_FCEx(Al)) = (k+1)-th_FCEx(Al)
proof
    k+1+1 <= k+1+1 & 0 < 0 + (k + 1) by NAT_1:5;
  then k+1 < k+2 & 1 <= k+1 by NAT_1:19;
  then consider Al2 being FO-alphabet such that
A2: (the FCEx-Sequence of Al,k+1).(k+1) = Al2 &
    (the FCEx-Sequence of Al,k+1).(k+2) = FCEx(Al2) by Def7;
  set X = (the FCEx-Sequence of Al,k+1).(k+1);
A3: X = k-th_FCEx(Al)
  proof
    defpred A[Element of NAT] means 0 < $1 & $1 <= k+1 implies (
      (the FCEx-Sequence of Al,k).$1 = (the FCEx-Sequence of Al,k+1).$1);
A4: A[0];
A5: for n being Element of NAT st A[n] holds A[n+1]
    proof
      let n be Element of NAT;
      assume
A6:   A[n];
      per cases;
      suppose
A7:     (n+1) <= k+1;
        per cases;
        suppose
A8:       n=0;
          (the FCEx-Sequence of Al,k).1 = Al by Def7
           .= (the FCEx-Sequence of Al,k+1).1 by Def7;
          hence A[n+1] by A8;
        end;
        suppose
A9:       n <> 0;
A10:      0 < 0 + n & n <= n+1 by A9,NAT_1:11;
          0 < 0 + n by A9,NAT_1:2;
          then
A12:      1 <= n by NAT_1:19;
A13:      n < k+1 by A7,NAT_1:13;
          then n < k+1+1 by NAT_1:13;
          then
          consider Al3 being FO-alphabet such that
A15:       (the FCEx-Sequence of Al,k+1).n = Al3 &
           (the FCEx-Sequence of Al,k+1).(n+1) = FCEx(Al3) by A12,Def7;
          consider Al4 being FO-alphabet such that
A16:       (the FCEx-Sequence of Al,k).n = Al4 &
           (the FCEx-Sequence of Al,k).(n+1) = FCEx(Al4) by A12,A13,Def7;
          thus A[n+1] by A6,A10,A15,A16,XXREAL_0:2;
        end;
      end;
      suppose not n+1 <= k+1;
        hence A[n+1];
      end;
    end;
    for n being Element of NAT holds A[n] from NAT_1:sch 1(A4,A5);
    hence thesis by NAT_1:5;
  end;
  reconsider X as FO-alphabet by A2;
  thus thesis by A2,A3;
end;

theorem Th6:
 for k for n st n <= k holds n-th_FCEx(Al) c= k-th_FCEx(Al)
proof
  let k;
  defpred P[Element of NAT] means $1 <= k implies ex j st j = k-$1 &
   j-th_FCEx(Al) c= k-th_FCEx(Al);
A1: P[0];
A2: for n st P[n] holds P[n+1]
  proof
    let n such that
A3:  P[n];
    per cases;
    suppose
A4:   n+1 <= k;
      then consider j such that
A5:    j = k-n & j-th_FCEx(Al) c= k-th_FCEx(Al) by A3,NAT_1:13;
      set j2=k-(n+1);
      reconsider j2 as Element of NAT by A4,NAT_1:21;
      FCEx(j2-th_FCEx(Al)) = j-th_FCEx(Al) by A5,Th5;
      then j2-th_FCEx(Al) c= j-th_FCEx(Al) by FO_TRANS:def 1;
      hence thesis by A5,XBOOLE_1:1;
    end;
    suppose not n+1 <= k;
      hence thesis;
    end;
  end;
A6: for n holds P[n] from NAT_1:sch 1(A1,A2);
  let n such that
A7: n <= k;
  set n2 = k - n;
  reconsider n2 as Element of NAT by A7,NAT_1:21;
  k = n + n2;
  then consider n3 being Element of NAT such that
A8: n3 = k-n2 & n3-th_FCEx(Al) c= k-th_FCEx(Al) by A6,NAT_1:11;
  thus thesis by A8;
end;

definition
  let Al;
  let PHI;
  let k;
  func k-th_EF(Al,PHI) -> Subset of CFO-WFF(k-th_FCEx(Al)) equals
  (the EF-Sequence of Al,PHI).k;
  coherence
  proof
    defpred P[Element of NAT] means (the EF-Sequence of Al,PHI).$1
      is Subset of CFO-WFF($1-th_FCEx(Al));
A1: P[0]
    proof
      0-th_FCEx(Al) = Al by Def7;
      hence thesis by Def9;
    end;
A3: for n being Element of NAT holds P[n] implies P[n+1]
    proof
      let n be Element of NAT;
      assume
A4:   P[n];
      set E = (the EF-Sequence of Al,PHI).(n+1);
      set S = (the EF-Sequence of Al,PHI).n;
A5:   Example_Formulae_of(n-th_FCEx(Al)) is Subset of
       CFO-WFF(FCEx(n-th_FCEx(Al))) & FCEx(n-th_FCEx(Al)) = (n+1)-th_FCEx(Al)
       by Th5;
      S is Subset of CFO-WFF(FCEx(n-th_FCEx(Al)))
      proof
        for p being set st p in S holds p in CFO-WFF(FCEx(n-th_FCEx(Al)))
        proof
          let p be set;
          assume
A6:        p in S;
          reconsider p as Element of CFO-WFF(n-th_FCEx(Al)) by A4,A6;
          p is Element of CFO-WFF(FCEx(n-th_FCEx(Al))) by FO_TRANS:7;
          hence thesis;
        end;
        hence thesis by TARSKI:def 3;
      end;
      then S \/ Example_Formulae_of(n-th_FCEx(Al))c= CFO-WFF((n+1)-th_FCEx(Al))
       by A5,XBOOLE_1:8;
      hence thesis by Def9;
    end;
    for n being Element of NAT holds P[n] from NAT_1:sch 1(A1,A3);
    hence thesis;
  end;
end;

theorem Th7:
 for r,s,x holds Al2-Cast(r 'or' s) = Al2-Cast(r) 'or' Al2-Cast(s) &
 Al2-Cast(Ex(x,r)) = Ex(Al2-Cast(x),Al2-Cast(r))
proof
  let r,s,x;
A1: Al2-Cast('not' r) = 'not' Al2-Cast(r) &
   Al2-Cast('not' s) = 'not' Al2-Cast(s) by FO_TRANS:8;
  thus Al2-Cast(r 'or' s)
    = Al2-Cast('not' ('not' r '&' 'not' s)) by FO_LANG2:def 3
   .= 'not' Al2-Cast('not' r '&' 'not' s) by FO_TRANS:8
   .= 'not' ('not' Al2-Cast(r) '&' 'not' Al2-Cast(s)) by A1,FO_TRANS:8
   .= Al2-Cast(r) 'or' Al2-Cast(s) by FO_LANG2:def 3;
  thus Al2-Cast(Ex(x,r)) = Al2-Cast('not' All(x,'not' r)) by FO_LANG2:def 5
   .= 'not' Al2-Cast(All(x,'not' r)) by FO_TRANS:8
   .= 'not' All(Al2-Cast(x),Al2-Cast('not' r)) by FO_TRANS:8
   .= 'not' All(Al2-Cast(x),'not' Al2-Cast(r)) by FO_TRANS:8
   .= Ex(Al2-Cast(x),Al2-Cast(r)) by FO_LANG2:def 5;
end;


theorem Th8:
 for p,q,A,J,v holds (J,v |= p or J,v |= q) iff J,v |= p 'or' q
proof
  let p,q,A,J,v;
  thus (J,v |= p or J,v |= q) implies J,v |= p 'or' q
  proof
    assume J,v |= p or J,v |= q;
    then not J,v |= 'not' p or not J,v |= 'not' q by FO_VALUA:17;
    then not J,v |= 'not' p '&' 'not' q by FO_VALUA:18;
    then J,v |= 'not' ('not' p '&' 'not' q) by FO_VALUA:17;
    hence thesis by FO_LANG2:def 3;
 end;
 thus J,v |= p 'or' q implies (J,v |= p or J,v |= q)
 proof
   assume J,v |= p 'or' q;
   then J,v |= 'not' ('not' p '&' 'not' q) by FO_LANG2:def 3;
   then not J,v |= 'not' p '&' 'not' q by FO_VALUA:17;
   then not J,v |= 'not' p or not J,v |= 'not' q by FO_VALUA:18;
   hence J,v |= p or J,v |= q by FO_VALUA:17;
 end;
end;

:: Ebbinghaus Lemma 5.3.4
theorem Th9:
  PHI \/ Example_Formulae_of(Al) is Consistent Subset of CFO-WFF(FCEx(Al))
proof
  reconsider Al2 = FCEx(Al) as Al-extending FO-alphabet;
  for s being set st s in CFO-WFF(Al) holds s in CFO-WFF(Al2)
  proof
    let s be set;
    assume s in CFO-WFF(Al);
    then reconsider s as Element of CFO-WFF(Al);
    s is Element of CFO-WFF(Al2) by FO_TRANS:7;
    hence thesis;
  end;
  then
A1: CFO-WFF(Al) c= CFO-WFF(Al2) by TARSKI:def 3;
  then PHI c= CFO-WFF(Al2) & Example_Formulae_of(Al) c= CFO-WFF(Al2)
   by XBOOLE_1:1;
  then reconsider B = PHI \/ Example_Formulae_of(Al) as Subset of CFO-WFF(Al2)
   by XBOOLE_1:8;
  B is Consistent
  proof
    assume B is Inconsistent;
    then consider CHI2 being Subset of CFO-WFF(Al2) such that
A2:  CHI2 c= B & CHI2 is finite & CHI2 is Inconsistent by HENMOD_2:7;
    reconsider CHI2 as finite Subset of CFO-WFF(Al2) by A2;
    consider Al1 being countable FO-alphabet such that
A3:  CHI2 is finite Subset of CFO-WFF(Al1) & Al2 is Al1-extending
     by FO_TRANS:21;
    reconsider Al2 as Al1-extending FO-alphabet by A3;
    consider CHI1 being Subset of CFO-WFF(Al1) such that
A4:  CHI1 = CHI2 by A3;
    reconsider CHI1 as finite Subset of CFO-WFF(Al1) by A4;
    set PHI1 = CHI1 /\ PHI;
    PHI1 c= CHI1 & CHI1 c= CFO-WFF(Al1) by XBOOLE_1:18;
    then reconsider PHI1 as Subset of CFO-WFF(Al1) by XBOOLE_1:1;
    reconsider Al2 as Al-extending FO-alphabet;
    PHI is Subset of CFO-WFF(Al2) by A1,XBOOLE_1:1;
    then consider PHIp being Subset of CFO-WFF(Al2) such that
A7:  PHIp = PHI;
    set PHI2 = CHI2 /\ PHIp;
    reconsider PHI2 as Subset of CFO-WFF(Al2);
    PHI1 is Consistent
    proof
      reconsider Al2 as Al1-extending FO-alphabet;
      PHI is Al2-Consistent by FO_TRANS:25;
      then PHIp is Consistent & PHI2 c= PHIp by A7,FO_TRANS:def 2,XBOOLE_1:18;
      then PHI2 is Consistent & PHI2 = PHI1 by A4,A7,FO_TRANS:24;
      then PHI2 is Al1-Consistent by FO_TRANS:19;
      hence PHI1 is Consistent by A4,A7,FO_TRANS:def 2;
    end;
    then reconsider PHI1 as Consistent Subset of CFO-WFF(Al1);
    still_not-bound_in PHI1 is finite by FO_TRANS:22;
    then consider CZ being Consistent Subset of CFO-WFF(Al1), JH being
     Henkin_interpretation of CZ such that
A9:  JH,valH(Al1) |= PHI1 by GOEDCP_2:34;
    consider A2 being non empty set, J2 being interpretation of Al2,A2,
     v2 being Element of Valuations_in (Al2,A2) such that
A10: J2,v2 |= PHI2 by A4,A7,A9,FO_TRANS:23;
    set Ex2 = CHI2 /\ Example_Formulae_of(Al);
    reconsider Ex2 as Subset of CFO-WFF(Al2);
A11: CHI2 = CHI2 /\ (PHIp \/ Example_Formulae_of(Al)) by A2,A7,XBOOLE_1:28
     .= PHI2 \/ Ex2 by XBOOLE_1:23;
    ex A being non empty set, J being interpretation of Al2,A, v being Element
     of Valuations_in(Al2,A) st J,v |= PHI2 \/ Ex2
    proof
      defpred P[set] means ex A being non empty set, J being interpretation of
       Al2,A, v being Element of Valuations_in(Al2,A), Ex3 being Subset of
       CFO-WFF(Al2) st Ex3 = $1 & J,v |= PHI2 \/ Ex3;
A12:  Ex2 is finite;
A13:  P[{}]
      proof
        reconsider em = {} as Subset of CFO-WFF(Al2) by XBOOLE_1:2;
        J2,v2 |= PHI2 \/ em by A10;
        hence thesis;
      end;
A14:  for b,B being set st b in Ex2 & B c= Ex2 & P[B] holds P[B \/ {b}]
      proof
        let b,B be set such that
A15:    b in Ex2 & B c= Ex2 & P[B];
        reconsider B as Subset of CFO-WFF(Al2) by A15;
        consider A being non empty set, J being interpretation of Al2,A, v
         being Element of Valuations_in(Al2,A) such that
A16:     J,v |= PHI2 \/ B by A15;
        Ex2 c= Example_Formulae_of(Al) by XBOOLE_1:18;
        then b in Example_Formulae_of(Al) by A15;
        then consider p,x such that
A17:     b = Example_Formula_of(p,x);
        set fc = Example_of(p,x);
        set x2 = Al2-Cast(x);
        set p2 = Al2-Cast(p);
        reconsider fc,x2 as bound_FO-variable of Al2;
        reconsider p2 as Element of CFO-WFF(Al2);
        reconsider b as Element of CFO-WFF(Al2) by A17;
A20:    J,v |= b implies thesis
        proof
          assume
A21:       J,v |= b;
          for q2 being Element of CFO-WFF(Al2) st q2 in PHI2 \/ (B \/ {b})
           holds J,v |= q2
          proof
            let q2 be Element of CFO-WFF(Al2);
            assume q2 in PHI2 \/ (B \/ {b});
            then q2 in (PHI2 \/ B) \/ {b} by XBOOLE_1:4;
            then
A22:        q2 in (PHI2 \/ B) or q2 in {b} by XBOOLE_0:def 3;
            per cases;
            suppose q2 in (PHI2 \/ B);
              hence thesis by A16,CALCL1_2:def 11;
            end;
            suppose not q2 in (PHI2 \/ B);
              hence thesis by A21,A22,TARSKI:def 1;
            end;
          end;
          then J,v |= PHI2 \/ (B \/ {b}) by CALCL1_2:def 11;
          hence thesis;
        end;
        per cases;
        suppose not J,v |= Ex(x2,p2);
          then J,v |= 'not' Ex(x2,p2) by FO_VALUA:17;
          hence thesis by A17,A20,Th8;
        end;
        suppose J,v |= Ex(x2,p2);
          then consider a being Element of A such that
A23:       J,v.(x2|a) |= p2 by GOEDCP_2:9;
          reconsider vp = v.(fc|a) as Element of Valuations_in(Al2,A);
A24:      for p2 being Element of CFO-WFF(Al2) st p2 is Element of CFO-WFF(Al)
          holds v|(still_not-bound_in p2) = vp|(still_not-bound_in p2)
          proof
            let p2 be Element of CFO-WFF(Al2);
            assume p2 is Element of CFO-WFF(Al);
            then consider pp being Element of CFO-WFF(Al) such that
A25:         pp = p2;
            not [the free_symbol of Al,[x,p]] in FO-symbols(Al) by Def2;
            then not [4,[the free_symbol of Al,[x,p]]] in
             [:{4},FO-symbols(Al):] by ZFMISC_1:87;
            then not fc in bound_FO-variables(Al) by FO_LANG1:def 4;
            then not fc in still_not-bound_in pp;
            then not fc in still_not-bound_in Al2-Cast(pp) by FO_TRANS:13;
            then not fc in still_not-bound_in p2 by A25,FO_TRANS:def 3;
            hence v|(still_not-bound_in p2) = vp|(still_not-bound_in p2)
             by CALCL1_2:26;
          end;
          p2 = p by FO_TRANS:def 3;
          then v|(still_not-bound_in p2) = vp|(still_not-bound_in p2) by A24;
          then v.(x2|a)|(still_not-bound_in p2) =
           vp.(x2|a)|(still_not-bound_in p2) by SUBLEM_2:64;
          then J,vp.(x2|a) |= p2 & vp.fc = a by A23,SUBLEM_2:49,68;
          then
A27:      J,vp |= p2.(x2,fc) by CALCL1_2:24;
          for q2 being Element of CFO-WFF(Al2) st q2 in PHI2 \/ (B \/ {b})
           holds J,vp |= q2
          proof
            let q2 be Element of CFO-WFF(Al2);
            assume q2 in PHI2 \/ (B \/ {b});
            then q2 in (PHI2 \/ B) \/ {b} by XBOOLE_1:4;
            then
A28:        q2 in (PHI2 \/ B) or q2 in {b} by XBOOLE_0:def 3;
```

```
            per cases;
            suppose
A29:          q2 in PHI2;
              then
A30:           q2 in PHI2 \/ B by XBOOLE_0:def 3;
              PHI2 c= PHI & PHI c= CFO-WFF(Al) by A7,XBOOLE_1:18;
              then PHI2 c= CFO-WFF(Al) by XBOOLE_1:1;
              then v|(still_not-bound_in q2) = vp|(still_not-bound_in q2)
               & J,v |= q2 by A16,A24,A29,A30,CALCL1_2:def 11;
              hence J,vp |= q2 by SUBLEM_2:68;
            end;
            suppose
A31:          q2 in B;
              B c= Example_Formulae_of(Al) by A15,XBOOLE_1:18;
              then q2 in Example_Formulae_of(Al) by A31;
              then consider r,y such that
A32:           q2 = Example_Formula_of(r,y);
              set fcr = Example_of(r,y);
              set y2 = Al2-Cast(y);
              set r2 = Al2-Cast(r);
              reconsider fcr,y2 as bound_FO-variable of Al2;
              reconsider r2 as Element of CFO-WFF(Al2);
              per cases;
              suppose fcr = fc;
                then [the free_symbol of Al,[x,p]] =
                 [the free_symbol of Al,[y,r]] by ZFMISC_1:27;
                then [x,p] = [y,r] by ZFMISC_1:27;
                then r = p & x = y by ZFMISC_1:27;
                hence thesis by A27,A32,Th8;
              end;
              suppose
A35:            not fcr = fc;
                q2 in PHI2 \/ B by A31,XBOOLE_0:def 3;
                then
A36:            J,v |= q2 by A16,CALCL1_2:def 11;
                per cases;
                suppose
A37:              J,v |= 'not' Ex(y2,r2);
                  set fml = 'not' Ex(y2,r2);
                  'not' Ex(y,r) = Al2-Cast('not' Ex(y,r)) by FO_TRANS:def 3
                   .= 'not' Al2-Cast(Ex(y,r)) by FO_TRANS:8
                   .= fml by Th7;
                   then v|(still_not-bound_in fml) =
                   vp|(still_not-bound_in fml) & J,v |= fml by A24,A37;
                   then J,vp |= fml by SUBLEM_2:68;
                   hence J,vp |= q2 by A32,Th8;
                end;
                suppose not J,v |= 'not' Ex(y2,r2);
                  then J,v |= r2.(y2,fcr) by A32,A36,Th8;
                  then consider a2 being Element of A such that
A38:               v.fcr = a2 & J,v.(y2|a2) |= r2 by CALCL1_2:24;
A39:              vp.fcr = a2 by A35,A38,SUBLEM_2:48;
                  not [the free_symbol of Al,[x,p]] in FO-symbols(Al)
                   by Def2;
                  then not [4,[the free_symbol of Al,[x,p]]] in
                   [:{4},FO-symbols(Al):] by ZFMISC_1:87;
                  then not fc in bound_FO-variables(Al) by FO_LANG1:def 4;
                  then not fc in still_not-bound_in r;
                  then not fc in still_not-bound_in r2 by FO_TRANS:13;
                  then v.(fc|a)|still_not-bound_in r2 =
                   v|still_not-bound_in r2 by CALCL1_2:26;
                  then vp.(y2|a2)|still_not-bound_in r2 =
                   v.(y2|a2)|still_not-bound_in r2 by SUBLEM_2:64;
                  then J,vp.(y2|a2) |= r2 by A38,SUBLEM_2:68;
                  then J,vp |= r2.(y2,fcr) by A39,CALCL1_2:24;
                  hence thesis by A32,Th8;
                 end;
              end;
            end;
            suppose not q2 in PHI2 & not q2 in B;
              then q2 = b by A28,TARSKI:def 1,XBOOLE_0:def 3;
              hence thesis by A17,A27,Th8;
            end;
          end;
          then J,vp |= PHI2 \/ (B \/ {b}) by CALCL1_2:def 11;
          hence thesis;
        end;
      end;
      P[Ex2] from FINSET_1:sch 2(A12,A13,A14);
      hence thesis;
    end;
    hence contradiction by A2,A11,HENMOD_2:12;
  end;
  hence thesis;
```

```
end;

:: Ebbinghaus Lemma 5.3.1
theorem Th10:
  ex Al2 being Al-extending FO-alphabet,
  PSI being Consistent Subset of CFO-WFF(Al2)
  st PHI c= PSI & PSI is with_examples
proof
  deffunc S(Element of NAT) = $1-th_FCEx(Al);
  deffunc PSI(Element of NAT) = $1-th_EF(Al,PHI);
  set Al2 = union {S(n) : not contradiction};
  set PSI = union {PSI(n) : not contradiction};
A1: PHI c= PSI
  proof
    PHI = PSI(0) by Def9;
    then PHI in {PSI(n) : not contradiction};
    hence PHI c= PSI by ZFMISC_1:74;
  end;
A2: Al c= Al2 & for n holds S(n) c= Al2
  proof
    Al = S(0) by Def7;
    then Al in {S(n) : not contradiction};
    hence Al c= Al2 by ZFMISC_1:74;
    let n;
    S(n) in {S(k) : not contradiction};
    hence S(n) c= Al2 by ZFMISC_1:74;
  end;
  reconsider Al2 as non empty set by A2;
  set Al2sym = union {FO-symbols(S(n)) : not contradiction};
    NAT c= Al2sym & Al2 = [:NAT,Al2sym:]
  proof
    for s being set st s in Al2 holds s in [:NAT,Al2sym:]
    proof
      let s be set such that
A4:   s in Al2;
      consider P being set such that
A5:   s in P & P in {S(n) : not contradiction} by A4,TARSKI:def 4;
      consider n being Element of NAT such that
A6:   P = S(n) by A5;
A7:   for y being set st y in FO-symbols(S(n)) holds y in Al2sym
      proof
        let y be set such that
A8:      y in FO-symbols(S(n));
         FO-symbols(S(n)) in {FO-symbols(S(k)) : not contradiction};
         hence y in Al2sym by A8,TARSKI:def 4;
      end;
      s in [:NAT,FO-symbols(S(n)):] by A6,A5,FO_LANG1:5;
      then ex k being set, y being set st k in NAT & y in FO-symbols(S(n)) &
        s = [k,y] by ZFMISC_1:def 2;
      then ex k being set,y being set st k in NAT &y in Al2sym & s=[k,y] by A7;
      hence thesis by ZFMISC_1:87;
    end;
    then
A9: Al2 c= [:NAT,Al2sym:] by TARSKI:def 3;
    FO-symbols(Al) = FO-symbols(S(0)) by Def7;
    then FO-symbols(Al) in {FO-symbols(S(n)) : not contradiction};
    then NAT c= FO-symbols(Al) & FO-symbols(Al) c= Al2sym
     by FO_LANG1:3,ZFMISC_1:74;
    hence NAT c= Al2sym by XBOOLE_1:1;
    for x being set st x in [:NAT,Al2sym:] holds x in Al2
    proof
      let x be set such that
A10:   x in [:NAT,Al2sym:];
      consider m,y being set such that
A11:   m in NAT & y in Al2sym & x = [m,y] by A10,ZFMISC_1:def 2;
      consider P being set such that
A12:  y in P & P in {FO-symbols(S(n)) : not contradiction} by A11,TARSKI:def 4;
      consider n being Element of NAT such that
A13:  P = FO-symbols(S(n)) by A12;
      [m,y] in [:NAT,FO-symbols(S(n)):] by A11,A12,A13,ZFMISC_1:87;
      then
A14:  [m,y] in S(n) by FO_LANG1:5;
      S(n) c= Al2 by A2;
      hence thesis by A11,A14;
    end;
    then [:NAT, Al2sym:] c= Al2 by TARSKI:def 3;
    hence Al2 = [:NAT, Al2sym:] by A9,XBOOLE_0:def 10;
  end;
  then reconsider Al2 as FO-alphabet by FO_LANG1:def 1;
  reconsider Al2 as Al-extending FO-alphabet by A2,FO_TRANS:def 1;
  for p being set st p in PSI holds p in CFO-WFF(Al2)
  proof
    let p be set such that
A15: p in PSI;
```

```
    consider P being set such that
A16: p in P & P in {PSI(n) : not contradiction} by A15,TARSKI:def 4;
    consider n being Element of NAT such that
A17: P = PSI(n) by A16;
    S(n) c= Al2 by A2;
    then Al2 is S(n)-extending FO-alphabet by FO_TRANS:def 1;
    then p is Element of CFO-WFF(Al2) by FO_TRANS:7,A16,A17;
    hence thesis;
  end;
  then reconsider PSI as Subset of CFO-WFF(Al2) by TARSKI:def 3;
  PSI is Consistent
  proof
    defpred C[Element of NAT] means PSI($1) is Consistent &
     PSI($1) is Al2-Consistent;
A19: C[0]
    proof
A20:  PSI(0) = PHI by Def9;
      PHI is Al2-Consistent by FO_TRANS:25;
      then S(0) = Al & for S being Subset of CFO-WFF(Al2) st PSI(0) = S holds
       S is Consistent by A20,Def7,FO_TRANS:def 2;
      hence PSI(0) is Consistent & PSI(0) is Al2-Consistent by FO_TRANS:def 2;
    end;
A21: for n holds C[n] implies C[n+1]
    proof
      let n;
A22:  FCEx(S(n)) = S(n+1) by Th5;
      S(n+1) c= Al2 by A2;
      then reconsider Al2 as S(n+1)-extending FO-alphabet by FO_TRANS:def 1;
      assume C[n];
      then reconsider PSIn = PSI(n) as Consistent Subset of CFO-WFF(S(n));
      PSI(n+1) = PSIn \/ Example_Formulae_of(S(n)) by Def9;
      then reconsider PSIn1 = PSI(n+1) as Consistent Subset of CFO-WFF(S(n+1))
       by A22,Th9;
      PSIn1 is Al2-Consistent by FO_TRANS:25;
      hence thesis;
    end;
A23: for n holds C[n] from NAT_1:sch 1(A19,A21);
A24: for n holds PSI(n) c= PSI
    proof
      let n;
      for p being set st p in PSI(n) holds p in PSI
      proof
        let p be set such that
A25:     p in PSI(n);
        PSI(n) in {PSI(k):not contradiction};
        hence p in PSI by A25,TARSKI:def 4;
      end;
      hence thesis by TARSKI:def 3;
    end;
A26: for n holds PSI(n) in bool CFO-WFF(Al2)
    proof
      let n;
      PSI(n) c= PSI & PSI c= CFO-WFF(Al2) by A24;
      then PSI(n) c= CFO-WFF(Al2) by XBOOLE_1:1;
      hence thesis;
    end;
    consider f being Function such that
A27: dom f = NAT & for n holds f.n = PSI(n) from FUNCT_1:sch 4;
    for y being set st y in rng f holds y in bool CFO-WFF(Al2)
    proof
      let y be set such that
A28:  y in rng f;
      consider x being set such that
A29:   x in dom f & y = f.x by A28,FUNCT_1:def 3;
      reconsider x as Element of NAT by A27,A29;
      f.x = PSI(x) by A27;
      hence thesis by A26,A29;
    end;
    then rng f c= bool CFO-WFF(Al2) by TARSKI:def 3;
    then reconsider f as Function of NAT,bool CFO-WFF(Al2) by A27,FUNCT_2:2;
    set PSIp = union rng f;
    f in Funcs(NAT,bool CFO-WFF(Al2)) by FUNCT_2:8;
    then union rng f c= union (bool CFO-WFF(Al2)) by ZFMISC_1:77,FUNCT_2:92;
    then reconsider PSIp as Subset of CFO-WFF(Al2) by ZFMISC_1:81;
    for n,m st m in dom f & n in dom f & n < m holds f.n is Consistent &
      f.n c= f.m
    proof
      let n,m such that
A30:   m in dom f & n in dom f & n < m;
      f.n is Subset of CFO-WFF(Al2) & f.n = PSI(n) & PSI(n) is Al2-Consistent
       by A23,A27;
      hence f.n is Consistent by FO_TRANS:def 2;
      defpred S[Element of NAT] means
       $1 <= m implies ex k st k=m-$1 & PSI(k) c= PSI(m);
```

```
A31:  S[0];
A32:  for k holds S[k] implies S[k+1]
      proof
        let k;
        assume
A33:     S[k];
        set j1 = m-k;
        set j2 = m-(k+1);
        per cases;
        suppose
A34:      k+1 <= m;
          then k <= m by NAT_1:13;
          then reconsider j1,j2 as Element of NAT by A34,NAT_1:21;
          PSI(j2+1) = (the EF-Sequence of Al,PHI).(j2)
                     \/ Example_Formulae_of(j2-th_FCEx(Al)) by Def9;
          then PSI(j2) c= PSI(j1) & PSI(j1) c= PSI(m)
           by A33,A34,NAT_1:13,XBOOLE_1:7;
          hence thesis by XBOOLE_1:1;
        end;
        suppose not k+1<=m;
          hence thesis;
        end;
      end;
A37:  for k holds S[k] from NAT_1:sch 1(A31,A32);
      set k = m-n;
      reconsider k as Element of NAT by A30,NAT_1:21;
      S[k] & k <= k+n by A37,NAT_1:11;
      then PSI(n) c= PSI(m) & f.n = PSI(n) & f.m = PSI(m) by A27;
      hence f.n c= f.m;
    end;
    then reconsider PSIp as Consistent Subset of CFO-WFF(Al2) by HENMOD_2:11;
    for y being set st y in {PSI(n): not contradiction} holds ex x being set
     st (x in dom f & y = f.x)
    proof
      let P be set such that
A38:   P in {PSI(n) : not contradiction};
      consider n such that
A39:   P = PSI(n) by A38;
      n in dom f & f.n = P by A27,A39;
      hence thesis;
    end;
    then
A40: {PSI(n) : not contradiction} c= rng f by FUNCT_1:9;
    for y being set st y in rng f holds y in {PSI(n) : not contradiction}
    proof
      let y be set such that
A41:  y in rng f;
      consider x being set such that
A42:   x in dom f & y = f.x by A41,FUNCT_1:def 3;
      reconsider x as Element of NAT by A27,A42;
      f.x = PSI(x) by A27;
      hence thesis by A42;
    end;
    then rng f c= {PSI(n) : not contradiction} by TARSKI:def 3;
    then PSIp = PSI by A40,XBOOLE_0:def 10;
    hence thesis;
  end;
  then reconsider PSI as Consistent Subset of CFO-WFF(Al2);
  set S = {S(n) : not contradiction};
  S(0) in S;
  then reconsider S as non empty set;
A46: for a,b being set st a in S & b in S holds ex c being set st
   a \/ b c= c & c in S
  proof
    let a,b be set such that
A43: a in S & b in S;
    consider i such that
A44: a = S(i) by A43;
    consider j such that
A45: b = S(j) by A43;
    per cases;
    suppose j <= i;
      then S(j) c= S(i) by Th6;
      hence thesis by A43,A44,A45,XBOOLE_1:8;
    end;
    suppose not j <= i;
      then S(i) c= S(j) by Th6;
      hence thesis by A43,A44,A45,XBOOLE_1:8;
    end;
  end;
A47: for p being Element of CFO-WFF(Al2) holds ex n st
   p is Element of CFO-WFF(S(n))
  proof
    defpred P[Element of CFO-WFF(Al2)] means ex n st $1 is Element of
```

```
     CFO-WFF(S(n));
A48: P[VERUM(Al2)]
    proof
      S(0) c= Al2 by A2;
      then reconsider Al2 as (S(0))-extending FO-alphabet by FO_TRANS:def 1;
      VERUM(S(0)) in CFO-WFF(S(0));
      then Al2-Cast(VERUM(S(0))) in CFO-WFF(S(0)) by FO_TRANS:def 3;
      then VERUM(Al2) in CFO-WFF(S(0)) by FO_TRANS:8;
      hence thesis;
    end;
A49: for k for P being FO-pred_symbol of k,Al2, l being CFO-variable_list
    of k,Al2 holds P[P!l]
    proof
      let k;
      let P be FO-pred_symbol of k,Al2, l be CFO-variable_list of k,Al2;
      ex n st rng l c= bound_FO-variables(S(n)) & P is FO-pred_symbol of k,S(n)
      proof
A50:    rng l c= bound_FO-variables(Al2) & {P} c= FO-pred_symbols(Al2)
         by ZFMISC_1:31;
        bound_FO-variables(Al2) c= FO-variables(Al2)
         & FO-variables(Al2) c= [:NAT,FO-symbols(Al2):] by FO_LANG1:4;
        then bound_FO-variables(Al2) c= [:NAT,FO-symbols(Al2):] &
         FO-pred_symbols(Al2) c= [:NAT,FO-symbols(Al2):]
         by FO_LANG1:6,XBOOLE_1:1;
        then
A51:     rng l c= [:NAT,FO-symbols(Al2):] & {P} c= [:NAT,FO-symbols(Al2):]
         by A50,XBOOLE_1:1;
        then rng l c= Al2 & {P} c= Al2 by FO_LANG1:5;
        then rng l \/ {P} c= union S & rng l \/ {P} is finite by XBOOLE_1:8;
        then consider a being set such that
A52:     a in S & rng l \/ {P} c= a by A46,COHSP_1:6,13;
        consider n such that
A53:     a = S(n) by A52;
        take n;
        rng l c= rng l \/ {P} & {P} c= rng l \/ {P} by XBOOLE_1:7;
        then
A54:    rng l c= S(n) & {P} c= S(n) by A52,A53,XBOOLE_1:1;
        for s being set st s in rng l holds s in bound_FO-variables(S(n))
        proof
          let s be set such that
A55:       s in rng l;
          s in bound_FO-variables(Al2) by A55;
          then s in [:{4}, FO-symbols(Al2):] by FO_LANG1:def 4;
          then consider s1,s2 being set such that
A56:       s1 in {4} & s2 in FO-symbols(Al2) & s = [s1,s2] by ZFMISC_1:def 2;
          s in S(n) by A54,A55;
          then s in [:NAT,FO-symbols(S(n)):] by FO_LANG1:5;
          then consider s3,s4 being set such that
A57:        s3 in NAT & s4 in FO-symbols(S(n)) & s = [s3,s4] by ZFMISC_1:def 2;
          s = [s1,s4] by A56,A57,ZFMISC_1:27;
          then s in [:{4},FO-symbols(S(n)):] by A56,A57,ZFMISC_1:def 2;
          hence thesis by FO_LANG1:def 4;
        end;
        hence rng l c= bound_FO-variables(S(n)) by TARSKI:def 3;
        thus P is FO-pred_symbol of k,S(n)
        proof
          P in [:NAT,FO-symbols(Al2):] by A51,ZFMISC_1:31;
          then consider p1,p2 being set such that
A58:       p1 in NAT & p2 in FO-symbols(Al2) & P = [p1,p2] by ZFMISC_1:def 2;
          rng l c= S(n) & P in S(n) by A54,ZFMISC_1:31;
          then P in [:NAT,FO-symbols(S(n)):] by FO_LANG1:5;
          then reconsider p2 as FO-symbol of S(n) by A58,ZFMISC_1:87;
A59:      P`1 =(the_arity_of P)+7 by FO_LANG1:def 8 .= k + 7 by FO_LANG1:11;
          reconsider p1 as Element of NAT by A58;
          P`1=7 + the_arity_of P & P`1=p1 by A58,FO_LANG1:def 8,MCART_1:def 1;
          then 7 <= p1 by NAT_1:11;
          then [p1,p2] in {[m,x] where x is FO-symbol of S(n): 7 <= m};
          then reconsider P as FO-pred_symbol of S(n) by A58,FO_LANG1:def 7;
          the_arity_of P = k by A59,FO_LANG1:def 8;
          then P in {Q where Q is FO-pred_symbol of S(n): the_arity_of Q=k};
          hence thesis by FO_LANG1:def 9;
        end;
      end;
      then consider n such that
A60:   rng l c= bound_FO-variables(S(n)) & P is FO-pred_symbol of k,S(n);
      rng l c= FO-variables(S(n)) by A60,XBOOLE_1:1;
      then l is CFO-variable_list of k,S(n) by A60,FINSEQ_1:def 4;
      then consider l2 being CFO-variable_list of k,S(n), P2 being
       FO-pred_symbol of k,S(n) such that
A61:   l2 = l & P = P2 by A60;
      S(n) c= Al2 by A2;
      then reconsider Al2 as (S(n))-extending FO-alphabet by FO_TRANS:def 1;
A62:  Al2-Cast(P2) = P & Al2-Cast(l2) = l by A61,FO_TRANS:def 5,def 6;
      P2!l2 = Al2-Cast(P2!l2) by FO_TRANS:def 3 .= P!l by A62,FO_TRANS:8;
      hence thesis;
    end;
A63: for r being Element of CFO-WFF(Al2) st P[r] holds P['not' r]
    proof
      let r be Element of CFO-WFF(Al2);
      assume P[r];
      then consider n such that
A64:   r is Element of CFO-WFF(S(n));
      consider r2 being Element of CFO-WFF(S(n)) such that
A65:   r = r2 by A64;
      S(n) c= Al2 by A2;
      then reconsider Al2 as (S(n))-extending FO-alphabet by FO_TRANS:def 1;
      'not' r2 = Al2-Cast('not' r2) by FO_TRANS:def 3
       .= 'not' Al2-Cast(r2) by FO_TRANS:8 .= 'not' r by A65,FO_TRANS:def 3;
      hence thesis;
    end;
A66: for r,s being Element of CFO-WFF(Al2) st P[r] & P[s] holds P[r '&' s]
    proof
      let r,s be Element of CFO-WFF(Al2);
      assume P[r] & P[s];
      then consider n,m such that
A67:   r is Element of CFO-WFF(S(n)) & s is Element of CFO-WFF(S(m));
      per cases;
      suppose n <= m;
        then S(n) c= S(m) by Th6;
        then reconsider Sm=S(m) as S(n)-extending FO-alphabet
         by FO_TRANS:def 1;
        r is Element of CFO-WFF(Sm) by A67,FO_TRANS:7;
        then consider r2,s2 being Element of CFO-WFF(Sm) such that
A68:     r2 = r & s2 = s by A67;
        S(m) c= Al2 by A2;
        then reconsider Al2 as Sm-extending FO-alphabet by FO_TRANS:def 1;
A69:    r = Al2-Cast(r2) & s = Al2-Cast(s2) by A68,FO_TRANS:def 3;
        r2 '&' s2 = Al2-Cast(r2 '&' s2) by FO_TRANS:def 3
         .= r '&' s by A69,FO_TRANS:8;
        hence thesis;
      end;
      suppose not n <= m;
        then S(m) c= S(n) by Th6;
        then reconsider Sn=S(n) as S(m)-extending FO-alphabet
         by FO_TRANS:def 1;
        s is Element of CFO-WFF(Sn) by A67,FO_TRANS:7;
        then consider r2,s2 being Element of CFO-WFF(Sn) such that
A70:     r2 = r & s2 = s by A67;
        S(n) c= Al2 by A2;
        then reconsider Al2 as Sn-extending FO-alphabet by FO_TRANS:def 1;
A71:    r = Al2-Cast(r2) & s = Al2-Cast(s2) by A70,FO_TRANS:def 3;
        r2 '&' s2 = Al2-Cast(r2 '&' s2) by FO_TRANS:def 3
         .= r '&' s by A71,FO_TRANS:8;
        hence thesis;
      end;
    end;
    for x being bound_FO-variable of Al2, r being Element of CFO-WFF(Al2) st
     P[r] holds P[All(x,r)]
    proof
      let x be bound_FO-variable of Al2, r be Element of CFO-WFF(Al2);
      x in FO-variables(Al2) & FO-variables(Al2) c= [:NAT,FO-symbols(Al2):]
       by FO_LANG1:4;
      then x in [:NAT,FO-symbols(Al2):] & Al2 = [:NAT,FO-symbols(Al2):]
       by FO_LANG1:5;
      then {x} c= union S & {x} is finite by ZFMISC_1:31;
      then consider a being set such that
A72:   a in S & {x} c= a by A46,COHSP_1:6,13;
      consider n such that
A73:   a = S(n) by A72;
      assume P[r];
      then consider m such that
A74:   r is Element of CFO-WFF(S(m));
      x in bound_FO-variables(Al2);
      then x in [:{4},FO-symbols(Al2):] by FO_LANG1:def 4;
      then consider x1,x2 being set such that
A75:   x1 in {4} & x2 in FO-symbols(Al2) & x = [x1,x2] by ZFMISC_1:def 2;
A76:  x in S(n) by A72,A73,ZFMISC_1:31;
      per cases;
      suppose n <= m;
        then
A77:     S(n) c= S(m) by Th6;
        then reconsider Sm=S(m) as S(n)-extending FO-alphabet
         by FO_TRANS:def 1;
        x in S(m) by A76,A77;
        then x in [:NAT, FO-symbols(S(m)):] by FO_LANG1:5;
        then consider x3,x4 being set such that
A78:     x3 in NAT & x4 in FO-symbols(S(m)) & x = [x3,x4] by ZFMISC_1:def 2;
        x = [x1,x4] by A75,A78,ZFMISC_1:27;
        then x in [:{4},FO-symbols(Sm):] by A75,A78,ZFMISC_1:def 2;
        then x is bound_FO-variable of Sm by FO_LANG1:def 4;
        then consider x2 being bound_FO-variable of Sm, r2 being Element of
         CFO-WFF(Sm) such that
A79:     x2 = x & r2 = r by A74;
        S(m) c= Al2 by A2;
        then reconsider Al2 as Sm-extending FO-alphabet by FO_TRANS:def 1;
A80:    r = Al2-Cast(r2) & x = Al2-Cast(x2) by A79,FO_TRANS:def 3,def 4;
        All(x2,r2) = Al2-Cast(All(x2,r2)) by FO_TRANS:def 3
         .= All(x,r) by A80,FO_TRANS:8;
        hence thesis;
      end;
      suppose not n <= m;
        then S(m) c= S(n) by Th6;
        then reconsider Sn=S(n) as S(m)-extending FO-alphabet
         by FO_TRANS:def 1;
        x in [:NAT,FO-symbols(Sn):] by A76,FO_LANG1:5;
        then consider x3,x4 being set such that
A81:     x3 in NAT & x4 in FO-symbols(Sn) & x = [x3,x4] by ZFMISC_1:def 2;
        x = [x1,x4] by A75,A81,ZFMISC_1:27;
        then x in [:{4},FO-symbols(Sn):] by A75,A81,ZFMISC_1:def 2;
        then x is bound_FO-variable of Sn & r is Element of CFO-WFF(Sn)
         by A74,FO_TRANS:7,FO_LANG1:def 4;
        then consider x2 being bound_FO-variable of Sn, r2 being Element of
         CFO-WFF(Sn) such that
A82:     x2 = x & r2 = r;
        S(n) c= Al2 by A2;
        then reconsider Al2 as Sn-extending FO-alphabet by FO_TRANS:def 1;
A83:    r = Al2-Cast(r2) & x = Al2-Cast(x2) by A82,FO_TRANS:def 3,def 4;
        All(x2,r2) = Al2-Cast(All(x2,r2)) by FO_TRANS:def 3
         .= All(x,r) by A83,FO_TRANS:8;
        hence thesis;
      end;
    end;
    then
A84: for r,s being Element of CFO-WFF(Al2), x being bound_FO-variable of
     Al2, k being Element of NAT, l being CFO-variable_list of k,Al2,
     P being FO-pred_symbol of k,Al2 holds P[VERUM(Al2)] & P[P!l] & (P[r]
     implies P['not' r]) & (P[r] & P[s] implies P[r '&' s]) &
     (P[r] implies P[All(x,r)]) by A48,A49,A63,A66;
    for p being Element of CFO-WFF(Al2) holds P[p] from CFO_LANG:sch 1(A84);
    hence thesis;
  end;
  PSI is with_examples
  proof
    for x being bound_FO-variable of Al2, p being Element of CFO-WFF(Al2) holds
     ex y be bound_FO-variable of Al2 st PSI |- ('not' Ex(x,p) 'or' (p.(x,y)))
    proof
      let x be bound_FO-variable of Al2, p be Element of CFO-WFF(Al2);
      ex n st (x is bound_FO-variable of S(n) & p is Element of CFO-WFF(S(n)))
      proof
        consider m such that
A85:     p is Element of CFO-WFF(S(m)) by A47;
        x in FO-variables(Al2) & FO-variables(Al2) c= [:NAT,FO-symbols(Al2):]
         by FO_LANG1:4;
        then x in [:NAT,FO-symbols(Al2):] & Al2 = [:NAT,FO-symbols(Al2):]
         by FO_LANG1:5;
        then {x} c= union S & {x} is finite by ZFMISC_1:31;
        then consider a being set such that
A86:     a in S & {x} c= a by A46,COHSP_1:6,13;
        consider n such that
A87:     a = S(n) by A86;
        x in bound_FO-variables(Al2);
        then x in [:{4},FO-symbols(Al2):] by FO_LANG1:def 4;
        then consider x1,x2 being set such that
A88:     x1 in {4} & x2 in FO-symbols(Al2) & x = [x1,x2] by ZFMISC_1:def 2;
A89:    x in S(n) by A86,A87,ZFMISC_1:31;
        per cases;
        suppose n <= m;
          then
A90:       S(n) c= S(m) by Th6;
          then reconsider Sm = S(m) as S(n)-extending FO-alphabet by
           FO_TRANS:def 1;
          x in S(m) by A89,A90;
          then x in [:NAT, FO-symbols(S(m)):] by FO_LANG1:5;
          then consider x3,x4 being set such that
A91:       x3 in NAT & x4 in FO-symbols(S(m)) & x = [x3,x4] by ZFMISC_1:def 2;
          x = [x1,x4] by A88,A91,ZFMISC_1:27;
          then x in [:{4},FO-symbols(Sm):] by A88,A91,ZFMISC_1:def 2;
          then x is bound_FO-variable of Sm by FO_LANG1:def 4;
          hence thesis by A85;
        end;
        suppose not n <= m;
          then S(m) c= S(n) by Th6;
          then reconsider Sn = S(n) as S(m)-extending FO-alphabet by
           FO_TRANS:def 1;
          x in [:NAT, FO-symbols(S(n)):] by A89,FO_LANG1:5;
          then consider x3,x4 being set such that
A92:       x3 in NAT & x4 in FO-symbols(S(n)) & x = [x3,x4] by ZFMISC_1:def 2;
          x = [x1,x4] by A88,A92,ZFMISC_1:27;
          then x in [:{4},FO-symbols(Sn):] by A88,A92,ZFMISC_1:def 2;
          then x is bound_FO-variable of Sn & p is Element of CFO-WFF(Sn)
           by A85,FO_TRANS:7,FO_LANG1:def 4;
          hence thesis;
        end;
      end;
      then consider n such that
A93:   x is bound_FO-variable of S(n) & p is Element of CFO-WFF(S(n));
A94:  FCEx(S(n)) = S(n+1) by Th5;
A95:  PSI(n+1) = PSI(n) \/ Example_Formulae_of(S(n)) by Def9;
      consider x2 being bound_FO-variable of S(n), p2 being Element of
       CFO-WFF(S(n)) such that
A96:   x2 = x & p2 = p by A93;
      Example_Formula_of(p2,x2) in Example_Formulae_of(S(n));
      then
A97:  Example_Formula_of(p2,x2) in PSI(n+1) by A95,XBOOLE_0:def 3;
      S(n) c= S(n+1) by Th6,NAT_1:11;
      then reconsider Sn1 = S(n+1) as S(n)-extending FO-alphabet
       by FO_TRANS:def 1;
      set y2 = Example_of(p2,x2);
      reconsider y2 as bound_FO-variable of Sn1 by Th5;
      S(n+1) c= Al2 by A2;
      then reconsider Al2 as Sn1-extending FO-alphabet by FO_TRANS:def 1;
      bound_FO-variables(Sn1) c= bound_FO-variables(Al2) by FO_TRANS:4;
      then y2 is bound_FO-variable of Al2 by TARSKI:def 3;
      then consider y being bound_FO-variable of Al2 such that
A98:   y = y2;
A99:  Sn1-Cast(p2) = p & Sn1-Cast(x2) = x by A96,FO_TRANS:def 3,def 4;
      then
A100:  Al2-Cast(Sn1-Cast(p2)) = p & Al2-Cast(Sn1-Cast(x2)) = x
       by FO_TRANS:def 3,def 4;
A101: Al2-Cast(Ex(Sn1-Cast(x2),Sn1-Cast(p2))) = Ex(x,p) by A100,Th7;
      reconsider p as Element of CFO-WFF(Al2);
      reconsider x as bound_FO-variable of Al2;
A102: (Sn1-Cast(p2)).(Sn1-Cast(x2),y2) = p.(x,y) by A98,A99,FO_TRANS:18;
A103: Example_Formula_of(p2,x2)
       = Al2-Cast(('not' Ex(Sn1-Cast(x2), Sn1-Cast(p2))) 'or'
         ((Sn1-Cast(p2)).(Sn1-Cast(x2),y2))) by A94,FO_TRANS:def 3
       .= Al2-Cast('not' Ex(Sn1-Cast(x2), Sn1-Cast(p2))) 'or'
        Al2-Cast((Sn1-Cast(p2)).(Sn1-Cast(x2),y2)) by Th7
       .= 'not' Al2-Cast(Ex(Sn1-Cast(x2), Sn1-Cast(p2))) 'or'
        Al2-Cast((Sn1-Cast(p2)).(Sn1-Cast(x2),y2)) by FO_TRANS:8
       .= 'not' Ex(x,p) 'or' p.(x,y) by A101,A102,FO_TRANS:def 3;
      set example = 'not' Ex(x,p) 'or' p.(x,y);
      reconsider example as Element of CFO-WFF(Al2);
      reconsider PSI as Consistent Subset of CFO-WFF(Al2);
      PSI(n+1) in {PSI(m) : not contradiction};
      then PSI(n+1) c= PSI by ZFMISC_1:74;
      hence thesis by A97,A103,GOEDCP_2:21;
    end;
    hence thesis by GOEDCP_2:def 2;
  end;
  hence thesis by A1;
end;

theorem Th11:
 PHI \/ {p} is Consistent or PHI \/ {'not' p} is Consistent
proof
  assume not PHI \/ {p} is Consistent & not PHI \/ {'not' p} is Consistent;
  then PHI |- 'not' p & PHI |- p by HENMOD_2:9,10;
  hence contradiction by HENMOD_2:def 2;
end;

:: Ebbinghaus Lemma 5.3.2
theorem Th12:
  for PSI being Consistent Subset of CFO-WFF(Al) holds
  ex THETA being Consistent Subset of CFO-WFF(Al)
  st THETA is negation_faithful & PSI c= THETA
proof
  let PSI be Consistent Subset of CFO-WFF(Al);
  set U = { PHI : PSI c= PHI };
A1: PSI in U;
  ( for Z being set st Z c= U & Z is c=-linear holds ex Y being set st
   ( Y in U & ( for X being set st X in Z holds X c= Y ) ) )
  proof
    let Z be set such that
A2: Z c= U & Z is c=-linear;
    per cases;
    suppose
A3:   Z is empty;
      PSI in U & for X being set st X in Z holds X c= PSI by A3;
      hence thesis;
    end;
    suppose
A4:   Z is non empty;
      set Y = union Z;
      for z being set st z in PSI holds z in Y
      proof
        let z be set such that
A5:     z in PSI;
        consider X being set such that
A6:     X in Z by A4, XBOOLE_0:7;
        X in U by A2,A6;
        then ex R being Consistent Subset of CFO-WFF(Al) st X = R & PSI c= R;
        hence z in Y by A5,A6,TARSKI:def 4;
      end;
      then
A7:   PSI c= Y by TARSKI:def 3;
A8:   Y is Consistent Subset of CFO-WFF(Al)
      proof
        for X being set st X in Z holds X c= CFO-WFF(Al)
        proof
          let X be set such that
A9:       X in Z;
          X in U by A2,A9;
          then ex R being Consistent Subset of CFO-WFF(Al) st X = R & PSI c= R;
          hence X c= CFO-WFF(Al);
        end;
        then reconsider Y as Subset of CFO-WFF(Al) by ZFMISC_1:76;
        Y is Consistent
        proof
          assume Y is Inconsistent;
          then consider X being Subset of CFO-WFF(Al) such that
A10:       X c= Y & X is finite & X is Inconsistent by HENMOD_2:7;
          ex Rs being finite Subset of Z st for x being set st x in X holds
          ex R being set st R in Rs & x in R
          proof
            defpred R[set] means ex Rs being finite Subset of Z
              st for x being set st x in $1 holds
              ex R being set st R in Rs & x in R;
A13:        R[ {} ]
            proof
              consider Rs being set such that
A14:          Rs in Z by A4, XBOOLE_0:7;
              set Rss = {Rs};
              reconsider Rss as finite Subset of Z by A14,ZFMISC_1:31;
              for x being set st x in {} ex R being set st R in Rss & x in R;
              hence thesis;
            end;
A15:         for x,B being set st x in X & B c= X & R[B] holds R[B \/ {x}]
            proof
              let x,B being set such that
A16:            x in X & B c= X & R[B];
              consider Rs being finite Subset of Z such that
A17:          for b being set st b in B holds ex R being set st
               R in Rs & b in R by A16;
              consider S being set such that
A18:           (x in S & S in Z) by A10,A16,TARSKI:def 4;
              set Rss = Rs \/ {S};
              Rs c= Z & {S} c= Z by A18, ZFMISC_1:31;
              then
A19:          Rss c= Z by XBOOLE_1:8;
              for y being set st y in B \/ {x} holds ex R being set
              st R in Rss & y in R
              proof
                let y be set such that
A20:              y in B \/ {x};
                per cases by A20,XBOOLE_0:def 3;
                suppose y in {x};
                  then
A22:              y = x by TARSKI:def 1;
                  S in {S} by TARSKI:def 1;
                  then S in Rss by XBOOLE_0:def 3;
                  hence thesis by A18,A22;
                end;
                suppose y in B;
                  then consider R being set such that
A23:               R in Rs & y in R by A17;
                  R in Rss by A23, XBOOLE_0:def 3;
                  hence thesis by A23;
                end;
              end;
              hence thesis by A19;
            end;
A24:        X is finite by A10;
            R[X] from FINSET_1:sch 2(A24,A13,A15);
            hence thesis;
          end;
          then consider Rs being finite Subset of Z such that
A25:     for x being set st x in X holds ex R being set st R in Rs & x in R;
A27:     for R,S being set st R in Rs & S in Rs holds R c= S or S c= R
          proof
            let R,S be set such that
A28:      R in Rs & S in Rs;
            R,S are_c=-comparable by A2,A28,ORDINAL1:def 8;
            hence thesis by XBOOLE_0:def 9;
          end;
          defpred F[set] means $1 is non empty implies union $1 in $1;
A30:     Rs is finite;
A31:     F[{}];
A32:     for x,B being set st x in Rs & B c= Rs & F[B] holds F[B \/ {x}]
          proof
            let x,B be set such that
A33:      x in Rs & B c= Rs & F[B];
            per cases;
            suppose
A34:        B is empty;
A35:        union (B \/ {x}) = x by A34,ZFMISC_1:25;
              thus thesis by A34,A35,TARSKI:def 1;
            end;
            suppose
A36:        B is non empty;
              per cases by A27,A33,A36;
              suppose
A38:          x c= union B;
                union (B \/ {x}) = union B \/ union {x} by ZFMISC_1:78
                     .= union B \/ x by ZFMISC_1:25
                     .= union B by A38,XBOOLE_1:12;
                hence thesis by A33,A36,XBOOLE_0:def 3;
              end;
              suppose
A39:          union B c= x;
A40:          x in {x} by TARSKI:def 1;
                union (B \/ {x}) = union B \/ union {x} by ZFMISC_1:78
                     .= union B \/ x by ZFMISC_1:25
                     .= x by A39, XBOOLE_1:12;
                hence thesis by A40,XBOOLE_0:def 3;
              end;
            end;
          end;
A41:    F[Rs] from FINSET_1:sch 2(A30,A31,A32);
          X is non empty
          proof
            assume
A42:      X is empty;
            X |- 'not' VERUM(Al) by A10,HENMOD_2:6;
            then X \/ {VERUM(Al)} is Inconsistent  &
            X \/ {VERUM(Al)} = {VERUM(Al)} by A42,HENMOD_2:10;
            hence contradiction by HENMOD_2:13;
          end;
          then consider x being set such that
A43:      x in X by XBOOLE_0:def 1;
          ex R being set st R in Rs & x in R by A25,A43;
          then union Rs in Z by A41;
          then union Rs in U by A2;
          then consider uRs being Consistent Subset of CFO-WFF(Al) such that
A44:     union Rs = uRs & PSI c= uRs;
          for x being set st x in X holds x in uRs
          proof
            let x be set such that
A45:       x in X;
           ex R being set st R in Rs & x in R by A25,A45;
           hence thesis by A44,TARSKI:def 4;
          end;
          then
A46:    X c= uRs by TARSKI:def 3;
          X |- 'not' VERUM(Al) by A10, GOEDCP_2:24;
          then consider f being FinSequence of CFO-WFF(Al) such that
A47:      rng f c= X & |- f^<*'not' VERUM(Al)*> by HENMOD_2:def 1;
          rng f c= uRs by A46,A47,XBOOLE_1:1;
          then uRs |- 'not' VERUM(Al) by A47, HENMOD_2:def 1;
          hence contradiction by GOEDCP_2:24;
        end;
        hence thesis;
      end;
      Y in U & for X being set st X in Z holds X c= Y by A7,A8,ZFMISC_1:74;
```

```
      hence thesis;
    end;
  end;
  then consider THETA being set such that
A48: (THETA in U & (for Z being set st Z in U & Z <> THETA holds
     not THETA c= Z )) by A1,ORDERS_1:65; ::Zorns Lemma
A49: ex PHI st PHI = THETA & PSI c= PHI by A48;
  then reconsider THETA as Consistent Subset of CFO-WFF(Al);
A50: for p holds (p in THETA or 'not' p in THETA)
  proof
    let p;
    per cases by Th11;
    suppose
A51:   THETA \/ {p} is Consistent;
      assume
A52:   not p in THETA;
      p in {p} by TARSKI:def 1;
      then
A54:  p in THETA \/ {p} & not p in THETA by XBOOLE_0:def 3, A52;
      PSI c= THETA \/ {p} by A49,XBOOLE_1:10;
      then THETA \/ {p} in U by A51;
      hence thesis by A48,A54,XBOOLE_1:10;
    end;
    suppose
A55:   THETA \/ {'not' p} is Consistent;
      'not' p in THETA
      proof
        assume
A56:    not ('not' p in THETA);
        'not' p in {'not' p} by TARSKI:def 1;
        then
A58:    'not' p in THETA \/ {'not' p} & not 'not' p in THETA
         by XBOOLE_0:def 3, A56;
        PSI c= THETA \/ {'not' p} by A49,XBOOLE_1:10;
        then THETA \/ {'not' p} in U by A55;
        hence thesis by A48,A58,XBOOLE_1:10;
      end;
      hence thesis;
    end;
  end;
  for p holds THETA |- p or THETA |- 'not' p
  proof
    let p;
    per cases by A50;
    suppose p in THETA;
      hence thesis by GOEDCP_2:21;
    end;
    suppose 'not' p in THETA;
      hence thesis by GOEDCP_2:21;
    end;
  end;
  then THETA is negation_faithful by GOEDCP_2:def 1;
  hence thesis by A49;
end;

theorem Th13:
  for THETA being Consistent Subset of CFO-WFF(Al)
  st PHI c= THETA & PHI is with_examples holds THETA is with_examples
proof
  let THETA be Consistent Subset of CFO-WFF(Al) such that
A1:  PHI c= THETA & PHI is with_examples;
  now
    let x be bound_FO-variable of Al, p be Element of CFO-WFF(Al);
    consider y being bound_FO-variable of Al such that
A4: PHI |- ('not' Ex(x,p) 'or' (p.(x,y))) by A1,GOEDCP_2:def 2;
    consider f being FinSequence of CFO-WFF(Al) such that
A5: rng f c= PHI & |- f^<*('not' Ex(x,p) 'or' (p.(x,y)))*>
    by A4, HENMOD_2:def 1;
A6: rng f c= THETA by XBOOLE_1:1, A1, A5;
    take y;
    thus THETA |- ('not' Ex(x,p) 'or' (p.(x,y))) by A5, A6, HENMOD_2:def 1;
  end;
  hence thesis by GOEDCP_2:def 2;
end;

:: Ebbinghaus Corollary 5.3.3
:: Model Existence Theorem
theorem Th14:
  PHI is satisfiable
proof
  consider Al2 being Al-extending FO-alphabet,
  PSI being Consistent Subset of CFO-WFF(Al2) such that
A1: PHI c= PSI & PSI is with_examples by Th10;
  consider THETA being Consistent Subset of CFO-WFF(Al2) such that
```

```
A2: THETA is negation_faithful & PSI c= THETA by Th12;
  set JH =the Henkin_interpretation of THETA;
  now
    let p be Element of CFO-WFF(Al2);
A3: THETA is with_examples by Th13, A2, A1;
    assume p in THETA;
    then THETA |- p by GOEDCP_2:21;
    hence JH,valH(Al2) |= p by GOEDCP_2:17,A2,A3;
  end;
  then
A4: JH,valH(Al2) |= THETA by CALCL1_2:def 11;
  ex A,J,v st J,v |= PHI by A1,A2,A4,FO_TRANS:10,XBOOLE_1:1;
  hence thesis by Def1;
end;

:: Completeness Theorem
theorem
  PSI |= p implies PSI |- p
proof
  assume A1: PSI |= p;
  assume A2: not PSI |- p;
  reconsider CHI = PSI \/ {'not' p} as Subset of CFO-WFF(Al);
  CHI is Consistent by A2,HENMOD_2:9;
  then CHI is satisfiable by Th14;
  then ex A,J,v st J,v |= CHI by Def1;
  hence contradiction by GOEDCP_2:37,A1;
end;
```

9 Appendix - Preliminaries

9.1 FO_LANG1

```
:: A First Order Language
::  by Piotr Rudnicki and Andrzej Trybulec
::
:: Received August 8, 1989
:: Copyright (c) 1990-2011 Association of Mizar Users
::           (Stowarzyszenie Uzytkownikow Mizara, Bialystok, Poland).
:: This code can be distributed under the GNU General Public Licence
:: version 3.0 or later, or the Creative Commons Attribution-ShareAlike
:: License version 3.0 or later, subject to the binding interpretation
:: detailed in file COPYING.interpretation.
:: See COPYING.GPL and COPYING.CC-BY-SA for the full text of these
:: licenses, or see http://www.gnu.org/licenses/gpl.html and
:: http://creativecommons.org/licenses/by-sa/3.0/.

environ

 vocabularies NUMBERS, XBOOLE_0, SUBSET_1, ZFMISC_1, TARSKI, XXREAL_0,
      MARGREL1, MCART_1, ARYTM_3, NAT_1, FINSEQ_1, RELAT_1, ORDINAL4, CARD_1,
      REALSET1, XBOOLEAN, BVFUNC_2, ZF_LANG, CLASSES2, FUNCT_1, FUNCOP_1,
      RCOMP_1, FO_LANG1;
 notations TARSKI, XBOOLE_0, ENUMSET1, ZFMISC_1, SUBSET_1, CARD_1, NUMBERS,
      XXREAL_0, MCART_1, NAT_1, RELAT_1, FUNCT_1, RELSET_1, FUNCT_2, FUNCOP_1,
      FINSEQ_1;
 constructors ENUMSET1, FUNCOP_1, XXREAL_0, XREAL_0, NAT_1, FINSEQ_1, RELSET_1;
 registrations XBOOLE_0, SUBSET_1, RELAT_1, FUNCT_1, ORDINAL1, XREAL_0,
      FINSEQ_1, CARD_1;
 requirements NUMERALS, REAL, SUBSET, BOOLE, ARITHM;
 definitions TARSKI, XBOOLE_0, FINSEQ_1, CARD_1;
 theorems ZFMISC_1, SUBSET_1, TARSKI, FINSEQ_1, MCART_1, NAT_1, FUNCT_1,
      FUNCT_2, RELSET_1, XBOOLE_0, XBOOLE_1, FUNCOP_1, XXREAL_0, ORDINAL1,
      CARD_1, RELAT_1;
 schemes NAT_1, FUNCT_2, CLASSES1, XBOOLE_0;

begin

:: Preliminaries

theorem :: FO_LANG1:1
  for D1 being non empty set, D2 being set, k being Element of D1 holds
  [: {k}, D2 :] c= [: D1, D2 :];

theorem :: FO_LANG1:2
  for D1 being non empty set, D2 being set, k1, k2, k3 being
  Element of D1 holds [: {k1, k2, k3}, D2 :] c= [: D1, D2 :];

definition
  mode FO-alphabet means
```

```
:: FO_LANG1:def 1
  it is non empty set & ex X being set st NAT c= X & it = [: NAT, X :];
end;

registration
cluster -> non empty Relation-like for FO-alphabet;
end;

reserve A for FO-alphabet;
reserve k,n,m for Element of NAT;

definition
  let A be FO-alphabet;
  func FO-symbols(A) -> non empty set equals
:: FO_LANG1:def 2
    rng A;
end;

definition
  let A be FO-alphabet;
  mode FO-symbol of A is
  Element of FO-symbols(A);
end;

theorem :: FO_LANG1:3
NAT c= FO-symbols(A) & 0 in FO-symbols(A);

registration
  let A be FO-alphabet;
  cluster FO-symbols(A) -> non empty;
end;

definition
  let A be FO-alphabet;
  func FO-variables(A) -> set equals
:: FO_LANG1:def 3
    [: {6}, NAT :] \/ [: {4,5}, FO-symbols(A) :];
end;

registration
  let A be FO-alphabet;
  cluster FO-variables(A) -> non empty;
end;

theorem :: FO_LANG1:4
  FO-variables(A) c= [: NAT, FO-symbols(A) :];


definition
  let A be FO-alphabet;
  mode FO-variable of A is Element of FO-variables(A);
  func bound_FO-variables(A) -> Subset of FO-variables(A) equals
:: FO_LANG1:def 4
  [: {4}, FO-symbols(A) :];
  func fixed_FO-variables(A) -> Subset of FO-variables(A) equals
:: FO_LANG1:def 5
  [: {5}, FO-symbols(A) :];
  func free_FO-variables(A) -> Subset of FO-variables(A) equals
:: FO_LANG1:def 6
  [: {6}, NAT :];
  func FO-pred_symbols(A) -> set equals
:: FO_LANG1:def 7
  { [n, x] where x is FO-symbol of A : 7 <= n };
end;

registration
  let A be FO-alphabet;
  cluster bound_FO-variables(A) -> non empty;
  cluster fixed_FO-variables(A) -> non empty;
  cluster free_FO-variables(A) -> non empty;
  cluster FO-pred_symbols(A) -> non empty;
end;

theorem :: FO_LANG1:5
  A = [: NAT, FO-symbols(A) :];

theorem :: FO_LANG1:6
  FO-pred_symbols(A) c= [: NAT, FO-symbols(A) :];

definition
  let A be FO-alphabet;
  mode FO-pred_symbol of A is
  Element of FO-pred_symbols(A);
end;
```

```
definition
  let A be FO-alphabet;
  let P be Element of FO-pred_symbols(A);
  func the_arity_of P -> Element of NAT means
:: FO_LANG1:def 8

  P‘1 = 7+it;
end;

reserve P for FO-pred_symbol of A;

definition
  let A;
  let k;
  func k-ary_FO-pred_symbols(A) -> Subset of FO-pred_symbols(A) equals
:: FO_LANG1:def 9
  { P :
  the_arity_of P = k };
end;

registration
  let k;
  let A;
  cluster k-ary_FO-pred_symbols(A) -> non empty;
end;

definition
  let A be FO-alphabet;
  mode bound_FO-variable of A is
  Element of bound_FO-variables(A);
  mode fixed_FO-variable of A is
  Element of fixed_FO-variables(A);
  mode free_FO-variable of A is
  Element of free_FO-variables(A);
  let k;
  mode FO-pred_symbol of k, A is
  Element of k-ary_FO-pred_symbols(A);
end;

registration
  let k be Element of NAT;
  let A be FO-alphabet;
  cluster k-element for FinSequence of FO-variables(A);
end;

definition
  let k be Element of NAT;
  let A be FO-alphabet;
  mode FO-variable_list of k, A is
  k-element FinSequence of FO-variables(A);
end;

definition
  let A be FO-alphabet;
  let D be set;
  attr D is A-closed means
:: FO_LANG1:def 10

   D is Subset of [:NAT, FO-symbols(A):]* &  :: Includes atomic formulae
  (for k being Element of NAT, p being (FO-pred_symbol of k,A),
  ll being FO-variable_list of k,A holds <*p*>^ll in D) &

:: Is closed under VERUM, 'not', '&', and quantification
  <*[0, 0]*> in D &
  (for p being FinSequence of [:NAT,FO-symbols(A):]
    st p in D holds <*[1, 0]*>^p in D) &
  (for p, q being FinSequence of [:NAT, FO-symbols(A):] st p in D &
    q in D holds <*[2, 0]*>^p^q in D) &
  (for x being bound_FO-variable of A,
    p being FinSequence of [:NAT, FO-symbols(A):]
    st p in D holds <*[3, 0]*>^<*x*>^p in D);
end;

definition
  let A be FO-alphabet;
  func FO-WFF(A) -> non empty set means
:: FO_LANG1:def 11

  it is A-closed & for D being non empty set st D is A-closed holds it c= D;
end;

theorem :: FO_LANG1:7
  FO-WFF(A) is A-closed;
```

```
registration
  let A be FO-alphabet;
  cluster A-closed non empty for set;
end;

definition
  let A be FO-alphabet;
  mode FO-formula of A is
  Element of FO-WFF(A);
end;

definition
  let A be FO-alphabet;
  let P be FO-pred_symbol of A;
  let l be FinSequence of FO-variables(A);
  assume
 the_arity_of P = len l;
  func P!l -> Element of FO-WFF(A) equals
:: FO_LANG1:def 12
  <*P*>^l;
end;

theorem :: FO_LANG1:8
  for k being Element of NAT, p being FO-pred_symbol of k, A, ll be
  FO-variable_list of k, A holds p!ll = <*p*>^ll;

definition
  let A be FO-alphabet;
  let p be Element of FO-WFF(A);
  func @p -> FinSequence of [:NAT, FO-symbols(A):] equals
:: FO_LANG1:def 13
  p;
end;

definition
  let A be FO-alphabet;
  func VERUM(A) -> FO-formula of A equals
:: FO_LANG1:def 14
  <*[0, 0]*>;
  let p be Element of FO-WFF(A);
  func 'not' p -> FO-formula of A equals
:: FO_LANG1:def 15
  <*[1, 0]*>^@p;
  let q be Element of FO-WFF(A);
  func p '&' q -> FO-formula of A equals
:: FO_LANG1:def 16
  <*[2, 0]*>^@p^@q;
end;

definition
  let A be FO-alphabet;
  let x be bound_FO-variable of A, p be Element of FO-WFF(A);
  func All(x, p) -> FO-formula of A equals
:: FO_LANG1:def 17
  <*[3, 0]*>^<*x*>^@p;
end;

reserve F for Element of FO-WFF(A);

scheme :: FO_LANG1:sch 1

  FOInd { A() -> FO-alphabet, Prop[Element of FO-WFF(A())] }:
          for F being Element of FO-WFF(A()) holds Prop[F]
provided
 for k being Element of NAT, P being (FO-pred_symbol of k, A()), ll being
FO-variable_list of k, A() holds Prop[P!ll] and
 Prop[VERUM(A())] and
 for p being Element of FO-WFF(A()) st Prop[p] holds Prop['not' p] and
 for p, q being Element of FO-WFF(A()) st Prop[p] & Prop[q] holds Prop[p
'&' q] and
 for x being bound_FO-variable of A(),
    p being Element of FO-WFF(A()) st Prop[p]
    holds Prop[All(x, p)];

definition
  let A be FO-alphabet;
  let F be Element of FO-WFF(A);
  attr F is atomic means
:: FO_LANG1:def 18

  ex k being Element of NAT, p being (
  FO-pred_symbol of k, A), ll being FO-variable_list of k, A st F = p!ll;
  attr F is negative means
```

```
:: FO_LANG1:def 19

  ex p being Element of FO-WFF(A) st F = 'not' p;
  attr F is conjunctive means
:: FO_LANG1:def 20

  ex p, q being Element of FO-WFF(A) st F = p '&' q;
  attr F is universal means
:: FO_LANG1:def 21

  ex x being bound_FO-variable of A,
     p being Element of FO-WFF(A) st F = All(x, p);
end;

theorem :: FO_LANG1:9
  for F being Element of FO-WFF(A) holds F = VERUM(A) or F is atomic or
  F is negative or F is conjunctive or F is universal;

theorem :: FO_LANG1:10
  for F being Element of FO-WFF(A) holds 1 <= len @F;

reserve Q for FO-pred_symbol of A;

theorem :: FO_LANG1:11
  for k being Element of NAT, P being FO-pred_symbol of k, A holds
  the_arity_of P = k;

reserve F, G for (Element of FO-WFF(A)), s for FinSequence;

theorem :: FO_LANG1:12
  ((@F.1)`1 = 0 implies F = VERUM(A)) & ((@F.1)`1 = 1 implies F is
  negative) & ((@F.1)`1 = 2 implies F is conjunctive) & ((@F.1)`1 = 3 implies F
  is universal) & ((ex k being Element of NAT
                     st @F.1 is FO-pred_symbol of k, A)
  implies F is atomic);

theorem :: FO_LANG1:13
  @F = @G^s implies @F = @G;

definition
  let A be FO-alphabet;
  let F be Element of FO-WFF(A) such that
 F is atomic;
  func the_pred_symbol_of F -> FO-pred_symbol of A means
:: FO_LANG1:def 22

  ex k being Element
of NAT, ll being (FO-variable_list of k, A),
       P being FO-pred_symbol of k, A st it = P
  & F = P!ll;
end;

definition
  let A be FO-alphabet;
  let F be Element of FO-WFF(A) such that
 F is atomic;
  func the_arguments_of F -> FinSequence of FO-variables(A) means
:: FO_LANG1:def 23

  ex k
being Element of NAT, P being (FO-pred_symbol of k, A),
                      ll being FO-variable_list
  of k, A st it = ll & F = P!ll;
end;

definition
  let A be FO-alphabet;
  let F be Element of FO-WFF(A) such that
 F is negative;
  func the_argument_of F -> FO-formula of A means
:: FO_LANG1:def 24

  F = 'not' it;
end;

definition
  let A be FO-alphabet;
  let F be Element of FO-WFF(A) such that
 F is conjunctive;
  func the_left_argument_of F -> FO-formula of A means
:: FO_LANG1:def 25

  ex q being Element of FO-WFF(A) st F = it '&' q;
end;
```

```
definition
  let A be FO-alphabet;
  let F be Element of FO-WFF(A) such that
 F is conjunctive;
  func the_right_argument_of F -> FO-formula of A means
:: FO_LANG1:def 26

  ex p being Element of FO-WFF(A) st F = p '&' it;
end;

definition
  let A be FO-alphabet;
  let F be Element of FO-WFF(A) such that
 F is universal;
  func bound_in F -> bound_FO-variable of A means
:: FO_LANG1:def 27

  ex p being Element of FO-WFF(A) st F = All(it, p);
  func the_scope_of F -> FO-formula of A means
:: FO_LANG1:def 28

  ex x being bound_FO-variable of A st F = All(x, it);
end;

reserve p for Element of FO-WFF(A);

theorem :: FO_LANG1:14
  p is negative implies len @the_argument_of p < len @p;

theorem :: FO_LANG1:15
  p is conjunctive implies len @the_left_argument_of p < len @p &
  len @the_right_argument_of p < len @p;

theorem :: FO_LANG1:16
  p is universal implies len @the_scope_of p < len @p;

scheme :: FO_LANG1:sch 2

  FOInd2 { A() -> FO-alphabet, P[Element of FO-WFF(A())] }:
    for p being Element of FO-WFF(A()) holds P[p]
provided
 for p being Element of FO-WFF(A()) holds (p is atomic implies P[p]) & P[
VERUM(A())] & (p is negative & P[the_argument_of p] implies P[p]) & (p is
conjunctive & P[the_left_argument_of p] & P[the_right_argument_of p] implies P[
p]) & (p is universal & P[the_scope_of p] implies P[p]);

reserve F for Element of FO-WFF(A);

theorem :: FO_LANG1:17
  for k being Element of NAT, P being FO-pred_symbol of k, A holds P
  `1 <> 0 & P`1 <> 1 & P`1 <> 2 & P`1 <> 3;

theorem :: FO_LANG1:18
  (@VERUM(A).1)`1 = 0 & (F is atomic implies ex k being Element of
NAT st @F.1 is FO-pred_symbol of k, A) & (F is negative implies (@F.1)`1 = 1) &
(F is conjunctive implies (@F.1)`1 = 2) & (F is universal implies (@F.1)`1 = 3)
;

theorem :: FO_LANG1:19
  F is atomic implies (@F.1)`1 <> 0 & (@F.1)`1 <> 1 & (@F.1)`1 <>
  2 & (@F.1)`1 <> 3;

reserve p for Element of FO-WFF(A);

theorem :: FO_LANG1:20
  not (VERUM(A) is atomic or VERUM(A) is negative or VERUM(A) is
  conjunctive or VERUM(A) is universal)
  & not (ex p st p is atomic & p is negative
  or p is atomic & p is conjunctive or p is atomic & p is universal or p is
  negative & p is conjunctive or p is negative & p is universal or p is
  conjunctive & p is universal);

scheme :: FO_LANG1:sch 3

  FOFuncEx { Al() -> FO-alphabet, D() -> non empty set,
           V() -> (Element of D()),
           A(Element of FO-WFF(Al())) -> (Element of D()),
           N(Element of D()) -> (Element of D()),
           C((Element of D()), Element of D()) -> (Element of D()),
           Q((Element of FO-WFF(Al())), Element of D()) -> Element of D()} :
        ex F being Function of FO-WFF(Al()), D() st F.VERUM(Al()) = V()
        & for p being Element of FO-WFF(Al()) holds
          (p is atomic implies F.p = A(p)) & (p is
          negative implies F.p = N(F.the_argument_of p))
        & (p is conjunctive implies F.p =
        C(F.the_left_argument_of p, F.the_right_argument_of p))
        & (p is universal implies F.p = Q(p, F.the_scope_of p));

reserve j,k for Element of NAT;

definition
  let A be FO-alphabet;
  let ll be FinSequence of FO-variables(A);
  func still_not-bound_in ll -> Subset of bound_FO-variables(A) equals
:: FO_LANG1:def 29
  { ll.k : 1
  <= k & k <= len ll & ll.k in bound_FO-variables(A) };
end;

reserve k for Element of NAT;

definition
  let A be FO-alphabet;
  let p be FO-formula of A;
  func still_not-bound_in p -> Subset of bound_FO-variables(A) means
:: FO_LANG1:def 30
  ex F being
  Function of FO-WFF(A), bool bound_FO-variables(A)
    st it = F.p & for p being Element
of FO-WFF(A) holds F.VERUM(A) = {}
   & (p is atomic implies F.p = { (the_arguments_of p
  ).k : 1 <= k & k <= len the_arguments_of p & (the_arguments_of p).k in
bound_FO-variables(A) })
  & (p is negative implies F.p = F.the_argument_of p) & (p
  is conjunctive implies F.p = (F.the_left_argument_of p) \/ (F.
the_right_argument_of p)) & (p is universal implies F.p = (F.the_scope_of p) \
  {bound_in p});
end;

definition
  let A be FO-alphabet;
  let p be FO-formula of A;
  attr p is closed means
:: FO_LANG1:def 31
  still_not-bound_in p = {};
end;
```

9.2 FO_LANG2

```
:: Connectives and Subformulae of the First Order Language
::  by Grzegorz Bancerek
::
:: Received November 23, 1989
:: Copyright (c) 1990-2011 Association of Mizar Users
::           (Stowarzyszenie Uzytkownikow Mizara, Bialystok, Poland).
:: This code can be distributed under the GNU General Public Licence
:: version 3.0 or later, or the Creative Commons Attribution-ShareAlike
:: License version 3.0 or later, subject to the binding interpretation
:: detailed in file COPYING.interpretation.
:: See COPYING.GPL and COPYING.CC-BY-SA for the full text of these
:: licenses, or see http://www.gnu.org/licenses/gpl.html and
:: http://creativecommons.org/licenses/by-sa/3.0/.

environ

 vocabularies NUMBERS, FINSEQ_1, FO_LANG1, SUBSET_1, ZF_LANG, XBOOLEAN,
      XXREAL_0, CARD_1, ORDINAL4, BVFUNC_2, FUNCT_1, CLASSES2, MCART_1,
      REALSET1, ARYTM_3, NAT_1, RELAT_1, ARYTM_1, TARSKI, XBOOLE_0, FO_LANG2;
 notations TARSKI, XBOOLE_0, ENUMSET1, SUBSET_1, XCMPLX_0, RELAT_1, FUNCT_1,
      NUMBERS, NAT_1, FINSEQ_1, MCART_1, FO_LANG1, XXREAL_0;
 constructors ENUMSET1, XXREAL_0, XREAL_0, NAT_1, FO_LANG1;
 registrations RELSET_1, XREAL_0, FINSEQ_1, ORDINAL1;
 requirements NUMERALS, REAL, SUBSET, BOOLE, ARITHM;
 definitions TARSKI, FO_LANG1, XBOOLE_0;
 theorems TARSKI, ENUMSET1, NAT_1, FUNCT_1, FINSEQ_1, FO_LANG1, XBOOLE_0,
      XBOOLE_1, XREAL_1, XXREAL_0, ORDINAL1;
 schemes NAT_1, XBOOLE_0;

begin

reserve A for FO-alphabet;

reserve sq for FinSequence,
  x,y,z for bound_FO-variable of A,
  p,q,p1,p2,q1 for Element of FO-WFF(A);
```

```
theorem :: FO_LANG2:1
  the_argument_of 'not' p = p;

theorem :: FO_LANG2:2
  p '&' q = p1 '&' q1 implies p = p1 & q = q1;

theorem :: FO_LANG2:3
  p is conjunctive implies p = (the_left_argument_of p) '&'
  the_right_argument_of p;

theorem :: FO_LANG2:4
  the_left_argument_of (p '&' q) = p & the_right_argument_of (p '&' q) = q;

theorem :: FO_LANG2:5
  All(x,p) = All(y,q) implies x = y & p = q;

theorem :: FO_LANG2:6
  p is universal implies p = All(bound_in p, the_scope_of p);

theorem :: FO_LANG2:7
  bound_in All(x,p) = x & the_scope_of All(x,p) = p;

definition
  let A be FO-alphabet;
  func FALSUM(A) -> FO-formula of A equals
:: FO_LANG2:def 1
  'not' VERUM(A);
  let p,q be Element of FO-WFF(A);
  func p => q -> FO-formula of A equals
:: FO_LANG2:def 2
  'not' (p '&' 'not' q);
  func p 'or' q -> FO-formula of A equals
:: FO_LANG2:def 3
  'not' ('not' p '&' 'not' q);
end;

definition
  let A be FO-alphabet;
  let p,q be Element of FO-WFF(A);
  func p <=> q -> FO-formula of A equals
:: FO_LANG2:def 4
  (p => q) '&' (q => p);
end;

definition
  let A be FO-alphabet;
  let x be bound_FO-variable of A, p be Element of FO-WFF(A);
  func Ex(x,p) -> FO-formula of A equals
:: FO_LANG2:def 5
  'not' All(x,'not' p);
end;

theorem :: FO_LANG2:8
  FALSUM(A) is negative & the_argument_of FALSUM(A) = VERUM(A);

theorem :: FO_LANG2:9
  p 'or' q = 'not' p => q;

theorem :: FO_LANG2:10
  p 'or' q = p1 'or' q1 implies p = p1 & q = q1;

theorem :: FO_LANG2:11
  p => q = p1 => q1 implies p = p1 & q = q1;

theorem :: FO_LANG2:12
  p <=> q = p1 <=> q1 implies p = p1 & q = q1;

theorem :: FO_LANG2:13
  Ex(x,p) = Ex(y,q) implies x = y & p = q;

definition
  let A be FO-alphabet;
  let x,y be bound_FO-variable of A, p be Element of FO-WFF(A);
  func All(x,y,p) -> FO-formula of A equals
:: FO_LANG2:def 6
  All(x,All(y,p));
  func Ex(x,y,p) -> FO-formula of A equals
:: FO_LANG2:def 7
  Ex(x,Ex(y,p));
end;

theorem :: FO_LANG2:14
  All(x,y,p) = All(x,All(y,p)) & Ex(x,y,p) = Ex(x,Ex(y,p));
```

```
theorem :: FO_LANG2:15
  for x1,x2,y1,y2 being bound_FO-variable of A st All(x1,y1,p1) = All(
  x2,y2,p2) holds x1 = x2 & y1 = y2 & p1 = p2;

theorem :: FO_LANG2:16
  All(x,y,p) = All(z,q) implies x = z & All(y,p) = q;

theorem :: FO_LANG2:17
  for x1,x2,y1,y2 being bound_FO-variable of A st Ex(x1,y1,p1) = Ex(x2,
  y2,p2) holds x1 = x2 & y1 = y2 & p1 = p2;

theorem :: FO_LANG2:18
  Ex(x,y,p) = Ex(z,q) implies x = z & Ex(y,p) = q;

theorem :: FO_LANG2:19
  All(x,y,p) is universal & bound_in All(x,y,p) = x & the_scope_of All(x
  ,y,p) = All(y,p);

definition
  let A be FO-alphabet;
  let x,y,z be bound_FO-variable of A, p be Element of FO-WFF(A);
  func All(x,y,z,p) -> FO-formula of A equals
:: FO_LANG2:def 8
  All(x,All(y,z,p));
  func Ex(x,y,z,p) -> FO-formula of A equals
:: FO_LANG2:def 9
  Ex(x,Ex(y,z,p));
end;

theorem :: FO_LANG2:20
  All(x,y,z,p) = All(x,All(y,z,p)) & Ex(x,y,z,p) = Ex(x,Ex(y,z,p));

theorem :: FO_LANG2:21
  for x1,x2,y1,y2,z1,z2 being bound_FO-variable of A st All(x1,y1,z1,p1) =
  All(x2,y2,z2,p2) holds x1 = x2 & y1 = y2 & z1 = z2 & p1 = p2;

reserve s,t for bound_FO-variable of A;

theorem :: FO_LANG2:22
  All(x,y,z,p) = All(t,q) implies x = t & All(y,z,p) = q;

theorem :: FO_LANG2:23
  All(x,y,z,p) = All(t,s,q) implies x = t & y = s & All(z,p) = q;

theorem :: FO_LANG2:24
  for x1,x2,y1,y2,z1,z2 being bound_FO-variable of A st Ex(x1,y1,z1,p1) = Ex(
  x2,y2,z2,p2) holds x1 = x2 & y1 = y2 & z1 = z2 & p1 = p2;

theorem :: FO_LANG2:25
  Ex(x,y,z,p) = Ex(t,q) implies x = t & Ex(y,z,p) = q;

theorem :: FO_LANG2:26
  Ex(x,y,z,p) = Ex(t,s,q) implies x = t & y = s & Ex(z,p) = q;

theorem :: FO_LANG2:27
  All(x,y,z,p) is universal & bound_in All(x,y,z,p) = x & the_scope_of
  All(x,y,z,p) = All(y,z,p);

definition
  let A be FO-alphabet;
  let H be Element of FO-WFF(A);
  attr H is disjunctive means
:: FO_LANG2:def 10
  ex p,q being Element of FO-WFF(A) st H = p 'or' q;
  attr H is conditional means
:: FO_LANG2:def 11

  ex p,q being Element of FO-WFF(A) st H = p => q;
  attr H is biconditional means
:: FO_LANG2:def 12
  ex p,q being Element of FO-WFF(A) st H = p <=> q;
  attr H is existential means
:: FO_LANG2:def 13

  ex x being bound_FO-variable of A, p being Element of FO-WFF(A)
   st H = Ex(x,p);
end;

theorem :: FO_LANG2:28
  Ex(x,y,p) is existential & Ex(x,y,z,p) is existential;

definition
  let A be FO-alphabet;
  let H be Element of FO-WFF(A);
  func the_left_disjunct_of H -> FO-formula of A equals
:: FO_LANG2:def 14
  the_argument_of
  the_left_argument_of the_argument_of H;
  func the_right_disjunct_of H -> FO-formula of A equals
:: FO_LANG2:def 15
  the_argument_of
  the_right_argument_of the_argument_of H;
  func the_antecedent_of H -> FO-formula of A equals
:: FO_LANG2:def 16
  the_left_argument_of
  the_argument_of H;
end;

notation
  let A be FO-alphabet;
  let H be Element of FO-WFF(A);
  synonym the_consequent_of H for the_right_disjunct_of H;
end;

definition
  let A be FO-alphabet;
  let H be Element of FO-WFF(A);
  func the_left_side_of H -> FO-formula of A equals
:: FO_LANG2:def 17
  the_antecedent_of
  the_left_argument_of H;
  func the_right_side_of H -> FO-formula of A equals
:: FO_LANG2:def 18
  the_consequent_of
  the_left_argument_of H;
end;

reserve F,G,H,H1 for Element of FO-WFF(A);

theorem :: FO_LANG2:29
  the_left_disjunct_of(F 'or' G) = F & the_right_disjunct_of(F
  'or' G) = G & the_argument_of F 'or' G = 'not' F '&' 'not' G;

theorem :: FO_LANG2:30
  the_antecedent_of(F => G) = F & the_consequent_of(F => G) = G &
  the_argument_of F => G = F '&' 'not' G;

theorem :: FO_LANG2:31
  the_left_side_of(F <=> G) = F & the_right_side_of(F <=> G) = G &
the_left_argument_of(F <=> G) = F => G & the_right_argument_of(F <=> G) = G =>
  F;

theorem :: FO_LANG2:32
  the_argument_of Ex(x,H) = All(x,'not' H);

theorem :: FO_LANG2:33
  H is disjunctive implies H is conditional & H is negative &
  the_argument_of H is conjunctive & the_left_argument_of the_argument_of H is
  negative & the_right_argument_of the_argument_of H is negative;

theorem :: FO_LANG2:34
  H is conditional implies H is negative & the_argument_of H is
  conjunctive & the_right_argument_of the_argument_of H is negative;

theorem :: FO_LANG2:35
  H is biconditional implies H is conjunctive & the_left_argument_of H
  is conditional & the_right_argument_of H is conditional;

theorem :: FO_LANG2:36
  H is existential implies H is negative & the_argument_of H is
  universal & the_scope_of the_argument_of H is negative;

theorem :: FO_LANG2:37
  H is disjunctive implies H = (the_left_disjunct_of H) 'or' (
  the_right_disjunct_of H);

theorem :: FO_LANG2:38
  H is conditional implies H = (the_antecedent_of H) => ( the_consequent_of H);

theorem :: FO_LANG2:39
  H is biconditional implies H = (the_left_side_of H) <=> (
  the_right_side_of H);

theorem :: FO_LANG2:40
  H is existential implies H = Ex(bound_in the_argument_of H,
  the_argument_of the_scope_of the_argument_of H);
```

```
::
::  Immediate constituents of FO-formulae
::

definition
  let A be FO-alphabet;
  let G,H be Element of FO-WFF(A);
  pred G is_immediate_constituent_of H means
:: FO_LANG2:def 19

  H = 'not' G or (ex F
  being Element of FO-WFF(A) st H = G '&' F or H = F '&' G) or ex x being
  bound_FO-variable of A st H = All(x,G);
end;

reserve x,y,z for bound_FO-variable of A,
  k,n,m for Element of NAT,
  P for ( FO-pred_symbol of k, A),
  V for FO-variable_list of k, A;

theorem :: FO_LANG2:41
  not H is_immediate_constituent_of VERUM(A);

theorem :: FO_LANG2:42
  not H is_immediate_constituent_of P!V;

theorem :: FO_LANG2:43
  F is_immediate_constituent_of 'not' H iff F = H;

theorem :: FO_LANG2:44
  H is_immediate_constituent_of FALSUM(A) iff H = VERUM(A);

theorem :: FO_LANG2:45
  F is_immediate_constituent_of G '&' H iff F = G or F = H;

theorem :: FO_LANG2:46
  F is_immediate_constituent_of All(x,H) iff F = H;

theorem :: FO_LANG2:47
  H is atomic implies not F is_immediate_constituent_of H;

theorem :: FO_LANG2:48
  H is negative implies (F is_immediate_constituent_of H iff F =
  the_argument_of H);

theorem :: FO_LANG2:49
  H is conjunctive implies (F is_immediate_constituent_of H iff F
  = the_left_argument_of H or F = the_right_argument_of H);

theorem :: FO_LANG2:50
  H is universal implies (F is_immediate_constituent_of H iff F =
  the_scope_of H);

::
:: Subformulae of FO-formulae
::

reserve L,L9 for FinSequence;

definition
  let A be FO-alphabet;
  let G,H be Element of FO-WFF(A);
  pred G is_subformula_of H means
:: FO_LANG2:def 20

  ex n,L st 1 <= n & len L = n & L.1
= G & L.n = H & for k st 1 <= k & k < n ex G1,H1 being Element of FO-WFF(A)
st L.k = G1 & L.(k+1) = H1 & G1 is_immediate_constituent_of H1;
  reflexivity;
end;

definition
  let A be FO-alphabet;
  let H,F be Element of FO-WFF(A);
  pred H is_proper_subformula_of F means
:: FO_LANG2:def 21

  H is_subformula_of F & H <> F;
  irreflexivity;
end;

theorem :: FO_LANG2:51
  H is_immediate_constituent_of F implies len @H < len @F;
```

```
theorem :: FO_LANG2:52
  H is_immediate_constituent_of F implies H is_subformula_of F;

theorem :: FO_LANG2:53
  H is_immediate_constituent_of F implies H is_proper_subformula_of F;

theorem :: FO_LANG2:54
  H is_proper_subformula_of F implies len @H < len @F;

theorem :: FO_LANG2:55
  H is_proper_subformula_of F implies ex G st G is_immediate_constituent_of F;

theorem :: FO_LANG2:56
  F is_proper_subformula_of G & G is_proper_subformula_of H
  implies F is_proper_subformula_of H;

theorem :: FO_LANG2:57
  F is_subformula_of G & G is_subformula_of H implies F is_subformula_of H;

theorem :: FO_LANG2:58
  G is_subformula_of H & H is_subformula_of G implies G = H;

theorem :: FO_LANG2:59
  not (G is_proper_subformula_of H & H is_subformula_of G);

theorem :: FO_LANG2:60
  not (G is_proper_subformula_of H & H is_proper_subformula_of G);

theorem :: FO_LANG2:61
  not (G is_subformula_of H & H is_immediate_constituent_of G);

theorem :: FO_LANG2:62
  not (G is_proper_subformula_of H & H is_immediate_constituent_of G);

theorem :: FO_LANG2:63
  F is_proper_subformula_of G & G is_subformula_of H or F
  is_subformula_of G & G is_proper_subformula_of H or F is_subformula_of G & G
  is_immediate_constituent_of H or F is_immediate_constituent_of G & G
  is_subformula_of H or F is_proper_subformula_of G & G
  is_immediate_constituent_of H or F is_immediate_constituent_of G & G
  is_proper_subformula_of H implies F is_proper_subformula_of H;

theorem :: FO_LANG2:64
  not F is_proper_subformula_of VERUM(A);

theorem :: FO_LANG2:65
  not F is_proper_subformula_of P!V;

theorem :: FO_LANG2:66
  F is_subformula_of H iff F is_proper_subformula_of 'not' H;

theorem :: FO_LANG2:67
  'not' F is_subformula_of H implies F is_proper_subformula_of H;

theorem :: FO_LANG2:68
  F is_proper_subformula_of FALSUM(A) iff F is_subformula_of VERUM(A);

theorem :: FO_LANG2:69
  F is_subformula_of G or F is_subformula_of H iff F
  is_proper_subformula_of G '&' H;

theorem :: FO_LANG2:70
  F '&' G is_subformula_of H implies F is_proper_subformula_of H & G
  is_proper_subformula_of H;

theorem :: FO_LANG2:71
  F is_subformula_of H iff F is_proper_subformula_of All(x,H);

theorem :: FO_LANG2:72
  All(x,H) is_subformula_of F implies H is_proper_subformula_of F;

theorem :: FO_LANG2:73
  F '&' 'not' G is_proper_subformula_of F => G & F
  is_proper_subformula_of F => G & 'not' G is_proper_subformula_of F => G & G
  is_proper_subformula_of F => G;

theorem :: FO_LANG2:74
  'not' F '&' 'not' G is_proper_subformula_of F 'or' G & 'not' F
is_proper_subformula_of F 'or' G & 'not' G is_proper_subformula_of F 'or' G & F
  is_proper_subformula_of F 'or' G & G is_proper_subformula_of F 'or' G;

theorem :: FO_LANG2:75
  H is atomic implies not F is_proper_subformula_of H;
```

```
theorem :: FO_LANG2:76
  H is negative implies the_argument_of H is_proper_subformula_of H;

theorem :: FO_LANG2:77
  H is conjunctive implies the_left_argument_of H
  is_proper_subformula_of H & the_right_argument_of H is_proper_subformula_of H
;

theorem :: FO_LANG2:78
  H is universal implies the_scope_of H is_proper_subformula_of H;

theorem :: FO_LANG2:79
  H is_subformula_of VERUM(A) iff H = VERUM(A);

theorem :: FO_LANG2:80
  H is_subformula_of P!V iff H = P!V;

theorem :: FO_LANG2:81
  H is_subformula_of FALSUM(A) iff H = FALSUM(A) or H = VERUM(A);

::
::   Set of subformulae of FO-formulae
::

definition
  let A be FO-alphabet;
  let H be Element of FO-WFF(A);
  func Subformulae H -> set means
:: FO_LANG2:def 22

  for a being set holds a in it iff ex F being Element of FO-WFF(A)
   st F = a & F is_subformula_of H;
end;

theorem :: FO_LANG2:82
  G in Subformulae H implies G is_subformula_of H;

theorem :: FO_LANG2:83
  F is_subformula_of H implies Subformulae F c= Subformulae H;

theorem :: FO_LANG2:84
  G in Subformulae H implies Subformulae G c= Subformulae H;

theorem :: FO_LANG2:85
  Subformulae(VERUM(A)) = { VERUM(A) };

theorem :: FO_LANG2:86
  Subformulae(P!V) = { P!V };

theorem :: FO_LANG2:87
  Subformulae(FALSUM(A)) = { VERUM(A), FALSUM(A) };

theorem :: FO_LANG2:88
  Subformulae 'not' H = Subformulae H \/ { 'not' H };

theorem :: FO_LANG2:89
  Subformulae (H '&' F) = Subformulae H \/ Subformulae F \/ { H '&' F };

theorem :: FO_LANG2:90
  Subformulae All(x,H) = Subformulae H \/ { All(x,H) };

theorem :: FO_LANG2:91
  Subformulae (F => G) = Subformulae F \/ Subformulae G \/ {
  'not' G, F '&' 'not' G, F => G };

theorem :: FO_LANG2:92
  Subformulae (F 'or' G) = Subformulae F \/ Subformulae G \/ {'not' G,
  'not' F, 'not' F '&' 'not' G, F 'or' G};

theorem :: FO_LANG2:93
  Subformulae (F <=> G) = Subformulae F \/ Subformulae G \/ { 'not' G, F
  '&' 'not' G, F => G, 'not' F, G '&' 'not' F, G => F, F <=> G };

theorem :: FO_LANG2:94
  H = VERUM(A) or H is atomic iff Subformulae H = { H };

theorem :: FO_LANG2:95
  H is negative implies Subformulae H = Subformulae the_argument_of H \/ { H };

theorem :: FO_LANG2:96
  H is conjunctive implies Subformulae H = Subformulae
  the_left_argument_of H \/ Subformulae the_right_argument_of H \/ { H };

theorem :: FO_LANG2:97
```

```
  H is universal implies Subformulae H = Subformulae the_scope_of H \/ { H };

theorem :: FO_LANG2:98
  (H is_immediate_constituent_of G or H is_proper_subformula_of G or H
  is_subformula_of G) & G in Subformulae F implies H in Subformulae F;
```

9.3 FO_LANG3

```
:: Variables in Formulae of the First Order Language
::  by Czes{\l}aw Byli\'nski and Grzegorz Bancerek
::
:: Received November 23, 1989
:: Copyright (c) 1990-2011 Association of Mizar Users
::           (Stowarzyszenie Uzytkownikow Mizara, Bialystok, Poland).
:: This code can be distributed under the GNU General Public Licence
:: version 3.0 or later, or the Creative Commons Attribution-ShareAlike
:: License version 3.0 or later, subject to the binding interpretation
:: detailed in file COPYING.interpretation.
:: See COPYING.GPL and COPYING.CC-BY-SA for the full text of these
:: licenses, or see http://www.gnu.org/licenses/gpl.html and
:: http://creativecommons.org/licenses/by-sa/3.0/.

environ

 vocabularies SUBSET_1, NUMBERS, FO_LANG1, FINSEQ_1, XBOOLE_0, FUNCT_1,
      ZF_LANG, XXREAL_0, CLASSES2, REALSET1, BVFUNC_2, XBOOLEAN, MARGREL1,
      ZF_LANG1, TARSKI, ZFMISC_1, FO_LANG2, RCOMP_1, ZF_MODEL, FO_LANG3;
 notations TARSKI, XBOOLE_0, ZFMISC_1, SUBSET_1, NUMBERS, FUNCT_1, FUNCT_2,
      FINSEQ_1, FO_LANG1, FO_LANG2, XXREAL_0;
 constructors FUNCT_2, XXREAL_0, MEMBERED, FO_LANG2, RELSET_1;
 registrations SUBSET_1, ORDINAL1, RELSET_1, MEMBERED, FO_LANG1;
 requirements NUMERALS, BOOLE, SUBSET;
 definitions TARSKI, FO_LANG1;
 theorems TARSKI, ZFMISC_1, FUNCT_2, FO_LANG1, FO_LANG2, XBOOLE_1;
 schemes FO_LANG1;

begin

reserve i,k for Element of NAT;

scheme :: FO_LANG3:sch 1

  FOFuncUniq { Al() -> FO-alphabet, D() -> non empty set,
              F1() -> (Function of FO-WFF(Al()), D()),
              F2() -> (Function of FO-WFF(Al()), D()),
              V() -> (Element of D()),
              A(set) -> (Element of D()),
              N(set) -> (Element of D()),
              C(set,set) -> (Element of D()),
              Q(set,set) -> Element of D()}
       :
   F1() = F2()
provided
 for p being Element of FO-WFF(Al()) for d1,d2 being Element of D() holds
(p = VERUM(Al()) implies F1().p = V()) &
(p is atomic implies F1().p = A(p)) & (p is negative &
d1 = F1().the_argument_of p implies F1().p = N(d1)) &
(p is conjunctive & d1 = F1().the_left_argument_of p &
d2 = F1().the_right_argument_of p implies F1().p = C(
d1, d2)) & (p is universal & d1 = F1().the_scope_of p implies
F1().p = Q(p, d1)) and
 for p being Element of FO-WFF(Al()) for d1,d2 being Element of D() holds
(p = VERUM(Al()) implies F2().p = V()) & (p is atomic implies F2().p = A(p)) &
(p is negative & d1 = F2().the_argument_of p implies F2().p = N(d1)) &
(p is conjunctive & d1 = F2().the_left_argument_of p &
d2 = F2().the_right_argument_of p implies F2().p = C(d1, d2)) &
(p is universal & d1 = F2().the_scope_of p implies F2().p = Q(p, d1));

scheme :: FO_LANG3:sch 2

  FODefD { Al() -> FO-alphabet,
          D() -> non empty set,
          V() -> (Element of D()),
          p() -> (Element of FO-WFF(Al())),
          A(Element of FO-WFF(Al())) -> (Element of D()),
          N(Element of D()) -> (Element of D()),
          C((Element of D()),
          Element of D()) -> (Element of D()),
          Q((Element of FO-WFF(Al())),
          Element of D()) -> Element of D()}
      :
```

```
      (ex d being (Element of D()), F being Function of FO-WFF(Al()),
      D() st d = F.p() & for p being Element of FO-WFF(Al())
      for d1,d2 being Element of D() holds
      (p = VERUM(Al()) implies F.p = V()) & (p is atomic implies F.p = A(p)) &
      (p is negative & d1 = F.the_argument_of p implies F.p = N(d1)) &
      (p is conjunctive & d1 = F.the_left_argument_of p &
       d2 = F.the_right_argument_of p implies F.p = C(d1, d2)) &
      (p is universal & d1 = F.the_scope_of p implies F.p = Q(p, d1)) ) &
      for x1,x2 being Element of D() st (ex F being Function of FO-WFF(Al()),
      D() st x1 = F.p() & for p being Element of FO-WFF(Al())
      for d1,d2 being Element of D() holds
      (p = VERUM(Al()) implies F.p = V()) & (p is atomic implies F.p = A(p)) &
      (p is negative & d1 = F.the_argument_of p implies F.p = N(d1)) &
      (p is conjunctive & d1 = F.the_left_argument_of p &
       d2 = F.the_right_argument_of p implies F.p = C(d1, d2)) &
      (p is universal & d1 = F.the_scope_of p implies F.p = Q(p, d1)) ) &
      (ex F being Function of FO-WFF(Al()), D() st x2 = F.p() &
      for p being Element of FO-WFF(Al()) for d1,d2 being Element of D()
      holds (p = VERUM(Al()) implies F.p = V()) &
      (p is atomic implies F.p = A(p)) &
      (p is negative & d1 = F.the_argument_of p implies F.p = N(d1)) &
      (p is conjunctive & d1 = F.the_left_argument_of p &
      d2 = F.the_right_argument_of p implies F.p = C(d1, d2)) &
      (p is universal & d1 = F.the_scope_of p implies F.p = Q(p, d1)) )
      holds x1 = x2;

scheme :: FO_LANG3:sch 3

  FODResult9VERUM { Al() -> FO-alphabet, D() -> non empty set,
    F(Element of FO-WFF(Al())) -> (Element of D()), V() -> (Element of D()),
    A(Element of FO-WFF(Al())) -> (Element of D()),
    N(Element of D()) -> (Element of D()), C((Element of D()),
    Element of D()) -> (Element of D()), Q((Element of FO-WFF(Al())),
    Element of D()) -> Element of D()} : F(VERUM(Al())) = V()
provided
 for p being FO-formula of Al(), d being Element of D() holds d = F(p)
iff ex F being Function of FO-WFF(Al()), D() st d = F.p &
for p being Element of FO-WFF(Al()) for
d1,d2 being Element of D() holds (p = VERUM(Al())
implies F.p = V()) & (p is atomic
implies F.p = A(p)) & (p is negative & d1 = F.the_argument_of p implies F.p = N
(d1)) & (p is conjunctive & d1 = F.the_left_argument_of p & d2 = F.
the_right_argument_of p implies F.p = C(d1, d2)) & (p is universal & d1 = F.
the_scope_of p implies F.p = Q(p, d1));

scheme :: FO_LANG3:sch 4

  FODResult9atomic { Al() -> FO-alphabet, D() -> non empty set,
    V() -> (Element of D()), F(Element of FO-WFF(Al())) -> (Element of D()),
    p() -> FO-formula of Al(), A(Element of FO-WFF(Al())) -> (Element of D()),
    N(Element of D()) -> (Element of D()), C((Element of D()),
    Element of D()) -> (Element of D()),
    Q((Element of FO-WFF(Al())), Element of D()) -> Element of D()}
  :
F(p()) = A(p())
provided
 for p being FO-formula of Al(), d being Element of D() holds d = F(p)
iff ex F being Function of FO-WFF(Al()), D() st d = F.p &
for p being Element of FO-WFF(Al()) for
d1,d2 being Element of D() holds (p = VERUM(Al()) implies F.p = V()) &
(p is atomic implies F.p = A(p)) & (p is negative &
d1 = F.the_argument_of p implies F.p = N(d1)) &
(p is conjunctive & d1 = F.the_left_argument_of p & d2 = F.
the_right_argument_of p implies F.p = C(d1, d2)) &
(p is universal & d1 = F.the_scope_of p implies F.p = Q(p, d1)) and
 p() is atomic;

scheme :: FO_LANG3:sch 5

  FODResult9negative { Al() -> FO-alphabet, D() -> non empty set,
    V() -> (Element of D()), p() -> FO-formula of Al(),
    A(Element of FO-WFF(Al())) -> (Element of D()),
    N(Element of D()) -> (Element of D()), C((Element of D()),
    Element of D()) -> (Element of D()), Q((Element of FO-WFF(Al())),
    Element of D()) -> (Element of D()), F(Element of FO-WFF(Al()))
      -> (Element of D()) }
  :
F(p()) = N(F(the_argument_of p()))
provided
 for p being FO-formula of Al(), d being Element of D() holds d = F(p)
iff ex F being Function of FO-WFF(Al()), D() st d = F.p
& for p being Element of FO-WFF(Al()) for
d1,d2 being Element of D() holds (p = VERUM(Al()) implies F.p = V())
& (p is atomic implies F.p = A(p)) &
```

```
(p is negative & d1 = F.the_argument_of p implies F.p = N (d1)) &
(p is conjunctive & d1 = F.the_left_argument_of p & d2 = F.
the_right_argument_of p implies F.p = C(d1, d2)) &
(p is universal & d1 = F.the_scope_of p implies F.p = Q(p, d1)) and
 p() is negative;

scheme :: FO_LANG3:sch 6

  FODResult9conjunctive { Al() -> FO-alphabet, D() -> non empty set,
    V() -> (Element of D()), A(Element of FO-WFF(Al())) -> (Element of D()),
    N(Element of D()) -> (Element of D()), C((Element of D()),
    Element of D()) -> (Element of D()), Q((Element of FO-WFF(Al())),
    Element of D()) -> (Element of D()),
    F(Element of FO-WFF(Al())) -> (Element of D()),
    p() -> FO-formula of Al() } : for d1,d2 being Element of D() st d1 = F(
    the_left_argument_of p()) & d2 = F(the_right_argument_of p())
    holds F(p()) = C(d1,d2)
provided
 for p being FO-formula of Al(), d being Element of D() holds d = F(p)
iff ex F being Function of FO-WFF(Al()), D() st d = F.p &
for p being Element of FO-WFF(Al()) for
d1,d2 being Element of D() holds (p = VERUM(Al()) implies F.p = V()) &
(p is atomic implies F.p = A(p)) &
(p is negative & d1 = F.the_argument_of p implies F.p = N
(d1)) & (p is conjunctive & d1 = F.the_left_argument_of p & d2 = F.
the_right_argument_of p implies F.p = C(d1, d2)) & (p is universal & d1 = F.
the_scope_of p implies F.p = Q(p, d1)) and
 p() is conjunctive;

scheme :: FO_LANG3:sch 7

  FODResult9universal { Al() -> FO-alphabet, D() -> non empty set,
    V() -> (Element of D()), p() -> FO-formula of Al(),
    A(Element of FO-WFF(Al())) -> (Element of D()),
    N(Element of D()) -> (Element of D()), C((Element of D()),
    Element of D()) -> (Element of D()), Q((Element of FO-WFF(Al())),
    Element of D()) -> (Element of D()),
    F(Element of FO-WFF(Al())) -> (Element of D()) }
  :
  F(p()) = Q(p(),F(the_scope_of p()))
provided
 for p being FO-formula of Al(), d being Element of D() holds d = F(p)
iff ex F being Function of FO-WFF(Al()), D() st d = F.p &
for p being Element of FO-WFF(Al()) for
d1,d2 being Element of D() holds (p = VERUM(Al()) implies F.p = V()) &
(p is atomic implies F.p = A(p)) & (p is negative &
d1 = F.the_argument_of p implies F.p = N(d1)) & (p is conjunctive &
d1 = F.the_left_argument_of p & d2 = F.the_right_argument_of p implies
F.p = C(d1, d2)) & (p is universal &
d1 = F.the_scope_of p implies F.p = Q(p, d1)) and
 p() is universal;

reserve A for FO-alphabet;
reserve x for bound_FO-variable of A;
reserve a for free_FO-variable of A;
reserve p,q for Element of FO-WFF(A);
reserve l for FinSequence of FO-variables(A);
reserve P,Q for FO-pred_symbol of A;
reserve V for non empty Subset of FO-variables(A);
reserve s,t for FO-symbol of A;

theorem :: FO_LANG3:1
  P is FO-pred_symbol of the_arity_of P, A;

definition
  let A;
  let l;
  let V;
  func variables_in(l,V) -> Subset of V equals
:: FO_LANG3:def 1
  { l.k : 1 <= k & k <= len l & l
  .k in V };
end;

theorem :: FO_LANG3:2
  still_not-bound_in l = variables_in(l,bound_FO-variables(A));

theorem :: FO_LANG3:3
  still_not-bound_in VERUM(A) = {};

theorem :: FO_LANG3:4
  for p being FO-formula of A st p is atomic holds still_not-bound_in p
  = still_not-bound_in the_arguments_of p;
```

```
theorem :: FO_LANG3:5
  for P being FO-pred_symbol of k,A for l being FO-variable_list of k, A
  holds still_not-bound_in (P!l) = still_not-bound_in l;

theorem :: FO_LANG3:6
  for p being FO-formula of A st p is negative holds still_not-bound_in
  p = still_not-bound_in the_argument_of p;

theorem :: FO_LANG3:7
  for p being FO-formula of A holds still_not-bound_in 'not' p =
  still_not-bound_in p;

theorem :: FO_LANG3:8
  still_not-bound_in FALSUM(A) = {};

theorem :: FO_LANG3:9
  for p being FO-formula of A st p is conjunctive holds
  still_not-bound_in p = (still_not-bound_in the_left_argument_of p) \/ (
  still_not-bound_in the_right_argument_of p);

theorem :: FO_LANG3:10
  for p,q being FO-formula of A holds still_not-bound_in(p '&' q) = (
  still_not-bound_in p) \/ (still_not-bound_in q);

theorem :: FO_LANG3:11
  for p being FO-formula of A st p is universal holds
  still_not-bound_in p = (still_not-bound_in the_scope_of p) \ {bound_in p};

theorem :: FO_LANG3:12
  for p being FO-formula of A holds still_not-bound_in All(x,p) = (
  still_not-bound_in p) \ {x};

theorem :: FO_LANG3:13
  for p being FO-formula of A st p is disjunctive holds
  still_not-bound_in p = (still_not-bound_in the_left_disjunct_of p) \/ (
  still_not-bound_in the_right_disjunct_of p);

theorem :: FO_LANG3:14
  for p,q being FO-formula of A holds still_not-bound_in p 'or' q = (
  still_not-bound_in p) \/ (still_not-bound_in q);

theorem :: FO_LANG3:15
  for p being FO-formula of A st p is conditional holds
  still_not-bound_in p = (still_not-bound_in the_antecedent_of p) \/ (
  still_not-bound_in the_consequent_of p);

theorem :: FO_LANG3:16
  for p,q being FO-formula of A holds still_not-bound_in p => q = (
  still_not-bound_in p) \/ (still_not-bound_in q);

theorem :: FO_LANG3:17
  for p being FO-formula of A st p is biconditional holds
  still_not-bound_in p = (still_not-bound_in the_left_side_of p) \/ (
  still_not-bound_in the_right_side_of p);

theorem :: FO_LANG3:18
  for p,q being FO-formula of A holds still_not-bound_in p <=> q = (
  still_not-bound_in p) \/ (still_not-bound_in q);

theorem :: FO_LANG3:19
  for p being FO-formula of A holds still_not-bound_in Ex(x,p) = (
  still_not-bound_in p) \ {x};

theorem :: FO_LANG3:20
  VERUM(A) is closed & FALSUM(A) is closed;

theorem :: FO_LANG3:21
  for p being FO-formula of A holds p is closed iff 'not' p is closed;

theorem :: FO_LANG3:22
  for p,q being FO-formula of A holds p is closed & q is closed iff p
  '&' q is closed;

theorem :: FO_LANG3:23
  for p being FO-formula of A holds All(x,p) is closed iff
  still_not-bound_in p c= {x};

theorem :: FO_LANG3:24
  for p being FO-formula of A st p is closed holds All(x,p) is closed;

theorem :: FO_LANG3:25
  for p,q being FO-formula of A holds p is closed & q is closed iff p 'or' q
  is closed;
```

```
theorem :: FO_LANG3:26
  for p,q being FO-formula of A holds p is closed &
   q is closed iff p => q is closed;

theorem :: FO_LANG3:27
  for p,q being FO-formula of A holds p is closed & q is closed iff p <=> q
  is closed;

theorem :: FO_LANG3:28
  for p being FO-formula of A holds Ex(x,p) is closed iff
  still_not-bound_in p c= {x};

theorem :: FO_LANG3:29
  for p being FO-formula of A st p is closed holds Ex(x,p) is closed;

definition
  let A;
  let s;
  func x.s -> bound_FO-variable of A equals
:: FO_LANG3:def 2
  [4,s];
end;

theorem :: FO_LANG3:30
  ex t st x.t = x;

definition
  let A;
  let k;
  func (A)a.k -> free_FO-variable of A equals
:: FO_LANG3:def 3
  [6,k];
end;

theorem :: FO_LANG3:31
  ex i st (A)a.i = a;

theorem :: FO_LANG3:32
  for c being Element of fixed_FO-variables(A) for a being Element of
  free_FO-variables(A) holds c <> a;

theorem :: FO_LANG3:33
  for c being Element of fixed_FO-variables(A) for x being Element of
  bound_FO-variables(A) holds c <> x;

theorem :: FO_LANG3:34
  for a being Element of free_FO-variables(A) for x being Element of
  bound_FO-variables(A) holds a <> x;

definition
  let A;
  let V;
  let p;
  func Vars(p,V) -> Subset of V means
:: FO_LANG3:def 4

  ex F being Function of FO-WFF(A),
bool V st it = F.p & for p being Element of FO-WFF(A)
for d1,d2 being Subset of V
holds (p = VERUM(A) implies F.p = {}(V)) &
(p is atomic implies F.p = variables_in
(the_arguments_of p,V)) & (p is negative & d1 = F.the_argument_of p implies F.p
  = d1) & (p is conjunctive & d1 = F.the_left_argument_of p & d2 = F.
  the_right_argument_of p implies F.p = d1 \/ d2) & (p is universal & d1 = F.
  the_scope_of p implies F.p = d1);
end;

theorem :: FO_LANG3:35
  Vars(VERUM(A),V) = {};

theorem :: FO_LANG3:36
  p is atomic implies Vars(p,V) = variables_in(the_arguments_of p,
V) & Vars(p,V) = { (the_arguments_of p).k : 1 <= k & k <= len (the_arguments_of
  p) & (the_arguments_of p).k in V };

theorem :: FO_LANG3:37
  for P being FO-pred_symbol of k, A for l being FO-variable_list of
  k, A holds Vars(P!l,V) = variables_in(l, V) &
  Vars((P!l),V) = { l.i : 1 <= i & i
  <= len l & l.i in V };

theorem :: FO_LANG3:38
  p is negative implies Vars(p,V) = Vars(the_argument_of p,V);
```

```
theorem :: FO_LANG3:39
  Vars('not' p,V) = Vars(p,V);

theorem :: FO_LANG3:40
  Vars(FALSUM(A),V) = {};

theorem :: FO_LANG3:41
  p is conjunctive implies Vars(p,V) = Vars(the_left_argument_of p,V) \/
  Vars(the_right_argument_of p,V);

theorem :: FO_LANG3:42
  Vars(p '&' q,V) = Vars(p,V) \/ Vars(q,V);

theorem :: FO_LANG3:43
  p is universal implies Vars(p,V) = Vars(the_scope_of p,V);

theorem :: FO_LANG3:44
  Vars(All(x,p),V) = Vars(p,V);

theorem :: FO_LANG3:45
  p is disjunctive implies Vars(p,V) = Vars(the_left_disjunct_of p
  ,V) \/ Vars(the_right_disjunct_of p,V);

theorem :: FO_LANG3:46
  Vars(p 'or' q, V) = Vars(p,V) \/ Vars(q,V);

theorem :: FO_LANG3:47
  p is conditional implies Vars(p,V) = Vars(the_antecedent_of p,V)
  \/ Vars(the_consequent_of p,V);

theorem :: FO_LANG3:48
  Vars(p => q,V) = Vars(p,V) \/ Vars(q,V);

theorem :: FO_LANG3:49
  p is biconditional implies Vars(p,V) = Vars(the_left_side_of p,V
  ) \/ Vars(the_right_side_of p,V);

theorem :: FO_LANG3:50
  Vars(p <=> q,V) = Vars(p,V) \/ Vars(q,V);

theorem :: FO_LANG3:51
  p is existential implies Vars(p,V) = Vars(the_argument_of the_scope_of
  the_argument_of p, V);

theorem :: FO_LANG3:52
  Vars(Ex(x,p), V) = Vars(p,V);

definition
  let A;
  let p;
  func Free p -> Subset of free_FO-variables(A) equals
:: FO_LANG3:def 5
  Vars(p,free_FO-variables(A));
end;

theorem :: FO_LANG3:53
  Free VERUM(A) = {};

theorem :: FO_LANG3:54
  for P being FO-pred_symbol of k, A for l being FO-variable_list of k, A
  holds Free(P!l) = { l.i : 1 <= i & i <= len l & l.i in free_FO-variables(A)};

theorem :: FO_LANG3:55
  Free 'not' p = Free p;

theorem :: FO_LANG3:56
  Free FALSUM(A) = {};

theorem :: FO_LANG3:57
  Free(p '&' q) = Free p \/ Free q;

theorem :: FO_LANG3:58
  Free(All(x,p)) = Free(p);

theorem :: FO_LANG3:59
  Free(p 'or' q) = Free p \/ Free q;

theorem :: FO_LANG3:60
  Free(p => q) = Free p \/ Free q;

theorem :: FO_LANG3:61
  Free(p <=> q) = Free p \/ Free q;

theorem :: FO_LANG3:62
```

```
  Free Ex(x,p) = Free p;

definition
  let A;
  let p;
  func Fixed p -> Subset of fixed_FO-variables(A) equals
:: FO_LANG3:def 6
  Vars(p,
  fixed_FO-variables(A));
end;

theorem :: FO_LANG3:63
  Fixed VERUM(A) = {};

theorem :: FO_LANG3:64
  for P being FO-pred_symbol of k,A for l being FO-variable_list of k,A
holds Fixed(P!l) = { l.i : 1 <= i & i <= len l & l.i in fixed_FO-variables(A)};

theorem :: FO_LANG3:65
  Fixed 'not' p = Fixed p;

theorem :: FO_LANG3:66
  Fixed FALSUM(A) = {};

theorem :: FO_LANG3:67
  Fixed(p '&' q) = Fixed p \/ Fixed q;

theorem :: FO_LANG3:68
  Fixed(All(x,p)) = Fixed(p);

theorem :: FO_LANG3:69
  Fixed(p 'or' q) = Fixed p \/ Fixed q;

theorem :: FO_LANG3:70
  Fixed(p => q) = Fixed p \/ Fixed q;

theorem :: FO_LANG3:71
  Fixed(p <=> q) = Fixed p \/ Fixed q;

theorem :: FO_LANG3:72
  Fixed Ex(x,p) = Fixed p;
```

9.4 CFO_LANG

```
:: A Classical First Order Language
::  by Czes{\l}aw Byli\'nski
::
:: Received May 11, 1990
:: Copyright (c) 1990-2011 Association of Mizar Users
::           (Stowarzyszenie Uzytkownikow Mizara, Bialystok, Poland).
:: This code can be distributed under the GNU General Public Licence
:: version 3.0 or later, or the Creative Commons Attribution-ShareAlike
:: License version 3.0 or later, subject to the binding interpretation
:: detailed in file COPYING.interpretation.
:: See COPYING.GPL and COPYING.CC-BY-SA for the full text of these
:: licenses, or see http://www.gnu.org/licenses/gpl.html and
:: http://creativecommons.org/licenses/by-sa/3.0/.

environ

 vocabularies SUBSET_1, NUMBERS, FO_LANG1, FINSEQ_1, PARTFUN1, XXREAL_0,
      FUNCT_1, RELAT_1, NAT_1, TARSKI, FUNCOP_1, FO_LANG3, XBOOLE_0, ZF_MODEL,
      FINSEQ_2, ZF_LANG, CARD_1, REALSET1, XBOOLEAN, BVFUNC_2, MARGREL1,
      CLASSES2, FUNCT_4, CFO_LANG, ZFMISC_1;
 notations TARSKI, XBOOLE_0, ENUMSET1, SUBSET_1, CARD_1, NUMBERS, RELAT_1,
      FUNCT_1, FUNCT_2, BINOP_1, PARTFUN1, FUNCOP_1, FUNCT_4, FINSEQ_1,
      FINSEQ_2, FO_LANG1, FO_LANG2, FO_LANG3, XXREAL_0, NAT_1, ZFMISC_1;
 constructors ENUMSET1, PARTFUN1, BINOP_1, FUNCOP_1, FUNCT_4, XXREAL_0,
      MEMBERED, FO_LANG2, FO_LANG3, FINSEQ_2, RELSET_1;
 registrations XBOOLE_0, RELSET_1, FUNCOP_1, FUNCT_4, MEMBERED, FO_LANG1,
      XXREAL_0, FINSEQ_2, CARD_1;
 requirements NUMERALS, SUBSET, BOOLE;
 definitions TARSKI, FUNCOP_1, FINSEQ_2, RELAT_1;
 theorems TARSKI, ZFMISC_1, FUNCT_1, FUNCT_2, FINSEQ_1, PARTFUN1, FUNCOP_1,
      FO_LANG1, FO_LANG2, FO_LANG3, FINSEQ_2, RELSET_1, FINSEQ_3, FUNCT_4,
      ORDINAL1, RELAT_1, CARD_1, XBOOLE_0;
 schemes FINSEQ_1, FO_LANG1, FO_LANG3;

begin

reserve A for FO-alphabet;
reserve i,j,k for Element of NAT;
```

```
theorem :: CFO_LANG:1
  for x being set holds x in FO-variables(A) iff x in
  fixed_FO-variables(A) or x in free_FO-variables(A) or
  x in bound_FO-variables(A);

definition
  let A;
  mode Substitution of A is PartFunc of free_FO-variables(A),FO-variables(A);
end;

reserve f for Substitution of A;

definition
  let A;
  let l be FinSequence of FO-variables(A);
  let f;
  func Subst(l,f) -> FinSequence of FO-variables(A) means
:: CFO_LANG:def 1

  len it = len l &
for k st 1 <= k & k <= len l holds (l.k in dom f implies it.k = f.(l.k)) & (not
  l.k in dom f implies it.k = l.k);
end;

registration
  let A;
  let k;
  let l be FO-variable_list of k, A;
  let f;
  cluster Subst(l,f) -> k-element;
end;

theorem :: CFO_LANG:2
  for x being bound_FO-variable of A, a being free_FO-variable of A holds
   a .--> x is Substitution of A;

definition
  let A;
  let a be free_FO-variable of A, x be bound_FO-variable of A;
  redefine func a .--> x -> Substitution of A;
end;

theorem :: CFO_LANG:3
  for x being bound_FO-variable of A, a being free_FO-variable of A,
  l, ll being FinSequence of FO-variables(A)
  holds
  f = a .--> x & ll = Subst(l,f) & 1 <= k & k <= len l implies (l.
  k = a implies ll.k = x) & (l.k <> a implies ll.k = l.k);

definition
  let A;
  func CFO-WFF(A) -> Subset of FO-WFF(A) equals
:: CFO_LANG:def 2
  {s where s is FO-formula of A: Fixed s = {} & Free s = {} };
end;

registration
  let A;
  cluster CFO-WFF(A) -> non empty;
end;

theorem :: CFO_LANG:4
  for p being Element of FO-WFF(A) holds
  p is Element of CFO-WFF(A) iff Fixed p = {} & Free p = {};

registration
  let A;
  let k;
  cluster bound_FO-variables(A)-valued for FO-variable_list of k, A;
end;

definition
  let A;
  let k;
  mode CFO-variable_list of k, A is bound_FO-variables(A)-valued
       FO-variable_list of k,A;
end;

theorem :: CFO_LANG:5
  for l being FO-variable_list of k, A holds l is CFO-variable_list
of k,A iff { l.i : 1 <= i & i <= len l & l.i in free_FO-variables(A) }
           = {} & { l.j
  : 1 <= j & j <= len l & l.j in fixed_FO-variables(A) } = {};
```

```
theorem :: CFO_LANG:6
  VERUM(A) is Element of CFO-WFF(A);

theorem :: CFO_LANG:7
  for P being FO-pred_symbol of k,A for l being FO-variable_list of
  k,A holds P!l is Element of CFO-WFF(A) iff
  { l.i : 1 <= i & i <= len l & l.i in
  free_FO-variables(A) } = {} & { l.j : 1 <= j & j <= len l & l.j in
  fixed_FO-variables(A) } = {};

definition
  let k;
  let A;
  let P be FO-pred_symbol of k,A;
  let l be CFO-variable_list of k,A;
  redefine func P!l -> Element of CFO-WFF(A);
end;

theorem :: CFO_LANG:8
  for p being Element of FO-WFF(A) holds
  'not' p is Element of CFO-WFF(A) iff p is Element of CFO-WFF(A);

theorem :: CFO_LANG:9
  for p,q being Element of FO-WFF(A) holds
  p '&' q is Element of CFO-WFF(A) iff p is Element of CFO-WFF(A) & q is
  Element of CFO-WFF(A);

definition
  let A;
  redefine func VERUM(A) -> Element of CFO-WFF(A);
  let r be Element of CFO-WFF(A);
  redefine func 'not' r -> Element of CFO-WFF(A);
  let s be Element of CFO-WFF(A);
  redefine func r '&' s -> Element of CFO-WFF(A);
end;

theorem :: CFO_LANG:10
  for r,s being Element of CFO-WFF(A) holds
  r => s is Element of CFO-WFF(A);

theorem :: CFO_LANG:11
  for r,s being Element of CFO-WFF(A) holds
  r 'or' s is Element of CFO-WFF(A);

theorem :: CFO_LANG:12
  for r,s being Element of CFO-WFF(A) holds
  r <=> s is Element of CFO-WFF(A);

definition
  let A;
  let r,s be Element of CFO-WFF(A);
  redefine func r => s -> Element of CFO-WFF(A);
  redefine func r 'or' s -> Element of CFO-WFF(A);
  redefine func r <=> s -> Element of CFO-WFF(A);
end;

theorem :: CFO_LANG:13
  for x being bound_FO-variable of A, p being Element of FO-WFF(A) holds
  All(x,p) is Element of CFO-WFF(A) iff p is Element of CFO-WFF(A);

definition
  let A;
  let x be bound_FO-variable of A,r be Element of CFO-WFF(A);
  redefine func All(x,r) -> Element of CFO-WFF(A);
end;

theorem :: CFO_LANG:14
  for x being bound_FO-variable of A,r being Element of CFO-WFF(A) holds
  Ex(x,r) is Element of CFO-WFF(A);

definition
  let A;
  let x be bound_FO-variable of A,r be Element of CFO-WFF(A);
  redefine func Ex(x,r) -> Element of CFO-WFF(A);
end;

scheme :: CFO_LANG:sch 1

  CFOInd { A() -> FO-alphabet, P[set] }
:
for r being Element of CFO-WFF(A()) holds P[r]
provided
```

```
 for r,s being Element of CFO-WFF(A())
for x being bound_FO-variable of A() for k
for l being CFO-variable_list of k, A() for P being
FO-pred_symbol of k,A() holds P[VERUM(A())] & P[P!l] &
 (P[r] implies P['not' r]) & (P[r]
& P[s] implies P[r '&' s]) & (P[r] implies P[All(x, r)]);

scheme :: CFO_LANG:sch 2

  CFOFuncEx { Al() -> FO-alphabet, D() -> non empty set,
V() -> (Element of D()), A(set,set,set) -> (Element of D()),
N(set) -> (Element of D()), C(set,set) -> (Element of D()),
Q(set,set) -> Element of D()} : ex F being Function of CFO-WFF(Al()),
D() st F.VERUM(Al()) = V() &
for r,s being Element of CFO-WFF(Al())
for x being bound_FO-variable of Al()
for k
for l being CFO-variable_list of k, Al()
for P being FO-pred_symbol of k,Al() holds F.(P!l) = A(k,P,l) &
F.('not' r) = N(F.r) & F.(r '&' s) = C(F.r,F.s)& F.All(x,r) = Q(x,F.r);

scheme :: CFO_LANG:sch 3

  CFOFuncUniq { Al() -> FO-alphabet, D() -> non empty set,
  F1() -> (Function of CFO-WFF(Al()), D()),
  F2()-> (Function of CFO-WFF(Al()), D()),
  V() -> (Element of D()), A(set,set,set) -> (Element of D()),
  N(set) -> (Element of D()), C(set,set) -> (Element of D()),
  Q(set,set) -> Element of D()} : F1() = F2()
provided
 F1().VERUM(Al()) = V() &
  for r,s being Element of CFO-WFF(Al())
  for x being bound_FO-variable of Al()
  for k
for l being CFO-variable_list of k, Al()
for P being FO-pred_symbol of k,Al() holds F1().(P!l) = A(k,P,l) &
F1().('not' r) = N(F1().r) & F1().(r '&' s) = C(F1().r,F1().s) &
F1().All(x,r) = Q(x,F1().r) and
 F2().VERUM(Al()) = V() &
  for r,s being Element of CFO-WFF(Al())
  for x being bound_FO-variable of Al()
  for k
for l being CFO-variable_list of k, Al()
for P being FO-pred_symbol of k,Al() holds F2().(P!l) =
A(k,P,l) & F2().('not' r) =
N(F2().r) & F2().(r '&' s) = C(F2().r,F2().s) & F2().All(x,r) = Q(x,F2().r);

scheme :: CFO_LANG:sch 4

  CFODefcorrectness { Al() -> FO-alphabet, D() -> non empty set,
    p() -> (Element of CFO-WFF(Al())), V() -> (Element of D()),
    A(set,set,set) -> (Element of D()), N(set) -> (Element of D()),
    C(set,set) -> (Element of D()), Q(set,set) -> Element of D()}
:
  (ex d being Element of D() st ex F being Function of CFO-WFF(Al()),
   D() st d = F.p() & F.VERUM(Al()) = V() &
  for r,s being Element of CFO-WFF(Al())
  for x being bound_FO-variable of Al()
  for k
   for l being CFO-variable_list of k, Al() for P being
   FO-pred_symbol of k,Al() holds F.(P!l) = A(k,P,l) & F.('not' r) = N(F.r) &
   F.(r '&' s) = C(F.r,F.s) & F.All(x,r) = Q(x,F.r) ) &
   for d1,d2 being Element of D() st (ex F being Function of CFO-WFF(Al()),
   D() st d1 = F.p() & F.VERUM(Al()) = V() &
  for r,s being Element of CFO-WFF(Al())
  for x being bound_FO-variable of Al()
  for k
   for l being CFO-variable_list of k, Al()
   for P being FO-pred_symbol of k,Al()
   holds F.(P!l) = A(k,P,l) & F.('not' r) = N(F.r) &
   F.(r '&' s) = C(F.r,F.s) & F.All(x,r) = Q(x,F.r) ) &
   (ex F being Function of CFO-WFF(Al()), D() st d2 = F.p() &
   F.VERUM(Al()) = V() &
  for r,s being Element of CFO-WFF(Al())
  for x being bound_FO-variable of Al()
  for k
for l being CFO-variable_list of k, Al()
   for P being FO-pred_symbol of k,Al() holds F.(P!l) = A(k,P,l) &
   F.('not' r) = N(F.r) & F.(r '&' s) = C(F.r,F.s) &
   F.All(x,r) = Q(x,F.r) ) holds d1 = d2;

scheme :: CFO_LANG:sch 5

  CFODefVERUM { Al() -> FO-alphabet, D() -> non empty set,
F(set) -> (Element of D()), V() -> (Element of D()),
A(set,set,set) -> (Element of D()), N(set) -> (Element of D()),
C(set,set) -> (Element of D()), Q(set,set) -> Element of D()} :
F(VERUM(Al())) = V()
provided
 for p being (Element of CFO-WFF(Al())), d being Element of D() holds
d = F (p) iff ex F being Function of CFO-WFF(Al()), D() st
d = F.p & F.VERUM(Al()) = V() &
  for r,s being Element of CFO-WFF(Al())
  for x being bound_FO-variable of Al()
  for k
for l being CFO-variable_list of k, Al()
for P being FO-pred_symbol of k,Al() holds F.(P!l) = A(k,P,l) &
F.('not' r) = N(F.r) & F.(r '&' s) = C(F.r,F.s) & F.All(x,r) = Q(x,F.r);

scheme :: CFO_LANG:sch 6

  CFODefatomic { Al() -> FO-alphabet, D() -> non empty set,
V() -> (Element of D()), F(set) -> (Element of D()),
A(set,set,set) -> (Element of D()), k() -> Element of NAT, P()
  -> (FO-pred_symbol of k(),Al()), l() -> (CFO-variable_list of k(), Al()),
  N(set) -> (Element of D()), C(set,set) -> (Element of D()),
  Q(set,set) -> Element of D()}
  : F(P()!l()) = A(k(),P(),l())
provided
 for p being (Element of CFO-WFF(Al())), d being Element of D() holds d = F
(p) iff ex F being Function of CFO-WFF(Al()), D()
st d = F.p & F.VERUM(Al()) = V() &
  for r,s being Element of CFO-WFF(Al())
  for x being bound_FO-variable of Al()
  for k
for l being CFO-variable_list of k, Al()
for P being FO-pred_symbol of k,Al() holds
F.(P!l) = A(k,P,l) & F.('not' r) = N(F.r) & F.(r '&' s) = C(F.r,F.s) &
F.All(x,r) = Q(x,F.r);

scheme :: CFO_LANG:sch 7

  CFODefnegative { Al() -> FO-alphabet, D() -> non empty set,
F(set) -> (Element of D()), V() -> (Element of D()),
A(set,set,set) -> (Element of D()), N(set) -> (Element of D()),
r() -> (Element of CFO-WFF(Al())), C(set,set) -> (Element of D()),
Q(set,set) -> Element of D()}
:
F('not' r()) = N(F(r()))
provided
 for p being (Element of CFO-WFF(Al())), d being Element of D() holds
d = F (p) iff ex F being Function of CFO-WFF(Al()), D() st
d = F.p & F.VERUM(Al()) = V() &
  for r,s being Element of CFO-WFF(Al())
  for x being bound_FO-variable of Al()
  for k
for l being CFO-variable_list of k, Al()
for P being FO-pred_symbol of k,Al() holds
F.(P!l) = A(k,P,l) & F.('not' r) = N(F.r) & F.(r '&' s) = C(F.r,F.s) &
F.All(x,r) = Q(x,F.r);

scheme :: CFO_LANG:sch 8

  FODefconjunctive { Al() -> FO-alphabet, D() -> non empty set,
F(set) -> (Element of D()), V() -> (Element of D()),
A(set,set,set) -> (Element of D()), N(set) -> (Element of D()),
C(set,set) -> (Element of D()), r() -> (Element of CFO-WFF(Al())),
s() -> (Element of CFO-WFF(Al())), Q(set,set) -> Element of D()}
:
F(r() '&' s()) = C(F(r()), F(s()))
provided
 for p being (Element of CFO-WFF(Al())), d being Element of D() holds d = F
(p) iff ex F being Function of CFO-WFF(Al()), D() st d = F.p &
F.VERUM(Al()) = V() &
  for r,s being Element of CFO-WFF(Al())
  for x being bound_FO-variable of Al()
  for k
for l being CFO-variable_list of k, Al()
for P being FO-pred_symbol of k,Al() holds
F.(P!l) = A(k,P,l) & F.('not' r) = N(F.r) & F.(r '&' s) = C(F.r,F.s) &
F.All(x,r) = Q(x,F.r);

scheme :: CFO_LANG:sch 9

  FODefuniversal { Al() -> FO-alphabet, D() -> non empty set,
F(set) -> (Element of D()), V() -> (Element of D()),
A(set,set,set) -> (Element of D()), N(set) -> (Element of D()),
C(set,set) -> (Element of D()), Q(set,set) -> (Element of D()),
x() -> bound_FO-variable of Al(), r() -> Element of CFO-WFF(Al())}
:
F(All(x(),r())) = Q(x(),F(r()))
provided
 for p being (Element of CFO-WFF(Al())), d being Element of D() holds d = F
(p) iff ex F being Function of CFO-WFF(Al()), D() st
d = F.p & F.VERUM(Al()) = V() &
  for r,s being Element of CFO-WFF(Al())
  for x being bound_FO-variable of Al()
  for k
for l being CFO-variable_list of k, Al()
for P being FO-pred_symbol of k,Al() holds F.(P!l) = A(k,P,l) &
F.('not' r) = N(F.r) & F.(r '&' s) = C(F.r,F.s) & F.All(x,r) = Q(x,F.r);

reserve x,y for bound_FO-variable of A;
reserve a for free_FO-variable of A;
reserve p,q for Element of FO-WFF(A);
reserve l,l1,l2,ll for FinSequence of FO-variables(A);
reserve r,s for Element of CFO-WFF(A);

definition
  let A;
  let p,x;
  func p.x -> Element of FO-WFF(A) means
:: CFO_LANG:def 3

  ex F being Function of FO-WFF(A),
FO-WFF(A) st it = F.p & for q holds F.VERUM(A) = VERUM(A) &
(q is atomic implies F.q =
(the_pred_symbol_of q)!Subst(the_arguments_of q,(A)a.0.-->x)) &
(q is negative implies F.q = 'not' (F.the_argument_of q) ) &
(q is conjunctive implies F.q = (F.the_left_argument_of q) '&'
(F.the_right_argument_of q)) & (q is universal implies F.q =
IFEQ(bound_in q,x,q,All(bound_in q,F.the_scope_of q)));
end;

theorem :: CFO_LANG:15
  VERUM(A).x = VERUM(A);

theorem :: CFO_LANG:16
  p is atomic implies p.x = (the_pred_symbol_of p)!Subst(
  the_arguments_of p,(A)a.0.-->x);

theorem :: CFO_LANG:17
  for P being FO-pred_symbol of k,A for l being FO-variable_list of
  k,A holds (P!l).x = P!Subst(l,(A)a.0.-->x);

theorem :: CFO_LANG:18
  p is negative implies p.x = 'not'((the_argument_of p).x);

theorem :: CFO_LANG:19
  ('not' p).x = 'not'(p.x);

theorem :: CFO_LANG:20
  p is conjunctive implies p.x = ((the_left_argument_of p).x) '&'
  ((the_right_argument_of p).x);

theorem :: CFO_LANG:21
  (p '&' q).x = (p.x) '&' (q.x);

theorem :: CFO_LANG:22
  p is universal & bound_in p = x implies p.x = p;

theorem :: CFO_LANG:23
  p is universal & bound_in p <> x implies p.x = All(bound_in p,(
  the_scope_of p).x);

theorem :: CFO_LANG:24
  (All(x,p)).x = All(x,p);

theorem :: CFO_LANG:25
  x<>y implies (All(x,p)).y = All(x,p.y);

theorem :: CFO_LANG:26
  Free p = {} implies p.x = p;

theorem :: CFO_LANG:27
  r.x = r;

theorem :: CFO_LANG:28
  Fixed(p.x) = Fixed p;

begin :: Addenda
:: from ALTCAT_1
```

```
reserve i,j,k for set;

theorem :: CFO_LANG:29
  (i,j):->k = [i,j].-->k;

theorem :: CFO_LANG:30
  ((i,j):->k).(i,j) = k;

:: from AMI_1, 2006.03.14, A.T.

theorem :: CFO_LANG:31
  for a,b,c being set holds (a,a) --> (b,c) = a .--> c;

:: from SCMPDS_9, 2006.03.26, A.T.

theorem :: CFO_LANG:32
  for f being Function, a,b,c being set st a <> c holds (f +* (a .-->b))
  .c = f.c;

theorem :: CFO_LANG:33
  for f being Function, a,b,c,d being set st a <> b holds (f +* ((a,b)
  -->(c,d))) .a = c & (f +* ((a,b)-->(c,d))) .b = d;
```

9.5 CFO_THE1

```
:: A First-Order Predicate Calculus.
:: Axiomatics, the Consequence Operation and a Concept of Proof
::  by Agata Darmochwa{\l}
::
:: Received May 25, 1990
:: Copyright (c) 1990-2011 Association of Mizar Users
::           (Stowarzyszenie Uzytkownikow Mizara, Bialystok, Poland).
:: This code can be distributed under the GNU General Public Licence
:: version 3.0 or later, or the Creative Commons Attribution-ShareAlike
:: License version 3.0 or later, subject to the binding interpretation
:: detailed in file COPYING.interpretation.
:: See COPYING.GPL and COPYING.CC-BY-SA for the full text of these
:: licenses, or see http://www.gnu.org/licenses/gpl.html and
:: http://creativecommons.org/licenses/by-sa/3.0/.

environ

 vocabularies NUMBERS, SUBSET_1, XXREAL_0, ARYTM_3, XBOOLE_0, TARSKI, FINSET_1,
      CARD_1, MCART_1, ZFMISC_1, CFO_LANG, FO_LANG1, XBOOLEAN, BVFUNC_2,
      FUNCT_1, FINSEQ_1, NAT_1, RELAT_1, ORDINAL4, ARYTM_1, CFO_THE1;
 notations TARSKI, XBOOLE_0, ZFMISC_1, SUBSET_1, ORDINAL1, XCMPLX_0, FUNCT_1,
      NUMBERS, NAT_1, FINSET_1, FINSEQ_1, MCART_1, FO_LANG1, CFO_LANG,
      XXREAL_0;
 constructors XXREAL_0, XREAL_0, NAT_1, CFO_LANG;
 registrations SUBSET_1, RELSET_1, FINSET_1, XXREAL_0, XREAL_0, FINSEQ_1,
      CFO_LANG, ORDINAL1, CARD_1;
 requirements NUMERALS, REAL, BOOLE, SUBSET, ARITHM;
 definitions TARSKI, XBOOLE_0;
 theorems TARSKI, ZFMISC_1, FINSET_1, FINSEQ_1, MCART_1, FUNCT_1, NAT_1,
      XBOOLE_0, XBOOLE_1, XREAL_1, XXREAL_0, ORDINAL1, RELAT_1;
 schemes NAT_1, FRAENKEL, XBOOLE_0;

begin

:: --------- Auxiliary theorems

reserve Al for FO-alphabet;

reserve i,j,n,k,l for Element of NAT;
reserve a for set;

theorem :: CFO_THE1:1
  {k: k <= n + 1} = {i: i <= n} \/ {n + 1};

theorem :: CFO_THE1:2
  for n holds {k: k <= n} is finite;

reserve X,Y,Z for set;

theorem :: CFO_THE1:3
  X is finite & X c= [:Y,Z:] implies
  ex A,B being set st A is finite & A c= Y & B is finite & B c= Z &
  X c= [:A,B:];

theorem :: CFO_THE1:4
  X is finite & Z is finite & X c= [:Y,Z:] implies
  ex A being set st A is finite & A c= Y & X c= [:A,Z:];

:: --------- The axiomatic of a first-order calculus

reserve T,S,X,Y for Subset of CFO-WFF(Al);
reserve p,q,r,t,F,H,G for Element of CFO-WFF(Al);
reserve s for FO-formula of Al;
reserve x,y for bound_FO-variable of Al;

definition
  let Al;
  let T;
  attr T is being_a_theory means
:: CFO_THE1:def 1

  VERUM(Al) in T & for p,q,r,s,x,y holds
  ('not' p => p) => p in T & p => ('not' p => q) in T &
  (p => q) => ('not'(q '&' r) => 'not'(p '&' r)) in T &
  p '&' q => q '&' p in T & (p in T & p => q in T implies q in T) &
  All(x,p) => p in T &
  (p => q in T & not x in still_not-bound_in p implies p => All(x,q) in T) &
  (s.x in CFO-WFF(Al) & s.y in CFO-WFF(Al) & not x in still_not-bound_in s &
s.x in T implies s.y in T);
end;

theorem :: CFO_THE1:5
  T is being_a_theory & S is being_a_theory implies T /\ S is being_a_theory;

:: --------- The consequence operation

definition
  let Al;
  let X;
  func Cn(X) -> Subset of CFO-WFF(Al) means
:: CFO_THE1:def 2

  t in it iff for T st T is being_a_theory & X c= T holds t in T;
end;

theorem :: CFO_THE1:6
  VERUM(Al) in Cn(X);

theorem :: CFO_THE1:7
  ('not' p => p) => p in Cn(X);

theorem :: CFO_THE1:8
  p => ('not' p => q) in Cn(X);

theorem :: CFO_THE1:9
  (p => q) => ('not'(q '&' r) => 'not'(p '&' r)) in Cn(X);

theorem :: CFO_THE1:10
  p '&' q => q '&' p in Cn(X);

theorem :: CFO_THE1:11
  p in Cn(X) & p => q in Cn(X) implies q in Cn(X);

theorem :: CFO_THE1:12
  All(x,p) => p in Cn(X);

theorem :: CFO_THE1:13
  p => q in Cn(X) & not x in still_not-bound_in p implies
  p => All(x,q) in Cn(X);

theorem :: CFO_THE1:14
  s.x in CFO-WFF(Al) & s.y in CFO-WFF(Al) & not x in still_not-bound_in s &
  s.x in Cn(X) implies s.y in Cn(X);

theorem :: CFO_THE1:15
  Cn(X) is being_a_theory;

theorem :: CFO_THE1:16
  T is being_a_theory & X c= T implies Cn(X) c= T;

theorem :: CFO_THE1:17
  X c= Cn(X);

theorem :: CFO_THE1:18
  X c= Y implies Cn(X) c= Cn(Y);

theorem :: CFO_THE1:19
  Cn(Cn(X)) = Cn(X);

theorem :: CFO_THE1:20
```

```
  T is being_a_theory iff Cn(T) = T;

:: ---------- The notion of proof

definition
  func Proof_Step_Kinds -> set equals
:: CFO_THE1:def 3
  {k: k <= 9};
end;

registration
  cluster Proof_Step_Kinds -> non empty;
end;

theorem :: CFO_THE1:21
  0 in Proof_Step_Kinds & 1 in Proof_Step_Kinds & 2 in Proof_Step_Kinds &
  3 in Proof_Step_Kinds & 4 in Proof_Step_Kinds & 5 in Proof_Step_Kinds &
  6 in Proof_Step_Kinds & 7 in Proof_Step_Kinds & 8 in Proof_Step_Kinds &
  9 in Proof_Step_Kinds;

theorem :: CFO_THE1:22
  Proof_Step_Kinds is finite;

reserve f,g for FinSequence of [:CFO-WFF(Al),Proof_Step_Kinds:];

theorem :: CFO_THE1:23
  for n being Nat holds 1 <= n & n <= len f implies
  (f.n)`2 = 0 or (f.n)`2 = 1 or (f.n)`2 = 2 or (f.n)`2 = 3 or (f.n)`2 = 4
  or (f.n)`2 = 5 or (f.n)`2 = 6 or (f.n)`2 = 7 or (f.n)`2 = 8 or (f.n)`2 = 9;

definition
  let Al;
  let PR be (FinSequence of [:CFO-WFF(Al),Proof_Step_Kinds:]),n be Nat,X;
  pred PR,n is_a_correct_step_wrt X means
:: CFO_THE1:def 4

  (PR.n)`1 in X if (PR.n)`2 = 0, (PR.n)`1 = VERUM(Al) if (PR.n)`2 = 1,
  ex p st (PR.n)`1 = ('not' p => p) => p if (PR.n)`2 = 2,
  ex p,q st (PR.n)`1 = p => ('not' p => q) if (PR.n)`2 = 3,
  ex p,q,r st (PR.n)`1 = (p => q) => ('not'(q '&' r) => 'not'(p '&' r))
  if (PR.n)`2 = 4, ex p,q st (PR.n)`1 = p '&' q => q '&' p if (PR.n)`2 = 5,
  ex p,x st (PR.n)`1 = All(x,p) => p if (PR.n)`2 = 6,
  ex i,j,p,q st 1 <= i & i < n & 1 <= j & j < i & p = (PR.j)`1 & q = (PR.n)`1 &
  (PR.i)`1 = p => q if (PR.n)`2 = 7,
  ex i,p,q,x st 1 <= i & i < n & (PR.i)`1 = p => q &
  not x in still_not-bound_in p & (PR.n)`1 = p => All(x,q) if (PR.n)`2 = 8,
  ex i,x,y,s st 1 <= i & i < n & s.x in CFO-WFF(Al) & s.y in CFO-WFF(Al) &
  not x in still_not-bound_in s & s.x = (PR.i)`1 & s.y = (PR.n)`1
  if (PR.n)`2 = 9;
end;

definition
  let Al;
  let X,f;
  pred f is_a_proof_wrt X means
:: CFO_THE1:def 5

  f <> {} & for n st 1 <= n & n <= len f holds f,n is_a_correct_step_wrt X;
end;

theorem :: CFO_THE1:24
  f is_a_proof_wrt X implies rng f <> {};

theorem :: CFO_THE1:25
  f is_a_proof_wrt X implies 1 <= len f;

theorem :: CFO_THE1:26
  f is_a_proof_wrt X implies (f.1)`2 = 0 or (f.1)`2 = 1 or
  (f.1)`2 = 2 or (f.1)`2 = 3 or (f.1)`2 = 4 or (f.1)`2 = 5 or (f.1)`2 = 6;

theorem :: CFO_THE1:27
  1 <= n & n <= len f implies
  (f,n is_a_correct_step_wrt X iff f^g,n is_a_correct_step_wrt X);

theorem :: CFO_THE1:28
  1 <= n & n <= len g & g,n is_a_correct_step_wrt X implies
  (f^g),(n+len f) is_a_correct_step_wrt X;

theorem :: CFO_THE1:29
  f is_a_proof_wrt X & g is_a_proof_wrt X implies f^g is_a_proof_wrt X;

theorem :: CFO_THE1:30
  f is_a_proof_wrt X & X c= Y implies f is_a_proof_wrt Y;
```

```
theorem :: CFO_THE1:31
  f is_a_proof_wrt X & 1 <= l & l <= len f implies (f.l)`1 in Cn(X);

definition
  let Al;
  let f;
  assume
 f <> {};
  func Effect(f) -> Element of CFO-WFF(Al) equals
:: CFO_THE1:def 6

  (f.(len f))`1;
end;

theorem :: CFO_THE1:32
  f is_a_proof_wrt X implies Effect(f) in Cn(X);

theorem :: CFO_THE1:33
  X c= {F: ex f st f is_a_proof_wrt X & Effect(f) = F};

theorem :: CFO_THE1:34
  for X holds Y = {p: ex f st f is_a_proof_wrt X & Effect(f) = p}
  implies Y is being_a_theory;

theorem :: CFO_THE1:35
  for X holds {p: ex f st f is_a_proof_wrt X & Effect(f) = p} = Cn(X);

theorem :: CFO_THE1:36
  p in Cn(X) iff ex f st f is_a_proof_wrt X & Effect(f) = p;

theorem :: CFO_THE1:37
  p in Cn(X) implies ex Y st Y c= X & Y is finite & p in Cn(Y);

:: --------- TAUT(Al) - the set of all tautologies

definition
  let Al;
  func TAUT(Al) -> Subset of CFO-WFF(Al) equals
:: CFO_THE1:def 7
  Cn({}(CFO-WFF(Al)));
end;

theorem :: CFO_THE1:38
  T is being_a_theory implies TAUT(Al) c= T;

theorem :: CFO_THE1:39
  TAUT(Al) c= Cn(X);

theorem :: CFO_THE1:40
  TAUT(Al) is being_a_theory;

theorem :: CFO_THE1:41
  VERUM(Al) in TAUT(Al);

theorem :: CFO_THE1:42
  ('not' p => p) =>p in TAUT(Al);

theorem :: CFO_THE1:43
  p => ('not' p => q) in TAUT(Al);

theorem :: CFO_THE1:44
  (p => q) => ('not'(q '&' r) => 'not' (p '&' r)) in TAUT(Al);

theorem :: CFO_THE1:45
  p '&' q => q '&' p in TAUT(Al);

theorem :: CFO_THE1:46
  p in TAUT(Al) & p => q in TAUT(Al) implies q in TAUT(Al);

theorem :: CFO_THE1:47
  All(x,p) => p in TAUT(Al);

theorem :: CFO_THE1:48
  p => q in TAUT(Al) & not x in still_not-bound_in p implies
  p => All(x,q) in TAUT(Al);

theorem :: CFO_THE1:49
  s.x in CFO-WFF(Al) & s.y in CFO-WFF(Al) & not x in still_not-bound_in s &
  s.x in TAUT(Al) implies s.y in TAUT(Al);

:: --------- Relation of consequence of a set of formulas

definition
  let Al;
  let X,s;
  pred X|-s means
:: CFO_THE1:def 8

  s in Cn(X);
end;

theorem :: CFO_THE1:50
  X |- VERUM(Al);

theorem :: CFO_THE1:51
  X |- ('not' p => p) => p;

theorem :: CFO_THE1:52
  X |- p => ('not' p => q);

theorem :: CFO_THE1:53
  X |- (p => q) => ('not'(q '&' r) => 'not'(p '&' r));

theorem :: CFO_THE1:54
  X |- p '&' q => q '&' p;

theorem :: CFO_THE1:55
  X |- p & X |- p => q implies X |- q;

theorem :: CFO_THE1:56
  X |- All(x,p) => p;

theorem :: CFO_THE1:57
  X |- p => q & not x in still_not-bound_in p implies X |- p => All(x,q);

theorem :: CFO_THE1:58
  s.y in CFO-WFF(Al) & not x in still_not-bound_in s &
X |- s.x implies X |- s.y;

definition
  let Al;
  let s;
  attr s is valid means
:: CFO_THE1:def 9

  {}(CFO-WFF(Al))|-s;
end;

definition
  let Al;
  let s;
  redefine attr s is valid means
:: CFO_THE1:def 10
  s in TAUT(Al);
end;

theorem :: CFO_THE1:59
  p is valid implies X |- p;

theorem :: CFO_THE1:60
  VERUM(Al) is valid;

theorem :: CFO_THE1:61
  ('not' p => p) =>p is valid;

theorem :: CFO_THE1:62
  p => ('not' p => q) is valid;

theorem :: CFO_THE1:63
  (p => q) => ('not'(q '&' r) => 'not'(p '&' r)) is valid;

theorem :: CFO_THE1:64
  p '&' q => q '&' p is valid;

theorem :: CFO_THE1:65
  p is valid & p => q is valid implies q is valid;

theorem :: CFO_THE1:66
  All(x,p) => p is valid;

theorem :: CFO_THE1:67
  p => q is valid & not x in still_not-bound_in p
  implies p => All(x,q) is valid;

theorem :: CFO_THE1:68
  s.y in CFO-WFF(Al) & not x in still_not-bound_in s &
  s.x is valid implies s.y is valid;
::>
```

```
::> 709: Irrelevant item in the "vocabularies" directive
```

9.6 CFO_SIM1

```
:: Similarity of Formulae
::  by Agata Darmochwa{\l} and Andrzej Trybulec
::
:: Received November 22, 1991
:: Copyright (c) 1991-2011 Association of Mizar Users
::           (Stowarzyszenie Uzytkownikow Mizara, Bialystok, Poland).
:: This code can be distributed under the GNU General Public Licence
:: version 3.0 or later, or the Creative Commons Attribution-ShareAlike
:: License version 3.0 or later, subject to the binding interpretation
:: detailed in file COPYING.interpretation.
:: See COPYING.GPL and COPYING.CC-BY-SA for the full text of these
:: licenses, or see http://www.gnu.org/licenses/gpl.html and
:: http://creativecommons.org/licenses/by-sa/3.0/.

environ

 vocabularies NUMBERS, FUNCT_1, RELAT_1, FUNCT_4, FUNCOP_1, TARSKI, XBOOLE_0,
      SUBSET_1, CFO_LANG, FO_LANG1, ZF_LANG, REALSET1, XXREAL_0, FINSEQ_1,
      XBOOLEAN, CLASSES2, BVFUNC_2, NAT_1, ARYTM_3, CARD_1, FUNCT_2, MARGREL1,
      FINSUB_1, ZFMISC_1, FINSET_1, ZF_LANG1, CFO_SIM1;
 notations TARSKI, XBOOLE_0, ZFMISC_1, SUBSET_1, NUMBERS, DOMAIN_1, MCART_1,
      SETFAM_1, RELAT_1, FUNCT_1, FUNCT_2, BINOP_1, PARTFUN1, XXREAL_0,
      FUNCOP_1, FINSEQ_1, FINSET_1, FINSUB_1, NAT_1, SETWISEO, FO_LANG1,
      FO_LANG2, FO_LANG3, CFO_LANG, FUNCT_4, RECDEF_1, SEQ_4;
 constructors SETFAM_1, PARTFUN1, BINOP_1, DOMAIN_1, FUNCT_4, SETWISEO,
      XXREAL_0, NAT_1, RECDEF_1, SEQ_4, FO_LANG3, CFO_LANG, XXREAL_2, RELSET_1;
 registrations XBOOLE_0, SUBSET_1, FUNCT_1, ORDINAL1, FUNCT_2, FINSUB_1,
      XXREAL_0, NAT_1, MEMBERED, FINSEQ_1, FO_LANG1, CFO_LANG, VALUED_0,
      FINSET_1, FUNCOP_1, XXREAL_2, RELSET_1, CARD_1;
 requirements NUMERALS, REAL, BOOLE, SUBSET;
 definitions TARSKI, XBOOLE_0, BINOP_1, FO_LANG3, SUBSET_1, RELAT_1, FUNCOP_1;
 theorems DOMAIN_1, FINSEQ_1, CFO_LANG, FO_LANG1, FO_LANG3,
      FUNCT_1, FUNCT_2, NAT_1, TARSKI, FUNCOP_1, FUNCT_4, FINSEQ_2, FO_LANG2,
      RELAT_1, RELSET_1, FINSEQ_3, XBOOLE_0, XXREAL_0, CARD_1;
 schemes CFO_LANG, FUNCT_2, CLASSES1, NAT_1, FRAENKEL, CARD_2, FO_LANG1,
      BINOP_1;

begin

reserve A for FO-alphabet;

theorem :: CFO_SIM1:1
  for x,y being set, f being Function holds Im(f+*(x .--> y),x) = { y};

theorem :: CFO_SIM1:2
  for K,L being set for x,y being set, f being Function holds (f+*(
  L --> y)).:K c= f.:K \/ {y};

theorem :: CFO_SIM1:3
  for x,y being set, g being Function, A being set holds (g +* (x
  .--> y)).:(A \ {x}) = g.:(A \ {x});

theorem :: CFO_SIM1:4
  for x,y being set for g being Function for A being set st not y
in g.:(A \ {x}) holds (g +* (x .--> y)).:(A \ {x}) = (g +* (x .--> y)).:A \ {y}
;

reserve i,j,k,l,m,n for Element of NAT;
reserve a,b,e for set;
reserve t,u for FO-symbol of A;

reserve p,q,r,s for Element of CFO-WFF(A);
reserve x for Element of bound_FO-variables(A);
reserve ll for CFO-variable_list of k,A;
reserve P for FO-pred_symbol of k,A;

theorem :: CFO_SIM1:5
  p is atomic implies ex k,P,ll st p = P!ll;

theorem :: CFO_SIM1:6
  p is negative implies ex q st p = 'not' q;

theorem :: CFO_SIM1:7
  p is conjunctive implies ex q,r st p = q '&' r;

theorem :: CFO_SIM1:8
  p is universal implies ex x,q st p = All(x,q);
```

```
theorem :: CFO_SIM1:9
  for l being FinSequence holds rng l = { l.i : 1 <= i & i <= len l };

scheme :: CFO_SIM1:sch 1

  FOFuncExN { Al() -> FO-alphabet, D() -> non empty set,
V() -> Element of D(), A(set) -> Element
of D(), N(set,set) -> Element of D(), C(set,set,set) -> Element of D(), Q(set,
set) -> Element of D()} : ex F being Function of FO-WFF(Al()), D() st
F.VERUM(Al()) = V() &
  for p being Element of FO-WFF(Al()) holds (p is atomic implies F.p = A(p)) &
(p is negative implies F.p = N(F.the_argument_of p,p)) &
(p is conjunctive implies F.p = C(F.the_left_argument_of p,
 F.the_right_argument_of p, p)) & (p is universal implies F.p =
Q(F.the_scope_of p, p));

scheme :: CFO_SIM1:sch 2

  CFOF2FuncEx { Al() -> FO-alphabet, D, E() -> non empty set,
V() -> Element of Funcs(D(),E()), A(set,set,set) -> Element of Funcs(D(),
E()), N(set,set) -> Element of Funcs(D(),E()),
C(set,set,set,set) -> Element of Funcs(D(),E()),
Q(set,set,set) -> Element of Funcs(D(),E()) }
:
ex F being Function of CFO-WFF(Al()), Funcs(D(),E()) st
F.VERUM(Al()) = V() & (for k for l being CFO-variable_list of k,Al()
for P being FO-pred_symbol of k,Al() holds F.(P!l) = A(k,P,l)) &
for r,s being Element of CFO-WFF(Al())
for x being Element of bound_FO-variables(Al())
holds F.('not' r) = N(F.r,r) & F.(r '&' s) = C(F.r,F.s,r,s) &
F.All(x,r) = Q(x,F.r,r);

scheme :: CFO_SIM1:sch 3

  CFOF2FUniq { Al() -> FO-alphabet, D, E() -> non empty set, F1,
             F2() -> Function of CFO-WFF(Al()),Funcs(D(),E()),
             V() -> Function of D(),E(), A(set,set,set) -> Function of D(),
             E(), N(set,set) -> Function of D(),E(),
             C(set,set,set,set) -> Function of D(),E(),
             Q(set,set,set) -> Function of D(),E() }
:
F1() = F2()
provided
 F1().VERUM(Al()) = V() and
 for k for ll be CFO-variable_list of k,Al()
    for P be FO-pred_symbol of k,Al()
    holds F1().(P!ll) = A(k,P,ll) and
 for r,s be Element of CFO-WFF(Al())
    for x be Element of bound_FO-variables(Al())
    holds F1().('not' r) = N(F1().r,r) &
     F1().(r '&' s) = C(F1().r,F1().s,r,s) & F1().All(x,r) = Q(x,F1().r,r) and
 F2().VERUM(Al()) = V() and
 for k for ll be CFO-variable_list of k,Al()
    for P be FO-pred_symbol of k,Al()
    holds F2().(P!ll) = A(k,P,ll) and
 for r,s be Element of CFO-WFF(Al())
    for x be Element of bound_FO-variables(Al())
    holds F2().('not' r) = N(F2().r,r) &
     F2().(r '&' s) = C(F2().r,F2().s,r,s) & F2().All(x,r) = Q(x,F2().r,r);

theorem :: CFO_SIM1:10
  p is_subformula_of 'not' p;

theorem :: CFO_SIM1:11
  p is_subformula_of p '&' q & q is_subformula_of p '&' q;

theorem :: CFO_SIM1:12
  p is_subformula_of All(x,p);

theorem :: CFO_SIM1:13
  for l being CFO-variable_list of k,A, i st 1<=i & i<=len l holds l
  .i in bound_FO-variables(A);

definition
  let A;
  let D be non empty set, f be Function of D, CFO-WFF(A);
  func NEGATIVE f -> Element of Funcs(D, CFO-WFF(A)) means
:: CFO_SIM1:def 1
  for a being
  Element of D for p being Element of CFO-WFF(A) st p=f.a holds it.a = 'not' p;
end;

reserve f,h for Element of Funcs(bound_FO-variables(A),bound_FO-variables(A)),
  K,L for Finite_Subset of bound_FO-variables(A);

definition
  let A;
  let f,g be Function of [:NAT,Funcs(bound_FO-variables(A),
                         bound_FO-variables(A)):],
  CFO-WFF(A), n be Element of NAT;
  func CON(f,g,n) -> Element of Funcs([:NAT,Funcs(bound_FO-variables(A),
  bound_FO-variables(A)):], CFO-WFF(A)) means
:: CFO_SIM1:def 2
  for k,h,p,q st p = f.(k,h) & q = g .(k+n,h) holds it.(k,h) = p '&' q;
end;

definition
  let A;
  let k;
  let l be CFO-variable_list of k,A;
  let f be Element of Funcs(bound_FO-variables(A),bound_FO-variables(A));
  redefine func f*l -> CFO-variable_list of k,A;
end;

definition
  let A;
  let k;
  let P be FO-pred_symbol of k,A, l be CFO-variable_list of k,A;
  func ATOMIC(P,l) -> Element of Funcs([:NAT,Funcs(bound_FO-variables(A),
  bound_FO-variables(A)):], CFO-WFF(A)) means
:: CFO_SIM1:def 3
  for n,h holds it.(n,h) = P!(h*l);
end;

definition
  let A;
  let p;
  func QuantNbr(p) -> Element of NAT means
:: CFO_SIM1:def 4

  ex F being Function of
  CFO-WFF(A), NAT st it = F.p & F.VERUM(A) = 0 & for r,s,x,k for l being
CFO-variable_list of k,A for P being FO-pred_symbol of k,A holds F.(P!l) = 0 &
F.('not' r) = F.r & F.(r '&' s) = F.r + F.s & F.All(x,r) = F.r + 1;
end;

definition
  let A;
  let f be Function of CFO-WFF(A), Funcs([:NAT,Funcs(bound_FO-variables(A),
  bound_FO-variables(A)):],CFO-WFF(A)), x be Element of CFO-WFF(A);
  redefine func f.x -> Element of Funcs([:NAT,Funcs(bound_FO-variables(A),
  bound_FO-variables(A)):],CFO-WFF(A));
end;

theorem :: CFO_SIM1:14
  QuantNbr(VERUM(A)) = 0;

theorem :: CFO_SIM1:15
  QuantNbr(P!ll) = 0;

theorem :: CFO_SIM1:16
  QuantNbr('not' p) = QuantNbr(p);

theorem :: CFO_SIM1:17
  QuantNbr(p '&' q) = QuantNbr(p) + QuantNbr(q);

theorem :: CFO_SIM1:18
  QuantNbr(All(x,p)) = QuantNbr(p) + 1;

theorem :: CFO_SIM1:19
  for p being Element of FO-WFF(A) holds still_not-bound_in p is finite;

scheme :: CFO_SIM1:sch 4

  MaxFinDomElem {D()->non empty set, X()->set, P[set,set] }: ex x being
Element of D() st x in X() & for y being Element of D() st y in X() holds P[x,y
  ]
provided
 X() is finite & X() <> {} & X() c= D() and
 for x,y being Element of D() holds P[x,y] or P[y,x] and
 for x,y,z being Element of D() st P[x,y] & P[y,z] holds P[x,z];

definition
  let X be set;
  redefine func id X -> Element of Funcs(X,X);
end;
```

9.7 FO_VALUA

```
:: Interpretation and Satisfiability in the First Order Logic
::  by Edmund Woronowicz
::
:: Received June 1, 1990
:: Copyright (c) 1990-2011 Association of Mizar Users
::           (Stowarzyszenie Uzytkownikow Mizara, Bialystok, Poland).
:: This code can be distributed under the GNU General Public Licence
:: version 3.0 or later, or the Creative Commons Attribution-ShareAlike
:: License version 3.0 or later, subject to the binding interpretation
:: detailed in file COPYING.interpretation.
:: See COPYING.GPL and COPYING.CC-BY-SA for the full text of these
:: licenses, or see http://www.gnu.org/licenses/gpl.html and
:: http://creativecommons.org/licenses/by-sa/3.0/.

environ

 vocabularies SUBSET_1, NUMBERS, XBOOLE_0, FUNCT_2, FO_LANG1, FUNCT_1, RELAT_1,
      TARSKI, MARGREL1, XBOOLEAN, CFO_LANG, ARYTM_3, FINSEQ_1, NAT_1, XXREAL_0,
      ZF_LANG, FUNCOP_1, REALSET1, BVFUNC_2, ZF_MODEL, ZF_LANG1, FO_LANG3,
      CARD_1, CLASSES2, FO_VALUA;
 notations TARSKI, XBOOLE_0, SUBSET_1, NUMBERS, RELAT_1, FUNCT_1, FUNCT_2,
      NAT_1, FINSEQ_1, FO_LANG1, FO_LANG3, CFO_LANG, MARGREL1, XXREAL_0;
 constructors XXREAL_0, MEMBERED, MARGREL1, FO_LANG3, CFO_LANG, RELSET_1;
 registrations XBOOLE_0, FUNCT_1, RELSET_1, MEMBERED, MARGREL1, FO_LANG1,
      CFO_LANG, XXREAL_0, FUNCT_2, CARD_1;
 requirements NUMERALS, SUBSET, BOOLE;
 definitions XBOOLEAN;
 theorems TARSKI, FINSEQ_1, FUNCT_1, FUNCT_2, FUNCOP_1, FO_LANG1, FO_LANG2,
      FO_LANG3, CFO_LANG, MARGREL1, RELSET_1, RELAT_1, FINSEQ_3, XBOOLE_0,
      XBOOLE_1, XBOOLEAN, ORDINAL1, CARD_1;
 schemes FO_LANG1, CFO_LANG, FUNCT_2;

begin

reserve Al for FO-alphabet;

reserve i,j,k for Element of NAT,
  A,D for non empty set;

definition
  let Al;
  let A be set;
  func Valuations_in(Al,A) -> set equals
:: FO_VALUA:def 1
  Funcs(bound_FO-variables(Al), A);
end;

registration
  let Al;
  let A;
  cluster Valuations_in(Al,A) -> non empty functional;
end;

theorem :: FO_VALUA:1
  for x being set st x is Element of Valuations_in(Al,A) holds x is
  Function of bound_FO-variables(Al) ,A;

definition
  let Al;
  let A;
  redefine func Valuations_in(Al,A) ->
       FUNCTION_DOMAIN of bound_FO-variables(Al), A;
end;

reserve f1,f2 for Element of Funcs(Valuations_in(Al,A),BOOLEAN),
  x,x1,y for bound_FO-variable of Al,
  v,v1 for Element of Valuations_in(Al,A);

definition
  let Al;
  let A, x;
  let p be Element of Funcs(Valuations_in(Al,A),BOOLEAN);
  func FOR_ALL(x,p) -> Element of Funcs(Valuations_in(Al,A),BOOLEAN) means
:: FO_VALUA:def 2

for v holds it.v =
   ALL{p.v9 where v9 is Element of Valuations_in(Al,A) :
       for y st x <> y holds v9.y = v.y};
end;

theorem :: FO_VALUA:2
  for p being Element of Funcs(Valuations_in(Al,A),BOOLEAN) holds
```

```
FOR_ALL(x,p).v = FALSE iff ex v1 st p.v1 = FALSE & for y st x <> y holds v1.y =
  v.y;

theorem :: FO_VALUA:3
  for p being Element of Funcs(Valuations_in(Al,A),BOOLEAN) holds
FOR_ALL(x,p).v = TRUE iff for v1 st for y st x <> y holds v1.y = v.y holds p.v1
  = TRUE;

reserve ll for CFO-variable_list of k,Al;

notation
  let Al;
  let A, k, ll, v;
  synonym v*'ll for v*ll;
end;

definition
  let Al;
  let A, k, ll, v;
  redefine func v*'ll -> FinSequence of A means
:: FO_VALUA:def 3

  len it = k & for i be Nat st 1 <= i & i <= k holds it.i = v.(ll.i);
end;

definition
  let Al;
  let A, k, ll;
  let r be Element of relations_on A;
  func ll 'in' r -> Element of Funcs(Valuations_in(Al,A),BOOLEAN) means
:: FO_VALUA:def 4

  for
  v being Element of Valuations_in(Al,A) holds
   (v*'ll in r iff it.v = TRUE) & (not v*'ll in r iff it.v = FALSE);
end;

definition
  let Al;
  let A;
  let F be Function of CFO-WFF(Al),Funcs(Valuations_in(Al,A), BOOLEAN);
  let p be Element of CFO-WFF(Al);
  redefine func F.p -> Element of Funcs(Valuations_in(Al,A), BOOLEAN);
end;

definition
  let Al;
  let D;
  mode interpretation of Al,D -> Function of FO-pred_symbols(Al),
                                             relations_on D
    means
:: FO_VALUA:def 5
    for P being (Element of FO-pred_symbols(Al)),
    r being Element of relations_on
    D st it.P = r holds r = empty_rel(D) or the_arity_of P = the_arity_of r;
end;

reserve p,q,s,t for Element of CFO-WFF(Al),
  J for interpretation of Al,A,
  P for FO-pred_symbol of k,Al,
  r for Element of relations_on A;

definition
  let Al;
  let A, k, J, P;
  redefine func J.P -> Element of relations_on A;
end;

definition
  let Al;
  let A, J, p;
  func Valid(p,J) -> Element of Funcs(Valuations_in(Al,A), BOOLEAN) means
:: FO_VALUA:def 6

ex F being Function of CFO-WFF(Al),Funcs(Valuations_in(Al,A), BOOLEAN)
st it = F.p & F.VERUM(Al) = (Valuations_in(Al,A) --> TRUE) &
for p,q being Element of CFO-WFF(Al), x being bound_FO-variable of Al,
k being Element of NAT, ll being CFO-variable_list of k,Al, P
  being FO-pred_symbol of k,Al holds F.(P!ll) = (ll 'in' (J.P)) & F.('not' p) =
'not'(F.p) & (F.(p '&' q)) = ((F.p) '&' (F.q)) & F.(All(x,p)) = (FOR_ALL(x,F.p)
  );
end;

theorem :: FO_VALUA:4
```

```
  Valid(VERUM(Al),J) = Valuations_in(Al,A) --> TRUE;

theorem :: FO_VALUA:5
  Valid(VERUM(Al),J).v = TRUE;

theorem :: FO_VALUA:6
  Valid(P!ll,J) = ll 'in' (J.P);

theorem :: FO_VALUA:7
  p = P!ll & r = J.P implies (v*'ll in r iff Valid(p,J).v = TRUE);

theorem :: FO_VALUA:8
  p = P!ll & r = J.P implies (not v*'ll in r iff Valid(p,J).v = FALSE);

theorem :: FO_VALUA:9
  Valid('not' p,J) = 'not' Valid(p,J);

theorem :: FO_VALUA:10
  Valid('not' p,J).v = 'not'(Valid(p,J).v);

theorem :: FO_VALUA:11
  Valid(p '&'q,J) = Valid(p,J) '&' Valid(q,J);

theorem :: FO_VALUA:12
  Valid(p '&'q,J).v = (Valid(p,J).v) '&' (Valid(q,J).v);

theorem :: FO_VALUA:13
  Valid(All(x,p),J) = FOR_ALL(x,Valid(p,J));

theorem :: FO_VALUA:14
  Valid(p '&' 'not' p,J).v = FALSE;

theorem :: FO_VALUA:15
  Valid('not'(p '&' 'not' p),J).v = TRUE;

definition
  let Al;
  let A, p, J, v;
  pred J,v |= p means
:: FO_VALUA:def 7

  Valid(p,J).v = TRUE;
end;

theorem :: FO_VALUA:16
  J,v |= P!ll iff (ll 'in' (J.P)).v = TRUE;

theorem :: FO_VALUA:17
  J,v |= 'not' p iff not J,v |= p;

theorem :: FO_VALUA:18
  J,v |= (p '&' q) iff J,v |= p & J,v |= q;

theorem :: FO_VALUA:19
  J,v |= All(x,p) iff FOR_ALL(x,Valid(p,J)).v = TRUE;

theorem :: FO_VALUA:20
  J,v |= All(x,p) iff for v1 st for y st x <> y holds v1.y = v.y
  holds Valid(p,J).v1 = TRUE;

theorem :: FO_VALUA:21
  Valid('not' 'not' p,J) = Valid(p,J);

theorem :: FO_VALUA:22
  Valid(p '&' p,J) = Valid(p,J);

theorem :: FO_VALUA:23
  J,v |= p => q iff Valid(p, J).v = FALSE or Valid(q, J).v = TRUE;

theorem :: FO_VALUA:24
  J,v |= p => q iff (J,v |= p implies J,v |= q);

theorem :: FO_VALUA:25
  for p being Element of Funcs(Valuations_in(Al,A),BOOLEAN) holds
  FOR_ALL(x,p).v = TRUE implies p.v = TRUE;

definition
  let Al;
  let A, J, p;
  pred J |= p means
:: FO_VALUA:def 8

  for v holds J,v |= p;
end;
```

```
reserve u,w,z for Element of BOOLEAN;

reserve w,v2 for Element of Valuations_in(Al,A),
  z for bound_FO-variable of Al;

theorem :: FO_VALUA:26
  for A be non empty set, Y, Z be bound_FO-variable of Al,
   V1, V2 be Element of Valuations_in(Al,A)
  ex v being Element of Valuations_in(Al,A) st
   (for x being bound_FO-variable of Al st x <> Y holds v.x = V1.x) &
   v.Y = V2.Z;

theorem :: FO_VALUA:27
  not x in still_not-bound_in p implies for v,w st for y st x<>y
  holds w.y = v.y holds Valid(p,J).v = Valid(p,J).w;

theorem :: FO_VALUA:28
  J,v |= p & not x in still_not-bound_in p implies for w st for y
  st x<>y holds w.y = v.y holds J,w |= p;

theorem :: FO_VALUA:29
  J,v |= All(x,p) iff for w st for y st x<>y holds w.y = v.y holds J,w |= p;

reserve u,w for Element of Valuations_in(Al,A);
reserve s9 for FO-formula of Al;

theorem :: FO_VALUA:30
  x <> y & p = s9.x & q = s9.y implies for v st v.x = v.y holds
  Valid(p,J).v = Valid(q,J).v;

theorem :: FO_VALUA:31
  x <> y & not x in still_not-bound_in s9 implies not x in
  still_not-bound_in (s9.y);

theorem :: FO_VALUA:32
  J,v |= VERUM(Al);

theorem :: FO_VALUA:33
  J,v |= p '&' q => q '&' p;

theorem :: FO_VALUA:34
  J,v |= ('not' p => p) => p;

theorem :: FO_VALUA:35
  J,v |= p => ('not' p => q);

theorem :: FO_VALUA:36
  J,v |= (p => q) => ('not'(q '&' t) => 'not'(p '&' t));

theorem :: FO_VALUA:37
  J,v |= p & J,v |= (p => q) implies J,v |= q;

theorem :: FO_VALUA:38
  J,v |= All(x,p) => p;

theorem :: FO_VALUA:39
  J |= VERUM(Al);

theorem :: FO_VALUA:40
  J |= p '&' q => q '&' p;

theorem :: FO_VALUA:41
  J |= ('not' p => p) => p;

theorem :: FO_VALUA:42
  J |= p => ('not' p => q);

theorem :: FO_VALUA:43
  J |= (p => q) => ('not'(q '&' t) => 'not'(p '&' t));

theorem :: FO_VALUA:44
  J |= p & J |= (p => q) implies J |= q;

theorem :: FO_VALUA:45
  J |= All(x,p) => p;

theorem :: FO_VALUA:46
  J |= p => q & not x in still_not-bound_in p implies J |= p => All(x,q);

theorem :: FO_VALUA:47
  for s being FO-formula of Al st p = s.x & q = s.y & not x in
  still_not-bound_in s & J |= p holds J |= q;
```

9.8 SUBST1_2

```
:: Substitution in First-Order Formulas: Elementary Properties
::  by Patrick Braselmann and Peter Koepke
::
:: Received September 25, 2004
:: Copyright (c) 2004-2011 Association of Mizar Users
::           (Stowarzyszenie Uzytkownikow Mizara, Bialystok, Poland).
:: This code can be distributed under the GNU General Public Licence
:: version 3.0 or later, or the Creative Commons Attribution-ShareAlike
:: License version 3.0 or later, subject to the binding interpretation
:: detailed in file COPYING.interpretation.
:: See COPYING.GPL and COPYING.CC-BY-SA for the full text of these
:: licenses, or see http://www.gnu.org/licenses/gpl.html and
:: http://creativecommons.org/licenses/by-sa/3.0/.

environ

 vocabularies NUMBERS, SUBSET_1, FO_LANG1, CFO_LANG, FINSEQ_1, PARTFUN1,
      XBOOLE_0, FUNCT_1, RELAT_1, XXREAL_0, NAT_1, TARSKI, FINSET_1, ZFMISC_1,
      ZF_LANG, CLASSES2, CARD_1, BVFUNC_2, ORDINAL4, REALSET1, XBOOLEAN,
      MARGREL1, MCART_1, ARYTM_3, SUBST1_2;
 notations TARSKI, XBOOLE_0, ZFMISC_1, SUBSET_1, RELAT_1, FUNCT_1, CARD_1,
      NUMBERS, FINSEQ_1, NAT_1, FO_LANG1, FO_LANG3, PARTFUN1, SEQ_4, CFO_LANG,
      FINSET_1, RELSET_1, FUNCT_2, DOMAIN_1, MCART_1, XXREAL_0, ORDINAL1,
      CARD_3;
 constructors PARTFUN1, DOMAIN_1, XXREAL_0, NAT_1, SEQ_4, FO_LANG3, CFO_SIM1,
      RELSET_1, ORDINAL1, CARD_3, ORDERS_1;
 registrations XBOOLE_0, SUBSET_1, FUNCT_1, ORDINAL1, RELSET_1, PARTFUN1,
      FINSET_1, MEMBERED, FINSEQ_1, RFINSEQ, FO_LANG1, CFO_LANG, XXREAL_0,
      CARD_1, CARD_3;
 requirements REAL, NUMERALS, SUBSET, BOOLE;
 definitions TARSKI, FUNCT_1, FO_LANG1, FO_LANG3;
 theorems TARSKI, FINSEQ_1, FUNCT_1, MCART_1, CFO_SIM1, XBOOLE_0, CFO_LANG,
      FO_LANG1, ZFMISC_1, RELAT_1, XBOOLE_1, CARD_3, FUNCT_2, PARTFUN1,
      RELSET_1, NAT_1, FO_LANG2, FINSEQ_3, CARD_1, XXREAL_0, ORDINAL1;
 schemes FUNCT_1, FUNCT_2, FO_LANG1, FO_LANG3, NAT_1, XBOOLE_0, FRAENKEL,
      FINSEQ_1, CLASSES1;

begin :: Preliminaries

reserve A for FO-alphabet;
reserve a,b,b1,b2,c,d for set,
  i,j,k,n for Element of NAT,
  x,y,x1,x2 for bound_FO-variable of A,
  P for FO-pred_symbol of k,A,
  ll for CFO-variable_list of k,A,
  l1 ,l2 for FinSequence of FO-variables(A),
  p for FO-formula of A,
  s,t for FO-symbol of A;

definition
  let A;
  func vSUB(A) equals
:: SUBST1_2:def 1
  PFuncs(bound_FO-variables(A),bound_FO-variables(A));
end;

registration
  let A;
  cluster vSUB(A) -> non empty;
end;

definition
  let A;
  mode CFO_Substitution of A is Element of vSUB(A);
end;

registration
  let A;
  cluster vSUB(A) -> functional;
end;

reserve Sub for CFO_Substitution of A;

definition
  let A;
  let Sub;
  func @Sub -> PartFunc of bound_FO-variables(A),bound_FO-variables(A) equals
:: SUBST1_2:def 2
  Sub;
end;

theorem :: SUBST1_2:1
```

```
  a in dom Sub implies Sub.a in bound_FO-variables(A);

definition
  let A;
  let l be FinSequence of FO-variables(A);
  let Sub;
  func CFO_Subst(l,Sub) -> FinSequence of FO-variables(A) means
:: SUBST1_2:def 3

  len it =
len l & for k st 1 <= k & k <= len l holds (l.k in dom Sub implies it.k = Sub.(
  l.k)) & (not l.k in dom Sub implies it.k = l.k);
end;

definition
  let A;
  let l be FinSequence of bound_FO-variables(A);
  func @l -> FinSequence of FO-variables(A) equals
:: SUBST1_2:def 4
  l;
end;

definition
  let A;
  let l be FinSequence of bound_FO-variables(A);
  let Sub;
  func CFO_Subst(l,Sub) -> FinSequence of bound_FO-variables(A) equals
:: SUBST1_2:def 5
  CFO_Subst(
  @l,Sub);
end;

definition
  let A;
  let Sub;
  let X be set;
  redefine func Sub|X -> CFO_Substitution of A;
end;

registration
  let A;
  cluster finite for CFO_Substitution of A;
end;

definition
  let A;
  let x, p, Sub;
  func RestrictSub(x,p,Sub) -> finite CFO_Substitution of A equals
:: SUBST1_2:def 6
  Sub|{y : y in
  still_not-bound_in p & y is Element of dom Sub & y <> x & y <> Sub.y};
end;

definition
  let A;
  let l1;
  func Bound_Vars(l1) -> Subset of bound_FO-variables(A) equals
:: SUBST1_2:def 7
  { l1.k : 1 <= k &
  k <= len l1 & l1.k in bound_FO-variables(A)};
end;

definition
  let A;
  let p;
  func Bound_Vars(p) -> Subset of bound_FO-variables(A) means
:: SUBST1_2:def 8

  ex F being
  Function of FO-WFF(A), bool bound_FO-variables(A)
   st it = F.p & for p being Element
of FO-WFF(A) for d1,d2 being Subset of bound_FO-variables(A)
holds (p = VERUM(A) implies
  F.p = {}(bound_FO-variables(A))) & (p is atomic implies F.p = Bound_Vars(
the_arguments_of p)) & (p is negative & d1 = F.the_argument_of p implies F.p =
  d1) & (p is conjunctive & d1 = F.the_left_argument_of p & d2 = F.
  the_right_argument_of p implies F.p = d1 \/ d2) & (p is universal & d1 = F.
  the_scope_of p implies F.p = (d1 \/ {bound_in p}));
end;

theorem :: SUBST1_2:2
  Bound_Vars(VERUM(A)) = {};

theorem :: SUBST1_2:3
```

```
  for p being FO-formula of A st p is atomic holds Bound_Vars(p) = Bound_Vars
  (the_arguments_of p);

theorem :: SUBST1_2:4
  for p being FO-formula of A st p is negative holds Bound_Vars(p) =
  Bound_Vars(the_argument_of p);

theorem :: SUBST1_2:5
  for p being FO-formula of A st p is conjunctive holds Bound_Vars(p) = (
  Bound_Vars(the_left_argument_of p)) \/ ( Bound_Vars(the_right_argument_of p))
;

theorem :: SUBST1_2:6
  for p being FO-formula of A st p is universal holds Bound_Vars(p) =
  Bound_Vars(the_scope_of p) \/ {bound_in p};

registration
  let A;
  let p;
  cluster Bound_Vars(p) -> finite;
end;

definition
  let A;
  let p;
  func Dom_Bound_Vars(p) -> finite Subset of FO-symbols(A) equals
:: SUBST1_2:def 9
  {s : x.s in Bound_Vars
  (p)};
end;

reserve finSub for finite CFO_Substitution of A;

definition
  let A;
  let finSub;
  func Sub_Var(finSub) -> finite Subset of FO-symbols(A) equals
:: SUBST1_2:def 10
  {s : x.s in rng finSub};
end;

definition
  let A;
  let p, finSub;
  func NSub(p,finSub) -> non empty Subset of FO-symbols(A) equals
:: SUBST1_2:def 11
  NAT\(Dom_Bound_Vars(p)\/ Sub_Var(finSub));
end;

definition
  let A;
  let finSub, p;
  func upVar(finSub,p) -> FO-symbol of A equals
:: SUBST1_2:def 12
  the Element of NSub(p,finSub);
end;

definition
  let A;
  let x, p, finSub;
  assume
 ex Sub st finSub = RestrictSub(x,All(x,p),Sub);
  func ExpandSub(x,p,finSub) -> CFO_Substitution of A equals
:: SUBST1_2:def 13
  finSub \/ {[x,x.upVar(finSub,p)]}
    if x in rng finSub otherwise finSub \/ {[x,x]};
end;

definition
  let A;
  let p, Sub, b;
  pred p,Sub PQSub b means
:: SUBST1_2:def 14

  (p is universal implies b = ExpandSub(
  bound_in p,the_scope_of p, RestrictSub(bound_in p,p,Sub))) & (not p is
  universal implies b = {});
end;

definition
  let A;
  func QSub(A) -> Function means
:: SUBST1_2:def 15
  a in it iff ex p,Sub,b st a = [[p,Sub],b] & p, Sub PQSub b;
```

```
end;

begin :: Definition and Properties of the
:: Formula - Substitution - Construction

reserve e for Element of vSUB(A);

theorem :: SUBST1_2:7
  [:FO-WFF(A),vSUB(A):] is Subset of [:[:NAT, FO-symbols(A):]*,vSUB(A):] &
(for k being Element of NAT, p being (FO-pred_symbol of k,A),
ll being FO-variable_list
  of k,A , e being Element of vSUB(A) holds [<*p*>^ll,e]
in [:FO-WFF(A),vSUB(A):]) & (for e
  being Element of vSUB(A) holds [<*[0, 0]*>,e] in
[:FO-WFF(A),vSUB(A):]) & (for p being
FinSequence of [:NAT,FO-symbols(A):], e being Element of vSUB(A)
st [p,e] in [:FO-WFF(A),vSUB(A):]
holds [<*[1, 0]*>^p,e] in [:FO-WFF(A),vSUB(A):]) &
(for p, q being FinSequence of [:
  NAT, FO-symbols(A):], e being Element of vSUB(A) st
[p,e] in [:FO-WFF(A),vSUB(A):] & [q,e] in [:
  FO-WFF(A),vSUB(A):] holds [<*[2, 0]*>^p^q,e]
in [:FO-WFF(A),vSUB(A):]) & (for x being
bound_FO-variable of A, p being FinSequence of [:NAT, FO-symbols(A):],
e being Element of vSUB(A)
st [p,(QSub(A)).[<*[3, 0]*>^<*x*>^p,e]] in [:FO-WFF(A),vSUB(A):]
holds [<*[3, 0]*>^<*x*>^p,e] in [:FO-WFF(A),vSUB(A):]);

definition
  let A;
  let IT be set;
  attr IT is A-Sub-closed means
:: SUBST1_2:def 16

  IT is Subset of [:[:NAT, FO-symbols(A):]*,vSUB(A):] &
  (for k being Element of NAT, p being (FO-pred_symbol of k,A), ll being
FO-variable_list of k,A, e being Element of vSUB(A) holds
[<*p*>^ll,e] in IT) & (for
e being Element of vSUB(A) holds [<*[0, 0]*>,e] in IT) &
(for p being FinSequence of [:NAT,FO-symbols(A):],
     e being Element of vSUB(A) st
[p,e] in IT holds [<*[1, 0]*>^p,e]
  in IT) & (for p, q being FinSequence of [:NAT, FO-symbols(A):],
  e being Element of vSUB(A)
  st [p,e] in IT & [q,e] in IT holds [<*[2, 0]*>^p^q,e] in IT) & (for x being
bound_FO-variable of A, p being FinSequence of [:NAT, FO-symbols(A):],
e being Element of vSUB(A) st [p,(QSub(A)).[<*[3, 0]*>^<*x*>^p,e]] in IT holds
[<*[3, 0]*>^<*x*>^p,e] in IT);
end;

registration
  let A;
  cluster A-Sub-closed non empty for set;
end;

definition
  let A;
  func FO-Sub-WFF(A) -> non empty set means
:: SUBST1_2:def 17

  it is A-Sub-closed & for D
  being non empty set st D is A-Sub-closed holds it c= D;
end;

reserve S,S9,S1,S2,S19,S29,T1,T2 for Element of FO-Sub-WFF(A);

theorem :: SUBST1_2:8
  ex p,e st S = [p,e];

registration
  let A;
  cluster FO-Sub-WFF(A) -> A-Sub-closed;
end;

definition
  let A;
  let P be FO-pred_symbol of A;
  let l be FinSequence of FO-variables(A);
  let e;
  assume
 the_arity_of P = len l;
  func Sub_P(P,l,e) -> Element of FO-Sub-WFF(A) equals
:: SUBST1_2:def 18
```

```
  [P!l,e];
end;

theorem :: SUBST1_2:9
  for k being Element of NAT, P being FO-pred_symbol of k,A, ll being
  FO-variable_list of k,A holds Sub_P(P,ll,e) = [P!ll,e];

definition
  let A;
  let S;
  attr S is A-Sub_VERUM means
:: SUBST1_2:def 19

  ex e st S = [VERUM(A),e];
end;

definition
  let A;
  let S;
  redefine func S`1 -> Element of FO-WFF(A);
  redefine func S`2 -> Element of vSUB(A);
end;

theorem :: SUBST1_2:10
  S = [S`1,S`2];

definition
  let A;
  let S;
  func Sub_not S -> Element of FO-Sub-WFF(A) equals
:: SUBST1_2:def 20
  ['not' S`1,S`2];
end;

definition
  let A;
  let S, S9;
  assume
 S`2 = (S9)`2;
  func Sub_&(S,S9) -> Element of FO-Sub-WFF(A) equals
:: SUBST1_2:def 21

  [(S`1) '&' ((S9)`1)
  ,S`2];
end;

reserve B for Element of [:FO-Sub-WFF(A),bound_FO-variables(A):];

definition
  let A;
  let B;
  redefine func B`1 -> Element of FO-Sub-WFF(A);
  redefine func B`2 -> Element of bound_FO-variables(A);
end;

definition
  let A;
  let B;
  attr B is quantifiable means
:: SUBST1_2:def 22

  ex e st (B`1)`2 = (QSub(A)).[All((B)`2,(B`1) `1),e];
end;

definition
  let A;
  let B;
  assume
 B is quantifiable;
  mode second_Q_comp of B -> Element of vSUB(A) means
:: SUBST1_2:def 23

    (B`1)`2 = (QSub(A)).[All (B`2,(B`1)`1),it];
end;

reserve SQ for second_Q_comp of B;

definition
  let A;
  let B, SQ;
  assume
 B is quantifiable;
  func Sub_All(B,SQ) -> Element of FO-Sub-WFF(A) equals
:: SUBST1_2:def 24
```

```
  [All(B`2,(B`1)`1)
  ,SQ];
end;

definition
  let A;
  let S, x;
  redefine func [S,x] -> Element of [:FO-Sub-WFF(A),bound_FO-variables(A):];
end;

scheme :: SUBST1_2:sch 1

  SubFOInd { Al() -> FO-alphabet, Pro[Element of FO-Sub-WFF(Al())]}:
for S being Element of FO-Sub-WFF(Al())
  holds Pro[S]
provided
 for k being Element of NAT, P being (FO-pred_symbol of k,Al()), ll being
FO-variable_list of k,Al(), e being Element of vSUB(Al()) holds
Pro[Sub_P(P,ll,e)] and
 for S being Element of FO-Sub-WFF(Al()) st
S is Al()-Sub_VERUM holds Pro[S] and
 for S being Element of FO-Sub-WFF(Al()) st Pro[S] holds Pro[Sub_not S] and
 for S,S9 being Element of FO-Sub-WFF(Al())
st S`2 = (S9)`2 & Pro[S] & Pro[S9] holds Pro[Sub_&(S,S9)] and
 for x being bound_FO-variable of Al(),
S being Element of FO-Sub-WFF(Al()), SQ
being second_Q_comp of [S,x] st [S,x] is quantifiable & Pro[S] holds Pro[
Sub_All([S,x], SQ)];

definition
  let A;
  let S;
  attr S is Sub_atomic means
:: SUBST1_2:def 25

  ex k being Element of NAT, P being
FO-pred_symbol of k,A, ll being FO-variable_list of k,A,
e being Element of vSUB(A) st
  S = Sub_P(P,ll,e);
end;

theorem :: SUBST1_2:11
  S is Sub_atomic implies S`1 is atomic;

registration
  let A;
  let k be Element of NAT;
  let P be (FO-pred_symbol of k,A), ll be FO-variable_list of k,A;
  let e be Element of vSUB(A);
  cluster Sub_P(P,ll,e) -> Sub_atomic;
end;

definition
  let A;
  let S;
  attr S is Sub_negative means
:: SUBST1_2:def 26

  ex S9 st S = Sub_not S9;
  attr S is Sub_conjunctive means
:: SUBST1_2:def 27

  ex S1,S2 st S = Sub_&(S1,S2) & S1`2 = S2`2;
end;

definition
  let A;
  let S;
  attr S is Sub_universal means
:: SUBST1_2:def 28

  ex B,SQ st S = Sub_All(B,SQ) & B is quantifiable;
end;

theorem :: SUBST1_2:12
  for S holds S is A-Sub_VERUM or S is Sub_atomic or S is
  Sub_negative or S is Sub_conjunctive or S is Sub_universal;

definition
  let A;
  let S such that
 S is Sub_atomic;
  func Sub_the_arguments_of S -> FinSequence of FO-variables(A) means
```

```
:: SUBST1_2:def 29

  ex k being Element of NAT, P being (FO-pred_symbol of k,A), ll being
FO-variable_list of k,A, e being Element of vSUB(A)
st it = ll & S = Sub_P(P,ll,e);
end;

definition
  let A;
  let S such that
 S is Sub_negative;
  func Sub_the_argument_of S -> Element of FO-Sub-WFF(A) means
:: SUBST1_2:def 30

  S = Sub_not it;
end;

definition
  let A;
  let S such that
 S is Sub_conjunctive;
  func Sub_the_left_argument_of S -> Element of FO-Sub-WFF(A) means
:: SUBST1_2:def 31

  ex S9 st S = Sub_&(it,S9) & it`2 = (S9)`2;
end;

definition
  let A;
  let S such that
 S is Sub_conjunctive;
  func Sub_the_right_argument_of S -> Element of FO-Sub-WFF(A) means
:: SUBST1_2:def 32

  ex S9 st S = Sub_&(S9,it) & (S9)`2 = it`2;
end;

definition
  let A;
  let S such that
 S is Sub_universal;
  func Sub_the_bound_of S -> bound_FO-variable of A means
:: SUBST1_2:def 33
  ex B,SQ st S = Sub_All(B, SQ) & B`2 = it & B is quantifiable;
end;

definition
  let A;
  let A2 be Element of FO-Sub-WFF(A) such that
 A2 is Sub_universal;
  func Sub_the_scope_of A2 -> Element of FO-Sub-WFF(A) means
:: SUBST1_2:def 34

  ex B,SQ st A2 = Sub_All(B,SQ) & B`1 = it & B is quantifiable;
end;

registration
  let A;
  let S;
  cluster Sub_not S -> Sub_negative;
end;

theorem :: SUBST1_2:13
  S1`2 = S2`2 implies Sub_&(S1,S2) is Sub_conjunctive;

theorem :: SUBST1_2:14
  B is quantifiable implies Sub_All(B,SQ) is Sub_universal;

theorem :: SUBST1_2:15
  Sub_not(S) = Sub_not(S9) implies S = S9;

theorem :: SUBST1_2:16
  Sub_the_argument_of(Sub_not(S)) = S;

theorem :: SUBST1_2:17
  S1`2 = S2`2 & (S19)`2 = (S29)`2 & Sub_&(S1,S2) = Sub_&(S19,S29)
  implies S1 = S19 & S2 = S29;

theorem :: SUBST1_2:18
  S1`2 = S2`2 implies Sub_the_left_argument_of(Sub_&(S1,S2)) = S1;

theorem :: SUBST1_2:19
  S1`2 = S2`2 implies Sub_the_right_argument_of(Sub_&(S1,S2)) = S2;
```

```
theorem :: SUBST1_2:20
  for B1,B2 being Element of [:FO-Sub-WFF(A),bound_FO-variables(A):], SQ1
being second_Q_comp of B1, SQ2 being second_Q_comp of B2 st B1 is quantifiable
  & B2 is quantifiable & Sub_All(B1,SQ1) = Sub_All(B2,SQ2) holds B1 = B2;

theorem :: SUBST1_2:21
  B is quantifiable implies Sub_the_scope_of(Sub_All(B,SQ)) = B`1;

scheme :: SUBST1_2:sch 2

  SubFOInd2 {Al() -> FO-alphabet, Pro[Element of FO-Sub-WFF(Al())]}:
for S being Element of FO-Sub-WFF(Al())
  holds Pro[S]
provided
 for S being Element of FO-Sub-WFF(Al()) holds (S is Sub_atomic implies Pro
[S]) & (S is Al()-Sub_VERUM implies Pro[S]) & (S is Sub_negative & Pro[
Sub_the_argument_of S] implies Pro[S]) & (S is Sub_conjunctive & Pro[
Sub_the_left_argument_of S] & Pro[Sub_the_right_argument_of S] implies Pro[S])
& (S is Sub_universal & Pro[Sub_the_scope_of S] implies Pro[S]);

theorem :: SUBST1_2:22
  S is Sub_negative implies len @((Sub_the_argument_of(S))`1) < len @(S`1);

theorem :: SUBST1_2:23
  S is Sub_conjunctive implies len @((Sub_the_left_argument_of(S))
  `1) < len @(S`1) & len @((Sub_the_right_argument_of(S))`1) < len @(S`1);

theorem :: SUBST1_2:24
  S is Sub_universal implies len@((Sub_the_scope_of(S))`1) < len @ (S`1);

theorem :: SUBST1_2:25
  (S is A-Sub_VERUM implies ((@S`1).1)`1 = 0) & (S is Sub_atomic
  implies ex k being Element of NAT st (@S`1).1 is FO-pred_symbol of k,A)
& (S is
Sub_negative implies ((@S`1).1)`1 = 1) & (S is Sub_conjunctive implies ((@S`1).
  1)`1 = 2) & (S is Sub_universal implies ((@S`1).1)`1 = 3);

theorem :: SUBST1_2:26
  S is Sub_atomic implies ((@S`1).1)`1 <> 0 & ((@S`1).1)`1 <> 1 &
  ((@S`1).1)`1 <> 2 & ((@S`1).1)`1 <> 3;

theorem :: SUBST1_2:27
  not (ex S st S is Sub_atomic Sub_negative or S is Sub_atomic
  Sub_conjunctive or S is Sub_atomic Sub_universal or S is Sub_negative
  Sub_conjunctive or S is Sub_negative Sub_universal or S is Sub_conjunctive
Sub_universal or S is A-Sub_VERUM Sub_atomic or S is A-Sub_VERUM
Sub_negative or S
  is A-Sub_VERUM Sub_conjunctive or S is A-Sub_VERUM Sub_universal );

scheme :: SUBST1_2:sch 3

  SubFuncEx { Al() -> FO-alphabet, D()-> non empty set,
  V() -> (Element of D()), A(Element of FO-Sub-WFF(Al())) -> (Element of D()),
  N(Element of D()) -> (Element of D()),
  C((Element of D()),(Element of D())) -> (Element of D()),
  R(set,Element of FO-Sub-WFF(Al()), Element of D()) -> Element of D()} :
ex F being Function of FO-Sub-WFF(Al()), D()
st for S being Element of FO-Sub-WFF(Al())
for d1,d2 being Element of D() holds (S is Al()-Sub_VERUM implies F.S = V()) &
(S is Sub_atomic implies F.S = A(S)) & (S is Sub_negative &
d1 = F.Sub_the_argument_of S implies F.S = N(d1)) & (S is Sub_conjunctive &
d1 = F.Sub_the_left_argument_of S & d2 = F.Sub_the_right_argument_of S
implies F.S = C(d1, d2)) & (S is Sub_universal & d1 = F.Sub_the_scope_of S
implies F.S = R(Al(),S,d1));

scheme :: SUBST1_2:sch 4

  SubFOFuncUniq { Al() -> FO-alphabet, D() -> non empty set,
      F1() -> (Function of FO-Sub-WFF(Al()), D()),
      F2() -> (Function of FO-Sub-WFF(Al()), D()), V() -> (Element of D()),
      A(set) -> (Element of D()), N(set) -> (Element of D()),
      C(set,set) -> (Element of D()), R(set,set,set) -> Element of D()}
:
F1() = F2()
provided
 for S being Element of FO-Sub-WFF(Al()) for d1,d2 being Element of D()
holds (S is Al()-Sub_VERUM implies F1().S = V()) &
(S is Sub_atomic implies F1().S =
A(S)) & (S is Sub_negative & d1 = F1().Sub_the_argument_of S implies F1().S = N
(d1)) & (S is Sub_conjunctive & d1 = F1().Sub_the_left_argument_of S & d2 = F1(
).Sub_the_right_argument_of S implies F1().S = C(d1, d2)) & (S is Sub_universal
& d1 = F1().Sub_the_scope_of S implies F1().S = R(Al(),S, d1)) and
 for S being Element of FO-Sub-WFF(Al()) for d1,d2 being Element of D()
holds (S is Al()-Sub_VERUM implies F2().S = V()) &
```

```
(S is Sub_atomic implies F2().S =
A(S)) & (S is Sub_negative & d1 = F2().Sub_the_argument_of S implies F2().S = N
(d1)) & (S is Sub_conjunctive & d1 = F2().Sub_the_left_argument_of S & d2 = F2(
).Sub_the_right_argument_of S implies F2().S = C(d1, d2)) & (S is Sub_universal
& d1 = F2().Sub_the_scope_of S implies F2().S = R(Al(),S, d1));

definition
  let A;
  let S;
  func @S -> Element of [:FO-WFF(A),vSUB(A):] equals
:: SUBST1_2:def 35
  S;
end;

reserve Z for Element of [:FO-WFF(A),vSUB(A):];

definition
  let A;
  let Z;
  redefine func Z`1 -> Element of FO-WFF(A);
  redefine func Z`2 -> CFO_Substitution of A;
end;

definition
  let A;
  let Z;
  func S_Bound(Z) -> bound_FO-variable of A equals
:: SUBST1_2:def 36
  x.upVar(RestrictSub(bound_in Z`1
,Z`1,Z`2),(the_scope_of Z`1)) if bound_in(Z`1) in rng(RestrictSub(bound_in Z`1,
  Z`1,Z`2)) otherwise bound_in(Z`1);
end;

definition
  let A;
  let S, p;
  func Quant(S,p) -> Element of FO-WFF(A) equals
:: SUBST1_2:def 37
  All(S_Bound(@S),p);
end;

begin :: Definition and Properties of Substitution
:: (Ebb et al, Chapter III, Definition 8.1/8.2)

definition
  let A;
  let S be Element of FO-Sub-WFF(A);
  func CFO_Sub(S) -> Element of FO-WFF(A) means
:: SUBST1_2:def 38

  ex F being Function of
FO-Sub-WFF(A),FO-WFF(A) st it = F.S & for S9 being Element of
FO-Sub-WFF(A) holds (S9 is
  A-Sub_VERUM implies F.S9 = VERUM(A)) & ( S9 is Sub_atomic implies F.S9 = (
the_pred_symbol_of ((S9)`1))! CFO_Subst(Sub_the_arguments_of S9,(S9)`2)) & (S9
  is Sub_negative implies F.S9 = 'not' (F.(Sub_the_argument_of S9))) & (S9 is
  Sub_conjunctive implies F.S9 = (F.Sub_the_left_argument_of S9) '&' (F.
Sub_the_right_argument_of S9)) & (S9 is Sub_universal implies F.S9 = Quant(S9,F
  .Sub_the_scope_of S9));
end;

theorem :: SUBST1_2:28
  S is Sub_negative implies CFO_Sub(S) = 'not' CFO_Sub( Sub_the_argument_of S);

theorem :: SUBST1_2:29
  CFO_Sub(Sub_not S) = 'not' CFO_Sub(S);

theorem :: SUBST1_2:30
  S is Sub_conjunctive implies CFO_Sub(S) = (CFO_Sub(
  Sub_the_left_argument_of S)) '&' (CFO_Sub(Sub_the_right_argument_of S));

theorem :: SUBST1_2:31
  S1`2 = S2`2 implies CFO_Sub(Sub_&(S1,S2)) = (CFO_Sub(S1)) '&' ( CFO_Sub(S2));

theorem :: SUBST1_2:32
  S is Sub_universal implies CFO_Sub(S) = Quant(S,CFO_Sub( Sub_the_scope_of S))
;

definition
  let A;
  func CFO-Sub-WFF(A) -> Subset of FO-Sub-WFF(A) equals
:: SUBST1_2:def 39
  {S : S`1 is Element of
  CFO-WFF(A)};
```

```
end;

registration
  let A;
  cluster CFO-Sub-WFF(A) -> non empty;
end;

theorem :: SUBST1_2:33
  S is A-Sub_VERUM implies CFO_Sub(S) is Element of CFO-WFF(A);

theorem :: SUBST1_2:34
  for h being FinSequence holds h is CFO-variable_list of k,A iff h
  is FinSequence of bound_FO-variables(A) & len h = k;

theorem :: SUBST1_2:35
  CFO_Sub(Sub_P(P,ll,e)) is Element of CFO-WFF(A);

theorem :: SUBST1_2:36
  CFO_Sub(S) is Element of CFO-WFF(A) implies CFO_Sub(Sub_not S) is
  Element of CFO-WFF(A);

theorem :: SUBST1_2:37
  S1`2 = S2`2 & CFO_Sub(S1) is Element of CFO-WFF(A) & CFO_Sub(S2) is
  Element of CFO-WFF(A) implies CFO_Sub(Sub_&(S1,S2)) is Element of CFO-WFF(A);

reserve xSQ for second_Q_comp of [S,x];

theorem :: SUBST1_2:38
  CFO_Sub(S) is Element of CFO-WFF(A) & [S,x] is quantifiable implies
  CFO_Sub(Sub_All([S,x],xSQ)) is Element of CFO-WFF(A);

reserve S for Element of CFO-Sub-WFF(A);

scheme :: SUBST1_2:sch 5

  SubCFOInd { Al() -> FO-alphabet, Pro[set] } :
for S being Element of CFO-Sub-WFF(Al()) holds Pro[S]
provided
 for S,S9 being Element of CFO-Sub-WFF(Al()),
x being bound_FO-variable of Al(),
SQ being second_Q_comp of [S,x],
k being Element of NAT,
ll being CFO-variable_list of k,Al(),
P being (FO-pred_symbol of k,Al()),
e being Element of vSUB(Al()) holds
Pro[Sub_P(P,ll,e)] &
(S is Al()-Sub_VERUM implies Pro[S]) &
(Pro[S] implies Pro[Sub_not S]) &
(S`2 = (S9)`2 & Pro[S] & Pro[S9] implies Pro[Sub_&(S,S9)]) &
([S,x] is quantifiable
& Pro[S] implies Pro[Sub_All([S,x], SQ)]);

definition
  let A;
  let S;
  redefine func CFO_Sub(S) -> Element of CFO-WFF(A);
end;

theorem :: SUBST1_2:39
  rng @Sub c= bound_FO-variables(A);
```

9.9 SUBLEM_2

```
:: Copyright (c) 2004-2011 Association of Mizar Users
::           (Stowarzyszenie Uzytkownikow Mizara, Bialystok, Poland).
:: This code can be distributed under the GNU General Public Licence
:: version 3.0 or later, or the Creative Commons Attribution-ShareAlike
:: License version 3.0 or later, subject to the binding interpretation
:: detailed in file COPYING.interpretation.
:: See COPYING.GPL and COPYING.CC-BY-SA for the full text of these
:: licenses, or see http://www.gnu.org/licenses/gpl.html and
:: http://creativecommons.org/licenses/by-sa/3.0/.

environ

 vocabularies SUBSET_1, NUMBERS, CFO_LANG, FO_LANG1, XBOOLE_0, FO_VALUA,
      FUNCT_1, FINSEQ_1, SUBST1_2, RELAT_1, TARSKI, FUNCT_4, FUNCOP_1,
      PARTFUN1, FUNCT_2, MCART_1, REALSET1, XBOOLEAN, ZF_MODEL, ORDINAL4,
      ZF_LANG, ARYTM_3, NAT_1, XXREAL_0, ZFMISC_1, BVFUNC_2, CLASSES2,
      ZF_LANG1, SUBLEM_2, CARD_1;
 notations TARSKI, XBOOLE_0, ZFMISC_1, SUBSET_1, FINSEQ_1, FUNCT_1, NAT_1,
      FO_LANG1, FO_LANG3, PARTFUN1, CARD_1, NUMBERS, XXREAL_0, FUNCOP_1,
      CFO_LANG, RELAT_1, FUNCT_4, SEQ_4, FO_VALUA, RELSET_1, FUNCT_2, MARGREL1,
      DOMAIN_1, MCART_1, SUBST1_2;
 constructors DOMAIN_1, FUNCT_4, XXREAL_0, SEQ_4, FO_LANG3, FO_VALUA, SUBST1_2,
      RELSET_1;
 registrations XBOOLE_0, SUBSET_1, RELAT_1, FUNCT_1, ORDINAL1, FUNCOP_1,
      MEMBERED, FO_LANG1, CFO_LANG, SUBST1_2, XXREAL_0, XXREAL_2, CARD_1,
      RELSET_1;
 requirements NUMERALS, SUBSET, BOOLE;
 definitions TARSKI, XBOOLE_0, FUNCOP_1;
 theorems TARSKI, FINSEQ_1, FUNCT_1, MCART_1, FO_VALUA, XBOOLE_0, XBOOLE_1,
      FINSEQ_2, CFO_LANG, FO_LANG1, ZFMISC_1, RELAT_1, FO_LANG3, FUNCOP_1,
      FUNCT_2, RELSET_1, FO_LANG2, SUBST1_2, FUNCT_4, FUNCT_7, ORDINAL1,
      CARD_1;
 schemes CFO_LANG, XBOOLE_0, SUBST1_2;

begin :: Preliminaries

reserve Al for FO-alphabet;

reserve a,b,c,d for set,
  i,k,n for Element of NAT,
  p,q for Element of CFO-WFF(Al),
  x,y,y1 for bound_FO-variable of Al,
  A for non empty set,
  J for interpretation of Al,A,
  v,w for Element of Valuations_in(Al,A),
  f,g for Function,
  P,P9 for FO-pred_symbol of k,Al,
  ll,ll9 for CFO-variable_list of k,Al,
  l1 for FinSequence of FO-variables(Al),
  Sub,Sub9,Sub1 for CFO_Substitution of Al,
  S,S9,S1,S2 for Element of CFO-Sub-WFF(Al),
  s for FO-symbol of Al;

theorem :: SUBLEM_2:1
  for f,g,h,h1,h2 being Function st dom h1 c= dom h & dom h2 c= dom
  h holds f+*g+*h = (f+*h1)+*(g+*h2)+*h;

theorem :: SUBLEM_2:2
  for vS1 being Function st x in dom vS1 holds (vS1|((dom vS1) \ {x
  })) +* (x .--> vS1.x) = vS1;

definition
  let Al;
  let A;
  mode Val_Sub of A,Al is PartFunc of bound_FO-variables(Al),A;
end;

reserve vS,vS1,vS2 for Val_Sub of A,Al;

notation
  let Al;
  let A, v, vS;
  synonym v.vS for v +* vS;
end;

definition
  let Al;
  let A, v, vS;
  redefine func v.vS -> Element of Valuations_in(Al,A);
end;

definition
  let Al;
  let S;
  redefine func S‘1 -> Element of CFO-WFF(Al);
end;

definition
  let Al;
  let S, A, v;
  func Val_S(v,S) -> Val_Sub of A,Al equals
:: SUBLEM_2:def 1
  (@S‘2)*v;
end;

theorem :: SUBLEM_2:3
  S is Al-Sub_VERUM implies CFO_Sub(S) = VERUM(Al);

definition
  let Al;
  let S, A, v, J;
  pred J,v |= S means
:: SUBLEM_2:def 2
```

```
  J,v |= S`1;
end;

theorem :: SUBLEM_2:4
  S is Al-Sub_VERUM implies for v holds (J,v |= CFO_Sub(S) iff J,v.
  Val_S(v,S) |= S);

theorem :: SUBLEM_2:5
  i in dom ll implies ll.i is bound_FO-variable of Al;

theorem :: SUBLEM_2:6
  S is Sub_atomic implies CFO_Sub(S) = (the_pred_symbol_of S`1)!
  CFO_Subst(Sub_the_arguments_of S,S`2);

theorem :: SUBLEM_2:7
  Sub_the_arguments_of Sub_P(P,ll,Sub) = Sub_the_arguments_of Sub_P(P9,
  ll9,Sub9) implies ll = ll9;

definition
  let k, Al, P, ll, Sub;
  redefine func Sub_P(P,ll,Sub) -> Element of CFO-Sub-WFF(Al);
end;

theorem :: SUBLEM_2:8
  CFO_Sub(Sub_P(P,ll,Sub)) = P!CFO_Subst(ll,Sub);

theorem :: SUBLEM_2:9
  P!CFO_Subst(ll,Sub) is Element of CFO-WFF(Al);

theorem :: SUBLEM_2:10
  CFO_Subst(ll,Sub) is CFO-variable_list of k,Al;

registration
  let Al;
  let k, ll, Sub;
  cluster CFO_Subst(ll,Sub) -> bound_FO-variables(Al) -valued k-element;
end;

theorem :: SUBLEM_2:11
  not x in dom S`2 implies (v.Val_S(v,S)).x = v.x;

theorem :: SUBLEM_2:12
  x in dom S`2 implies (v.Val_S(v,S)).x = Val_S(v,S).x;

theorem :: SUBLEM_2:13
  (v.Val_S(v,Sub_P(P,ll,Sub)))*'ll = v*'(CFO_Subst(ll,Sub));

theorem :: SUBLEM_2:14
  Sub_P(P,ll,Sub)`1 = P!ll;

theorem :: SUBLEM_2:15
  for v holds (J,v |= CFO_Sub(Sub_P(P,ll,Sub)) iff J,v.Val_S(v,
  Sub_P(P,ll,Sub)) |= Sub_P(P,ll,Sub));

theorem :: SUBLEM_2:16
  (Sub_not S)`1 = 'not' S`1 & (Sub_not S)`2 = S`2;

definition
  let Al;
  let S;
  redefine func Sub_not S -> Element of CFO-Sub-WFF(Al);
end;

theorem :: SUBLEM_2:17
  not J,v.Val_S(v,S) |= S iff J,v.Val_S(v,S) |= Sub_not S;

theorem :: SUBLEM_2:18
  Val_S(v,S) = Val_S(v,Sub_not S);

theorem :: SUBLEM_2:19
  (for v holds (J,v |= CFO_Sub(S) iff J,v.Val_S(v,S) |= S))
  implies for v holds (J,v |= CFO_Sub(Sub_not S) iff J,v.Val_S(v,Sub_not S) |=
  Sub_not S);

definition
  let Al;
  let S1, S2;
  assume
 S1`2 = S2`2;
  func CFOSub_&(S1,S2) -> Element of CFO-Sub-WFF(Al) equals
:: SUBLEM_2:def 3

  Sub_&(S1,S2);
```

```
end;

theorem :: SUBLEM_2:20
  S1`2 = S2`2 implies CFOSub_&(S1,S2)`1 = (S1`1) '&' (S2`1) &
  CFOSub_&(S1,S2)`2 = S1`2;

theorem :: SUBLEM_2:21
  S1`2 = S2`2 implies CFOSub_&(S1,S2)`2 = S1`2;

theorem :: SUBLEM_2:22
  S1`2 = S2`2 implies Val_S(v,S1) = Val_S(v,CFOSub_&(S1,S2)) & Val_S(v,
  S2) = Val_S(v,CFOSub_&(S1,S2));

theorem :: SUBLEM_2:23
  S1`2 = S2`2 implies CFO_Sub(CFOSub_&(S1,S2)) = (CFO_Sub(S1)) '&'
  (CFO_Sub(S2));

theorem :: SUBLEM_2:24
  S1`2 = S2`2 implies (J,v.Val_S(v,S1) |= S1 & J,v.Val_S(v,S2) |=
  S2 iff J,v.Val_S(v,CFOSub_&(S1,S2)) |= CFOSub_&(S1,S2));

theorem :: SUBLEM_2:25
  S1`2 = S2`2 & (for v holds (J,v |= CFO_Sub(S1) iff J,v.Val_S(v,
  S1) |= S1)) & (for v holds (J,v |= CFO_Sub(S2) iff J,v.Val_S(v,S2) |= S2))
implies for v holds (J,v |= CFO_Sub(CFOSub_&(S1,S2)) iff J,v.Val_S(v,CFOSub_&(
  S1,S2)) |= CFOSub_&(S1,S2));

reserve B for Element of [:FO-Sub-WFF(Al),bound_FO-variables(Al):],
  SQ for second_Q_comp of B;

theorem :: SUBLEM_2:26
  B is quantifiable implies Sub_All(B,SQ)`1 = All(B`2,(B`1)`1) &
  Sub_All(B,SQ)`2 = SQ;

definition
  let Al;
  let B be Element of [:FO-Sub-WFF(Al),bound_FO-variables(Al):];
  attr B is CFO-WFF-like means
:: SUBLEM_2:def 4

  B`1 in CFO-Sub-WFF(Al);
end;

registration
  let Al;
  cluster CFO-WFF-like for Element of [:FO-Sub-WFF(Al),
   bound_FO-variables(Al):];
end;

definition
  let Al;
  let S, x;
  redefine func [S,x] -> CFO-WFF-like Element of [:FO-Sub-WFF(Al),
  bound_FO-variables(Al):];
end;

reserve B for CFO-WFF-like Element of [:FO-Sub-WFF(Al),
  bound_FO-variables(Al):],
  xSQ for second_Q_comp of [S,x],
  SQ for second_Q_comp of B;

definition
  let Al;
  let B;
  redefine func B`1 -> Element of CFO-Sub-WFF(Al);
end;

definition
  let Al;
  let B, SQ;
  assume
 B is quantifiable;
  func CFOSub_All(B,SQ) -> Element of CFO-Sub-WFF(Al) equals
:: SUBLEM_2:def 5

  Sub_All(B,SQ);
end;

theorem :: SUBLEM_2:27
  B is quantifiable implies CFOSub_All(B,SQ) is Sub_universal;

definition
  let Al;
  let S such that
 S is Sub_universal;
  func CFOSub_the_scope_of S -> Element of CFO-Sub-WFF(Al) equals
:: SUBLEM_2:def 6

  Sub_the_scope_of S;
end;

definition
  let Al;
  let S1, p;
  assume that
 S1 is Sub_universal and
 p = CFO_Sub(CFOSub_the_scope_of S1);
  func CFOQuant(S1,p) -> Element of CFO-WFF(Al) equals
:: SUBLEM_2:def 7

  Quant(S1,p);
end;

theorem :: SUBLEM_2:28
  S is Sub_universal implies CFO_Sub(S) = CFOQuant(S,CFO_Sub(
  CFOSub_the_scope_of S));

theorem :: SUBLEM_2:29
  B is quantifiable implies CFOSub_the_scope_of(CFOSub_All(B,SQ)) = B`1;

begin :: The Substitution Lemma

theorem :: SUBLEM_2:30
  [S,x] is quantifiable implies CFOSub_the_scope_of(CFOSub_All([S,
  x],xSQ)) = S & CFOQuant(CFOSub_All([S,x],xSQ),CFO_Sub(CFOSub_the_scope_of
  CFOSub_All([S,x],xSQ))) = CFOQuant(CFOSub_All([S,x],xSQ),CFO_Sub(S));

theorem :: SUBLEM_2:31
  [S,x] is quantifiable implies CFOQuant(CFOSub_All([S,x],xSQ),
  CFO_Sub(S)) = All(S_Bound(@CFOSub_All([S,x],xSQ)),CFO_Sub(S));

theorem :: SUBLEM_2:32
  x in dom S`2 implies v.((@S`2).x) = v.Val_S(v,S).x;

theorem :: SUBLEM_2:33
  x in dom (@S`2) implies (@S`2).x is bound_FO-variable of Al;

theorem :: SUBLEM_2:34
  [:FO-WFF(Al),vSUB(Al):] c= dom QSub(Al);

reserve B1 for Element of [:FO-Sub-WFF(Al),bound_FO-variables(Al):];
reserve SQ1 for second_Q_comp of B1;

theorem :: SUBLEM_2:35
  B is quantifiable & B1 is quantifiable & Sub_All(B,SQ) = Sub_All
  (B1,SQ1) implies B`2 = B1`2 & SQ = SQ1;

theorem :: SUBLEM_2:36
  B is quantifiable & B1 is quantifiable & CFOSub_All(B,SQ) =
  Sub_All(B1,SQ1) implies B`2 = B1`2 & SQ = SQ1;

theorem :: SUBLEM_2:37
  [S,x] is quantifiable implies Sub_the_bound_of CFOSub_All([S,x],xSQ) = x;

theorem :: SUBLEM_2:38
  [S,x] is quantifiable & x in rng RestrictSub(x,All(x,S`1),xSQ)
implies not S_Bound(@CFOSub_All([S,x],xSQ)) in rng RestrictSub(x,All(x,S`1),xSQ
  ) & not S_Bound(@CFOSub_All([S,x],xSQ)) in Bound_Vars(S`1);

theorem :: SUBLEM_2:39
  [S,x] is quantifiable & not x in rng RestrictSub(x,All(x,S`1),
xSQ) implies not S_Bound(@CFOSub_All([S,x],xSQ)) in rng RestrictSub(x,All(x,S`1
  ),xSQ);

theorem :: SUBLEM_2:40
  [S,x] is quantifiable implies not S_Bound(@CFOSub_All([S,x],xSQ)
  ) in rng RestrictSub(x,All(x,S`1),xSQ);

theorem :: SUBLEM_2:41
  [S,x] is quantifiable implies S`2 = ExpandSub(x,S`1,RestrictSub(
  x,All(x,S`1),xSQ));

theorem :: SUBLEM_2:42
  still_not-bound_in VERUM(Al) c= Bound_Vars(VERUM(Al));

theorem :: SUBLEM_2:43
  still_not-bound_in (P!ll) = Bound_Vars(P!ll);
```

```
theorem :: SUBLEM_2:44
  still_not-bound_in (p) c= Bound_Vars(p) implies
  still_not-bound_in ('not' p) c= Bound_Vars('not' p);

theorem :: SUBLEM_2:45
  still_not-bound_in p c= Bound_Vars(p) & still_not-bound_in q c=
  Bound_Vars(q) implies still_not-bound_in (p '&' q) c= Bound_Vars(p '&' q);

theorem :: SUBLEM_2:46
  still_not-bound_in p c= Bound_Vars(p) implies still_not-bound_in
  All(x,p) c= Bound_Vars(All(x,p));

theorem :: SUBLEM_2:47
  for p holds still_not-bound_in p c= Bound_Vars(p);

notation
  let Al;
  let A;
  let x;
  let a be Element of A;
  synonym x|a for x .--> a;
end;

definition
  let Al;
  let A;
  let x;
  let a be Element of A;
  redefine func x|a -> Val_Sub of A,Al;
end;

reserve a for Element of A;

theorem :: SUBLEM_2:48
  x <> b implies v.(x|a).b = v.b;

theorem :: SUBLEM_2:49
  x = y implies v.(x|a).y = a;

theorem :: SUBLEM_2:50
  J,v |= All(x,p) iff for a holds J,v.(x|a) |= p;

definition
  let Al;
  let S, x, xSQ, A, v;
  func NEx_Val(v,S,x,xSQ) -> Val_Sub of A,Al equals
:: SUBLEM_2:def 8
  (@RestrictSub(x,All(x,S`1),
  xSQ))*v;
end;

definition
  let Al;
  let A;
  let v,w be Val_Sub of A,Al;
  redefine func v+*w -> Val_Sub of A,Al;
end;

theorem :: SUBLEM_2:51
  [S,x] is quantifiable & x in rng RestrictSub(x,All(x,S`1),xSQ)
implies S_Bound(@CFOSub_All([S,x],xSQ)) = x.upVar(RestrictSub(x,All(x,S`1),xSQ)
  ,S`1);

theorem :: SUBLEM_2:52
  [S,x] is quantifiable & not x in rng RestrictSub(x,All(x,S`1),
  xSQ) implies S_Bound(@CFOSub_All([S,x],xSQ)) = x;

theorem :: SUBLEM_2:53
  [S,x] is quantifiable implies for a holds Val_S(v.((S_Bound(@
CFOSub_All([S,x],xSQ)))|a),S) = NEx_Val(v.((S_Bound(@CFOSub_All([S,x],xSQ)))|a)
  ,S,x,xSQ)+*(x|a) & dom RestrictSub(x,All(x,S`1),xSQ) misses {x};

theorem :: SUBLEM_2:54
  [S,x] is quantifiable implies ((for a holds J,(v.((S_Bound(@
  CFOSub_All([S,x],xSQ)))|a)). Val_S(v.((S_Bound(@CFOSub_All([S,x],xSQ)))|a),S)
|= S) iff for a holds J,(v.((S_Bound(@CFOSub_All([S,x],xSQ)))|a)). (NEx_Val(v.(
  (S_Bound(@CFOSub_All([S,x],xSQ)))|a),S,x,xSQ)+*(x|a)) |= S);

theorem :: SUBLEM_2:55
  [S,x] is quantifiable implies for a holds NEx_Val(v.((S_Bound(@
  CFOSub_All([S,x],xSQ)))|a),S,x,xSQ) = NEx_Val(v,S,x,xSQ);

theorem :: SUBLEM_2:56
  [S,x] is quantifiable implies ((for a holds J,(v.((S_Bound(@
CFOSub_All([S,x],xSQ)))|a)). (NEx_Val(v.((S_Bound(@CFOSub_All([S,x],xSQ)))|a),S
,x,xSQ)+*(x|a)) |= S) iff for a holds J,(v.((S_Bound(@CFOSub_All([S,x],xSQ)))|a
  )). (NEx_Val(v,S,x,xSQ)+*(x|a)) |= S );

begin :: The Coincidence Lemma

theorem :: SUBLEM_2:57
  rng ll c= bound_FO-variables(Al) implies still_not-bound_in ll = rng ll;

theorem :: SUBLEM_2:58
  dom v = bound_FO-variables(Al) & dom (x|a) = {x};

theorem :: SUBLEM_2:59
  v*'ll = ll*(v|still_not-bound_in ll);

theorem :: SUBLEM_2:60
  for v,w holds (v|still_not-bound_in (P!ll) = w|
  still_not-bound_in (P!ll) implies (J,v |= P!ll iff J,w |= P!ll));

theorem :: SUBLEM_2:61
  (for v,w holds v|still_not-bound_in p = w|still_not-bound_in p
  implies (J,v |= p iff J,w |= p)) implies for v,w holds v|still_not-bound_in
'not' p = w|still_not-bound_in 'not' p implies (J,v |= 'not' p iff J,w |= 'not'
  p);

theorem :: SUBLEM_2:62
  (for v,w holds v|still_not-bound_in p = w|still_not-bound_in p
  implies (J,v |= p iff J,w |= p)) & (for v,w holds v|still_not-bound_in q = w|
still_not-bound_in q implies (J,v |= q iff J,w |= q)) implies for v,w holds v|
still_not-bound_in p '&' q = w|still_not-bound_in p '&' q implies (J,v |= p '&'
  q iff J,w |= p '&' q);

theorem :: SUBLEM_2:63
  for X being set st X c= bound_FO-variables(Al) holds dom (v|X) = dom
  (v.(x|a)|X) & dom (v|X) = X;

theorem :: SUBLEM_2:64
  v|still_not-bound_in p = w|still_not-bound_in p implies v.(x|a)|
  still_not-bound_in p = w.(x|a)|still_not-bound_in p;

theorem :: SUBLEM_2:65
  still_not-bound_in p c= still_not-bound_in (All(x,p)) \/ {x};

theorem :: SUBLEM_2:66
  v|(still_not-bound_in p \ {x}) = w|(still_not-bound_in p \ {x})
  implies v.(x|a)|still_not-bound_in p = w.(x|a)|still_not-bound_in p;

theorem :: SUBLEM_2:67
  (for v,w holds v|still_not-bound_in p = w|still_not-bound_in p
implies (J,v |= p iff J,w |= p)) implies for v,w holds v|still_not-bound_in All
(x,p) = w|still_not-bound_in All(x,p) implies (J,v |= All(x,p) iff J,w |= All(x
  ,p));

:: Coincidence Lemma (Ebb et al, Chapter III, 5.1)

theorem :: SUBLEM_2:68
  for p holds for v,w holds v|still_not-bound_in p = w|
  still_not-bound_in p implies (J,v |= p iff J,w |= p);

theorem :: SUBLEM_2:69
  [S,x] is quantifiable implies (v.((S_Bound(@CFOSub_All([S,x],xSQ
)))|a)). (NEx_Val(v,S,x,xSQ)+*(x|a))|still_not-bound_in S`1 = (v.(NEx_Val(v,S,x
  ,xSQ)+*(x|a)))|still_not-bound_in S`1;

theorem :: SUBLEM_2:70
  [S,x] is quantifiable implies ((for a holds J,(v.((S_Bound(@
CFOSub_All([S,x],xSQ)))|a)).(NEx_Val(v,S,x,xSQ)+*(x|a)) |= S) iff for a holds J
  ,v.(NEx_Val(v,S,x,xSQ)+*(x|a)) |= S );

theorem :: SUBLEM_2:71
  dom NEx_Val(v,S,x,xSQ) = dom RestrictSub(x,All(x,S`1),xSQ);

theorem :: SUBLEM_2:72
  (for a holds J,v.(NEx_Val(v,S,x,xSQ)+*(x|a)) |= S) iff for a
  holds J,(v.NEx_Val(v,S,x,xSQ)).(x|a) |= S;

theorem :: SUBLEM_2:73
  (for a holds J,(v.NEx_Val(v,S,x,xSQ)).(x|a) |= S) iff for a
  holds J,(v.NEx_Val(v,S,x,xSQ)).(x|a) |= S`1;

theorem :: SUBLEM_2:74
  for v,vS,vS1,vS2 st (for y st y in dom vS1 holds not y in
  still_not-bound_in ll) & (for y st y in dom vS2 holds vS2.y = v.y) & dom vS
  misses dom vS2 holds (v.vS)*'ll = (v.(vS+*vS1+*vS2))*'ll;
```

```
theorem :: SUBLEM_2:75
  for v,vS,vS1,vS2 st (for y st y in dom vS1 holds not y in
still_not-bound_in (P!ll)) & (for y st y in dom vS2 holds vS2.y = v.y) & dom vS
  misses dom vS2 holds J,v.vS |= P!ll iff J,v.(vS+*vS1+*vS2) |= P!ll;

theorem :: SUBLEM_2:76
  (for v,vS,vS1,vS2 st (for y st y in dom vS1 holds not y in
  still_not-bound_in p) & (for y st y in dom vS2 holds vS2.y = v.y) & dom vS
misses dom vS2 holds J,v.vS |= p iff J,v.(vS+*vS1+*vS2) |= p) implies for v,vS,
vS1,vS2 st (for y st y in dom vS1 holds not y in still_not-bound_in 'not' p) &
(for y st y in dom vS2 holds vS2.y = v.y) & dom vS misses dom vS2 holds J,v.vS
  |= 'not' p iff J,v.(vS+*vS1+*vS2) |= 'not' p;

theorem :: SUBLEM_2:77
  (for v,vS,vS1,vS2 st (for y st y in dom vS1 holds not y in
  still_not-bound_in p) & (for y st y in dom vS2 holds vS2.y = v.y) & dom vS
misses dom vS2 holds J,v.vS |= p iff J,v.(vS+*vS1+*vS2) |= p) & (for v,vS,vS1,
vS2 st (for y st y in dom vS1 holds not y in still_not-bound_in q) & (for y st
y in dom vS2 holds vS2.y = v.y) & dom vS misses dom vS2 holds J,v.vS |= q iff J
  ,v.(vS+*vS1+*vS2) |= q) implies for v,vS,vS1,vS2 st (for y st y in dom vS1
holds not y in still_not-bound_in p '&' q) & (for y st y in dom vS2 holds vS2.y
  = v.y) & dom vS misses dom vS2 holds J,v.vS |= p '&' q iff J,v.(vS+*vS1+*vS2)
  |= p '&' q;

theorem :: SUBLEM_2:78
  (for y st y in dom vS1 holds not y in still_not-bound_in All(x,p
  )) implies for y st y in (dom vS1) \ {x} holds not y in still_not-bound_in p;

theorem :: SUBLEM_2:79
  for vS1 being Function holds (for y st y in dom vS1 holds vS1.y
= v.y) & dom vS misses dom vS1 implies for y st y in (dom vS1) \ {x} holds vS1|
  ((dom vS1) \ {x}).y = (v.vS).y;

theorem :: SUBLEM_2:80
  (for v,vS,vS1,vS2 st (for y st y in dom vS1 holds not y in
  still_not-bound_in p) & (for y st y in dom vS2 holds vS2.y = v.y) & dom vS
misses dom vS2 holds J,v.vS |= p iff J,v.(vS+*vS1+*vS2) |= p) implies for v,vS,
vS1,vS2 st (for y st y in dom vS1 holds not y in still_not-bound_in All(x,p)) &
(for y st y in dom vS2 holds vS2.y = v.y) & dom vS misses dom vS2 holds J,v.vS
  |= All(x,p) iff J,v.(vS+*vS1+*vS2) |= All(x,p);

theorem :: SUBLEM_2:81
  for p holds for v,vS,vS1,vS2 st (for y st y in dom vS1 holds not
y in still_not-bound_in p) & (for y st y in dom vS2 holds vS2.y = v.y) & dom vS
  misses dom vS2 holds J,v.vS |= p iff J,v.(vS+*vS1+*vS2) |= p;

definition
  let Al;
  let p;
  func RSub1(p) -> set means
:: SUBLEM_2:def 9

  b in it iff ex x st x = b & not x in still_not-bound_in p;
end;

definition
  let Al;
  let p, Sub;
  func RSub2(p,Sub) -> set means
:: SUBLEM_2:def 10

  b in it iff ex x st x = b & x in still_not-bound_in p & x = (@Sub).x;
end;

theorem :: SUBLEM_2:82
  dom ((@Sub)|RSub1(p)) misses dom ((@Sub)|RSub2(p,Sub));

theorem :: SUBLEM_2:83
  @RestrictSub(x,All(x,p),Sub) = @Sub \ ((@Sub)|RSub1(All(x,p)) +*
  (@Sub)|RSub2(All(x,p),Sub));

theorem :: SUBLEM_2:84
  dom @RestrictSub(x,p,Sub) misses dom ((@Sub)|RSub1(p)) \/ dom ((
  @Sub)|RSub2(p,Sub));

theorem :: SUBLEM_2:85
  [S,x] is quantifiable implies @(CFQSub_All([S,x],xSQ))`2 = @
RestrictSub(x,All(x,S`1),xSQ) +* (@xSQ)|RSub1(All(x,S`1)) +* (@xSQ)|RSub2(All(x
  ,S`1),xSQ);

theorem :: SUBLEM_2:86
  [S,x] is quantifiable implies ex vS1,vS2 st (for y st y in dom
  vS1 holds not y in still_not-bound_in All(x,S`1)) & (for y st y in dom vS2
  holds vS2.y = v.y) & dom NEx_Val(v,S,x,xSQ) misses dom vS2 & v.Val_S(v,
  CFOSub_All([S,x],xSQ)) = v.(NEx_Val(v,S,x,xSQ) +* vS1 +* vS2);

theorem :: SUBLEM_2:87
  [S,x] is quantifiable implies for v holds (J,v.NEx_Val(v,S,x,xSQ
) |= All(x,S`1) iff J,v.Val_S(v,CFOSub_All([S,x],xSQ)) |= CFOSub_All([S,x],xSQ)
  );

theorem :: SUBLEM_2:88
  [S,x] is quantifiable & (for v holds (J,v |= CFO_Sub(S) iff J,v.
  Val_S(v,S) |= S)) implies for v holds (J,v |= CFO_Sub(CFOSub_All([S,x],xSQ))
  iff J,v.Val_S(v,CFOSub_All([S,x],xSQ)) |= CFOSub_All([S,x],xSQ));

scheme :: SUBLEM_2:sch 1

  SubCFOInd1 { Al() -> FO-alphabet, Pro[set] } :
  for S being Element of CFO-Sub-WFF(Al())  holds Pro[S]
provided
 for S,S9 being Element of CFO-Sub-WFF(Al()),
x being bound_FO-variable of Al(),
SQ being second_Q_comp of [S,x],
k being Element of NAT,
ll being CFO-variable_list of k, Al(),
P being (FO-pred_symbol of k,Al()),
e being Element of vSUB(Al())
holds Pro[Sub_P(P,ll,e)] & (S is Al()-Sub_VERUM implies Pro[S]) &
(Pro[S] implies Pro[Sub_not S]) &
(S`2 = (S9)`2 & Pro[S] & Pro[S9] implies Pro[CFOSub_&(S,S9)]) &
([S,x] is quantifiable & Pro[S] implies Pro[CFOSub_All([S,x], SQ)]);

:: Substitution Lemma (Ebb et al, Chapter III, 8.3)

theorem :: SUBLEM_2:89
  for S, v holds (J,v |= CFO_Sub(S) iff J,v.Val_S(v,S) |= S);
```

9.10 SUBST2_2

```
:: Substitution in First-Order Formulas -- Part II. {T}he Construction of
:: First-Order Formulas
::  by Patrick Braselmann and Peter Koepke
::
:: Received September 25, 2004
:: Copyright (c) 2004-2011 Association of Mizar Users
::           (Stowarzyszenie Uzytkownikow Mizara, Bialystok, Poland).
:: This code can be distributed under the GNU General Public Licence
:: version 3.0 or later, or the Creative Commons Attribution-ShareAlike
:: License version 3.0 or later, subject to the binding interpretation
:: detailed in file COPYING.interpretation.
:: See COPYING.GPL and COPYING.CC-BY-SA for the full text of these
:: licenses, or see http://www.gnu.org/licenses/gpl.html and
:: http://creativecommons.org/licenses/by-sa/3.0/.

environ

 vocabularies NUMBERS, SUBSET_1, CFO_LANG, FO_LANG1, SUBST1_2, MCART_1,
      MARGREL1, REALSET1, FINSEQ_1, ORDINAL4, XBOOLEAN, CARD_1, ZFMISC_1,
      RELAT_1, BVFUNC_2, XBOOLE_0, FUNCT_1, TARSKI, ZF_LANG, FUNCT_4, FUNCOP_1,
      CLASSES2, SUBLEM_2, PARTFUN1, CFO_SIM1, ARYTM_3, XXREAL_0, ARYTM_1,
      SUBST2_2;
 notations TARSKI, XBOOLE_0, ZFMISC_1, SUBSET_1, FINSEQ_1, FUNCT_1, FO_LANG1,
      FO_LANG2, FO_LANG3, PARTFUN1, NUMBERS, XCMPLX_0, XXREAL_0, NAT_1,
      CFO_LANG, FUNCOP_1, RELAT_1, FUNCT_4, FUNCT_2, CFO_SIM1, DOMAIN_1,
      MCART_1, SUBST1_2, SUBLEM_2;
 constructors PARTFUN1, DOMAIN_1, XXREAL_0, NAT_1, INT_1, FO_LANG3, CFO_SIM1,
      SUBLEM_2, RELSET_1;
 registrations XBOOLE_0, SUBSET_1, RELAT_1, FUNCT_1, ORDINAL1, FUNCOP_1,
      XXREAL_0, XREAL_0, NAT_1, INT_1, FO_LANG1, CFO_LANG, SUBST1_2, SUBLEM_2,
      CARD_1;
 requirements REAL, NUMERALS, SUBSET, BOOLE, ARITHM;
 definitions TARSKI, FUNCOP_1;
 theorems TARSKI, FUNCT_1, MCART_1, XBOOLE_0, XBOOLE_1, CFO_LANG, FO_LANG1,
      ZFMISC_1, RELAT_1, FO_LANG3, PARTFUN1, RELSET_1, FO_LANG2, SUBST1_2,
      FUNCT_4, SUBLEM_2, CFO_SIM1, FUNCT_2, NAT_1, INT_1, XREAL_1, XXREAL_0,
      FUNCOP_1, CARD_1;
 schemes CFO_LANG, NAT_1;

begin :: Further Properties of Substitution

reserve Al for FO-alphabet;

reserve a,b,b1 for set,
  i,j,k,n for Element of NAT,
  p,q,r,s for Element of CFO-WFF(Al),
  x,y,y1 for bound_FO-variable of Al,
  P for FO-pred_symbol of k,Al,
  l,ll for CFO-variable_list of k,Al,
  Sub,Sub1 for CFO_Substitution of Al,
  S,S1,S2 for Element of CFO-Sub-WFF(Al),
  P1,P2 for Element of FO-pred_symbols(Al);

theorem :: SUBST2_2:1
  for Sub holds ex S st S`1 = VERUM(Al) & S`2 = Sub;

theorem :: SUBST2_2:2
  for Sub holds ex S st S`1 = P!ll & S`2 = Sub;

theorem :: SUBST2_2:3
  for k,l being Element of NAT st P is (FO-pred_symbol of k,Al) & P is (
  FO-pred_symbol of l,Al) holds k = l;

theorem :: SUBST2_2:4
  (for Sub holds ex S st S`1 = p & S`2 = Sub) implies for Sub holds
  ex S st S`1 = 'not' p & S`2 = Sub;

theorem :: SUBST2_2:5
  (for Sub holds ex S st S`1 = p & S`2 = Sub) & (for Sub holds ex S
st S`1 = q & S`2 = Sub) implies for Sub holds ex S st S`1 = p '&' q & S`2 = Sub
;

definition
  let Al;
  let p, Sub;
  redefine func [p,Sub] -> Element of [:FO-WFF(Al),vSUB(Al):];
end;

theorem :: SUBST2_2:6
  dom RestrictSub(x,All(x,p),Sub) misses {x};

theorem :: SUBST2_2:7
  x in rng RestrictSub(x,All(x,p),Sub) implies S_Bound([All(x,p),
  Sub]) = x.upVar(RestrictSub(x,All(x,p),Sub),p);

theorem :: SUBST2_2:8
  not x in rng RestrictSub(x,All(x,p),Sub) implies S_Bound([All(x,p ),Sub]) = x
;

theorem :: SUBST2_2:9
  ExpandSub(x,p,RestrictSub(x,All(x,p),Sub)) = @RestrictSub(x,All(x
  ,p),Sub) +* (x|S_Bound([All(x,p),Sub]));

theorem :: SUBST2_2:10
  S`2 = @RestrictSub(x,All(x,p),Sub) +* (x|S_Bound([All(x,p),Sub])
  ) & S`1 = p implies [S,x] is quantifiable & ex S1 st S1 = [All(x,p),Sub];

theorem :: SUBST2_2:11
  (for Sub holds ex S st S`1 = p & S`2 = Sub) implies for Sub
  holds ex S st S`1 = All(x,p) & S`2 = Sub;

theorem :: SUBST2_2:12
  for p, Sub holds ex S st S`1 = p & S`2 = Sub;

definition
  let Al;
  let p,Sub;
  redefine func [p,Sub] -> Element of CFO-Sub-WFF(Al);
end;

notation
  let Al;
  let x,y;
  synonym Sbst(x,y) for x .--> y;
end;

definition
  let Al;
  let x,y;
  redefine func Sbst(x,y) -> CFO_Substitution of Al;
end;

begin :: Facts about Substitution and Quantifiers of a Formula

definition
  let Al;
  let p,x,y;
  func p.(x,y) -> Element of CFO-WFF(Al) equals
:: SUBST2_2:def 1
```

```
  CFO_Sub([p,Sbst(x,y)]);
end;

scheme :: SUBST2_2:sch 1

  CFOInd1 { Al() -> FO-alphabet, P[set]} :
for p being Element of CFO-WFF(Al()) holds P[p]
provided
 for p being Element of CFO-WFF(Al()) st QuantNbr(p) = 0 holds P[p] and
 for k st for p being Element of CFO-WFF(Al()) st QuantNbr(p) = k
holds P[p] holds for p being Element of CFO-WFF(Al()) st QuantNbr(p) = k+1
holds P[p];

scheme :: SUBST2_2:sch 2

  CFOInd2 {Al() -> FO-alphabet, P[set]}:
   for p being Element of CFO-WFF(Al()) holds P[p]
provided
 for p being Element of CFO-WFF(Al()) st QuantNbr(p) <= 0 holds P[p] and
 for k st for p being Element of CFO-WFF(Al()) st QuantNbr(p) <= k
holds P[p] holds for p being Element of CFO-WFF(Al())
st QuantNbr(p) <= k+1 holds P[p];

theorem :: SUBST2_2:13
  VERUM(Al).(x,y) = VERUM(Al);

theorem :: SUBST2_2:14
  (P!l).(x,y) = P!CFO_Subst(l,Sbst(x,y)) & QuantNbr(P!l) = QuantNbr((P!l
  ).(x,y));

theorem :: SUBST2_2:15
  QuantNbr(P!l) = QuantNbr(CFO_Sub([P!l,Sub]));

definition
  let Al;
  let S be Element of FO-Sub-WFF(Al);
  redefine func S`2 -> CFO_Substitution of Al;
end;

theorem :: SUBST2_2:16
  ['not' p,Sub] = Sub_not [p,Sub];

theorem :: SUBST2_2:17
  'not' p.(x,y) = 'not' (p.(x,y)) & (QuantNbr(p) = QuantNbr(p.(x,y))
  implies QuantNbr('not' p) = QuantNbr('not' p.(x,y)));

theorem :: SUBST2_2:18
  (for Sub holds QuantNbr(p) = QuantNbr(CFO_Sub([p,Sub]))) implies
  for Sub holds QuantNbr('not' p) = QuantNbr(CFO_Sub(['not' p,Sub]));

theorem :: SUBST2_2:19
  [p '&' q,Sub] = CFOSub_&([p,Sub],[q,Sub]);

theorem :: SUBST2_2:20
  (p '&' q).(x,y) = (p.(x,y)) '&' (q.(x,y)) & ( QuantNbr(p) = QuantNbr(p
.(x,y)) & QuantNbr(q) = QuantNbr(q.(x,y)) implies QuantNbr(p '&'q) = QuantNbr((
  p '&' q).(x,y)));

theorem :: SUBST2_2:21
  (for Sub holds QuantNbr(p) = QuantNbr(CFO_Sub([p,Sub]))) & (for
  Sub holds QuantNbr(q) = QuantNbr(CFO_Sub([q,Sub]))) implies for Sub holds
  QuantNbr(p '&' q) = QuantNbr(CFO_Sub[p '&' q,Sub]);

definition
  let Al;
  func CFQ(Al) -> Function of CFO-Sub-WFF(Al),vSUB(Al) equals
:: SUBST2_2:def 2
  (QSub(Al))|CFO-Sub-WFF(Al);
end;

definition
  let Al;
  let p,x,Sub;
  func QScope(p,x,Sub) -> CFO-WFF-like Element of [:FO-Sub-WFF(Al),
  bound_FO-variables(Al):] equals
:: SUBST2_2:def 3
  [[p,(CFQ(Al)).[All(x,p),Sub]],x];
end;

definition
  let Al;
  let p,x,Sub;
  func Qsc(p,x,Sub) -> second_Q_comp of QScope(p,x,Sub) equals
:: SUBST2_2:def 4
```

```
  Sub;
end;

theorem :: SUBST2_2:22
  [All(x,p),Sub] = CFOSub_All(QScope(p,x,Sub),Qsc(p,x,Sub)) &
  QScope(p,x,Sub) is quantifiable;

theorem :: SUBST2_2:23
  (for Sub holds QuantNbr(p) = QuantNbr(CFO_Sub([p,Sub]))) implies
  for Sub holds QuantNbr(All(x,p)) = QuantNbr(CFO_Sub([All(x,p),Sub]));

theorem :: SUBST2_2:24
  QuantNbr(VERUM(Al)) = QuantNbr(CFO_Sub([VERUM(Al),Sub]));

theorem :: SUBST2_2:25
  for p, Sub holds QuantNbr(p) = QuantNbr(CFO_Sub([p,Sub]));

theorem :: SUBST2_2:26
  p is atomic implies ex k,P,ll st p = P!ll;

scheme :: SUBST2_2:sch 3

  CFOInd3 {Al() -> FO-alphabet, P[set]} :
for p being Element of CFO-WFF(Al()) st QuantNbr(p) = 0 holds P[p]
provided
 for r,s being Element of CFO-WFF(Al())
for x being bound_FO-variable of Al()
for k
for l being CFO-variable_list of k,Al()
for P being FO-pred_symbol of k,Al()
holds P[VERUM(Al())] & P[P!l] &
(P[r] implies P['not' r]) & (P[r] & P[s] implies P[r '&' s]);

begin :: Results about the Construction of Formulas

reserve F1,F2,F3 for FO-formula of Al,
  L for FinSequence;

definition
  let Al;
  let G,H be FO-formula of Al;
  assume
 G is_subformula_of H;
  mode PATH of G,H -> FinSequence means
:: SUBST2_2:def 5

    1 <= len it & it.1 = G & it.(
len it) = H & for k st 1 <= k & k < len it ex G1,H1
being Element of FO-WFF(Al) st it.k = G1 & it.(k+1) = H1 &
G1 is_immediate_constituent_of H1;
end;

theorem :: SUBST2_2:27
  for L being PATH of F1,F2 st F1 is_subformula_of F2 & 1 <= i & i <=
  len L holds ex F3 st F3 = L.i & F3 is_subformula_of F2;

theorem :: SUBST2_2:28
  for L being PATH of F1,p st F1 is_subformula_of p & 1 <= i & i
  <= len L holds L.i is Element of CFO-WFF(Al);

theorem :: SUBST2_2:29
  for L being PATH of q,p st QuantNbr(p) <= n & q is_subformula_of
  p & 1 <= i & i <= len L holds ex r st r = L.i & QuantNbr(r) <= n;

theorem :: SUBST2_2:30
  QuantNbr(p) = n & q is_subformula_of p implies QuantNbr(q) <= n;

theorem :: SUBST2_2:31
  for n,p st (for q st q is_subformula_of p holds QuantNbr(q) = n) holds n = 0;

theorem :: SUBST2_2:32
  for p st (for q st q is_subformula_of p holds for x,r holds q <> All(x
  ,r)) holds QuantNbr(p) = 0;

theorem :: SUBST2_2:33
  for p st for q st q is_subformula_of p holds QuantNbr(q) <> 1
  holds QuantNbr(p) = 0;

theorem :: SUBST2_2:34
  1 <= QuantNbr(p) implies ex q st q is_subformula_of p & QuantNbr(q)=1;
```

9.11 CALCL1_2

```
:: A Sequent Calculus for First-Order Logic
::  by Patrick Braselmann and Peter Koepke
::
:: Received September 25, 2004
:: Copyright (c) 2004-2011 Association of Mizar Users
::           (Stowarzyszenie Uzytkownikow Mizara, Bialystok, Poland).
:: This code can be distributed under the GNU General Public Licence
:: version 3.0 or later, or the Creative Commons Attribution-ShareAlike
:: License version 3.0 or later, subject to the binding interpretation
:: detailed in file COPYING.interpretation.
:: See COPYING.GPL and COPYING.CC-BY-SA for the full text of these
:: licenses, or see http://www.gnu.org/licenses/gpl.html and
:: http://creativecommons.org/licenses/by-sa/3.0/.

environ

 vocabularies NUMBERS, SUBSET_1, CFO_LANG, FO_LANG1, XBOOLE_0, FO_VALUA,
      SUBST1_2, FINSEQ_1, RELAT_1, ARYTM_3, XXREAL_0, CARD_1, NAT_1, TARSKI,
      FUNCT_1, ORDINAL4, FINSEQ_2, ZFMISC_1, CFO_THE1, MCART_1, XBOOLEAN,
      BVFUNC_2, ZF_MODEL, SUBST2_2, SUBLEM_2, FUNCOP_1, FUNCT_4, FINSET_1,
      ORDINAL1, CALCL1_2;
 notations TARSKI, XBOOLE_0, ZFMISC_1, SUBSET_1, FINSEQ_1, RELAT_1, FUNCT_1,
      FO_LANG1, CARD_1, NUMBERS, ORDINAL1, FINSEQ_2, XXREAL_0, NAT_1, FUNCOP_1,
      CFO_LANG, FINSET_1, FO_VALUA, CFO_THE1, RELSET_1, PARTFUN1, FUNCT_2,
      DOMAIN_1, MCART_1, SUBST1_2, SUBST2_2, SUBLEM_2;
 constructors PARTFUN1, WELLORD2, DOMAIN_1, XXREAL_0, XREAL_0, NAT_1, FINSEQ_2,
      CFO_THE1, SUBST2_2, RELSET_1;
 registrations XBOOLE_0, SUBSET_1, FUNCT_1, ORDINAL1, RELSET_1, FUNCOP_1,
      FINSET_1, XXREAL_0, XREAL_0, NAT_1, FINSEQ_1, FO_LANG1, CFO_LANG, CARD_1,
      FINSEQ_2, RELAT_1;
 requirements REAL, NUMERALS, SUBSET, BOOLE, ARITHM;
 definitions TARSKI, XBOOLE_0, FINSEQ_2, FUNCOP_1;
 theorems TARSKI, FUNCT_1, MCART_1, XBOOLE_0, XBOOLE_1, FO_LANG1, ZFMISC_1,
      RELAT_1, FO_LANG3, FO_LANG2, SUBST1_2, FUNCT_4, SUBLEM_2, NAT_1,
      FINSEQ_1, FINSEQ_3, FO_VALUA, FINSEQ_2, SUBST2_2, FUNCOP_1, CFO_THE1,
      FINSET_1, CARD_2, CFO_SIM1, CARD_1, CARD_4, GRFUNC_1, XREAL_1, ORDINAL1,
      XXREAL_0;
 schemes XBOOLE_0, NAT_1, CLASSES1;

begin :: Preliminaries

reserve Al for FO-alphabet;

reserve a,b,c,d for set,
  i,j,k,m,n for Element of NAT,
  p,q,r for Element of CFO-WFF(Al),
  x,y,y0 for bound_FO-variable of Al,
  X for Subset of CFO-WFF(Al),
  A for non empty set,
  J for interpretation of Al,A,
  v,w for Element of Valuations_in(Al,A),
  Sub for CFO_Substitution of Al,
  f,f1,g,h,h1 for FinSequence of CFO-WFF(Al);

definition
  let D be non empty set, f be FinSequence of D;
  func Ant(f) -> FinSequence of D means
:: CALCL1_2:def 1

  for i st len f = i+1 holds it = f|(Seg i) if len f > 0 otherwise it = {};
end;

definition
  let Al;
  let f be FinSequence of CFO-WFF(Al);
  func Suc(f) -> Element of CFO-WFF(Al) equals
:: CALCL1_2:def 2

  f.(len f) if len f > 0
  otherwise VERUM(Al);
end;

definition
  let f be Relation, p be set;
  pred p is_tail_of f means
:: CALCL1_2:def 3

  p in rng f;
end;

definition
  let Al;
  let f,g;
  pred f is_Subsequence_of g means
:: CALCL1_2:def 4

  ex N being Subset of NAT st f c= Seq (g|N);
end;

theorem :: CALCL1_2:1
  f is_Subsequence_of g implies rng f c= rng g & ex N being Subset
  of NAT st rng f c= rng (g|N);

theorem :: CALCL1_2:2
  len f > 0 implies len Ant(f)+1 = len f & len Ant(f) < len f;

theorem :: CALCL1_2:3
  len f > 0 implies f = Ant(f)^<*Suc(f)*> & rng f = rng Ant(f) \/ { Suc(f)};

theorem :: CALCL1_2:4
  len f > 1 implies len Ant(f) > 0;

theorem :: CALCL1_2:5
  Suc(f^<*p*>) = p & Ant(f^<*p*>) = f;

reserve fin,fin1 for FinSequence;

theorem :: CALCL1_2:6
  len fin <= len (fin^fin1) & len fin1 <= len (fin^fin1) & (fin <>
  {} implies 1 <= len fin & len fin1 < len (fin1^fin));

theorem :: CALCL1_2:7
  Seq ((f^g)|dom f) = (f^g)|dom f;

theorem :: CALCL1_2:8
  f is_Subsequence_of f^g;

theorem :: CALCL1_2:9
  1 < len (fin^<*b*>^<*c*>);

theorem :: CALCL1_2:10
  1 <= len (fin^<*b*>) & len (fin^<*b*>) in dom (fin^<*b*>);

theorem :: CALCL1_2:11
  0 < m implies len (Sgm (Seg n \/ {n+m})) = n+1;

theorem :: CALCL1_2:12
  0 < m implies dom (Sgm (Seg n \/ {n+m})) = Seg (n+1);

theorem :: CALCL1_2:13
  0 < len f implies f is_Subsequence_of Ant(f)^g^<*Suc(f)*>;

theorem :: CALCL1_2:14
  1 in dom <*c,d*> & 2 in dom <*c,d*> & (f^<*c,d*>).(len f + 1) =
  c & (f^<*c,d*>).(len f + 2) = d;

begin :: A Sequent calculus

definition
  let Al;
  let f;
  func still_not-bound_in f -> Subset of bound_FO-variables(Al) means
:: CALCL1_2:def 5

  a in it iff ex i,p st i in dom f & p = f.i & a in still_not-bound_in p;
end;

definition
  let Al;
  func set_of_CFO-WFF-seq(Al) means
:: CALCL1_2:def 6

  a in it iff a is FinSequence of CFO-WFF(Al);
end;

reserve PR,PR1 for FinSequence of [:set_of_CFO-WFF-seq(Al),Proof_Step_Kinds:];

definition
  let Al;
  let PR;
  let n be Nat;
  pred PR,n is_a_correct_step means
:: CALCL1_2:def 7

  ex f st Suc(f) is_tail_of Ant(f) &
  (PR.n)`1 = f if (PR.n)`2 = 0, ex f st (PR.n)`1 = f^<*VERUM(Al)*>
  if (PR.n)‘2 = 1,
ex i,f,g st 1 <= i & i < n & Ant(f) is_Subsequence_of Ant(g) & Suc(f) = Suc(g)
& (PR.i)‘1 = f & (PR.n)‘1 = g if (PR.n)‘2 = 2, ex i,j,f,g st 1 <= i & i < n & 1
  <= j & j < i & len f > 1 & len g > 1 & Ant(Ant(f)) = Ant(Ant(g)) & ’not’ Suc(
Ant(f)) = Suc(Ant(g)) & Suc(f) = Suc(g) & f = (PR.j)‘1 & g = (PR.i)‘1 & Ant(Ant
(f))^<*Suc(f)*> = (PR.n)‘1 if (PR.n)‘2 = 3, ex i,j,f,g,p st 1 <= i & i < n & 1
<= j & j < i & len f > 1 & Ant(f) = Ant(g) & Suc(Ant(f)) = ’not’ p & ’not’ Suc(
f) = Suc(g) & f = (PR.j)‘1 & g = (PR.i)‘1 & Ant(Ant(f))^<*p*> = (PR.n)‘1 if (PR
.n)‘2 = 4, ex i,j,f,g st 1 <= i & i < n & 1 <= j & j < i & Ant(f) = Ant(g) & f
= (PR.j)‘1 & g = (PR.i)‘1 & Ant(f)^<*(Suc(f)) ’&’ (Suc(g))*> = (PR.n)‘1 if (PR.
n)‘2 = 5, ex i,f,p,q st 1 <= i & i < n & p ’&’ q = Suc(f) & f = (PR.i)‘1 & Ant(
  f)^<*p*> = (PR.n)‘1 if (PR.n)‘2 = 6, ex i,f,p,q st 1 <= i & i < n & p ’&’ q =
Suc(f) & f = (PR.i)‘1 & Ant(f)^<*q*>= (PR.n)‘1 if (PR.n)‘2 = 7, ex i,f,p,x,y st
1 <= i & i < n & Suc(f) = All(x,p) & f = (PR.i)‘1 & Ant(f)^<*p.(x,y)*> = (PR.n)
  ‘1 if (PR.n)‘2 = 8, ex i,f,p,x,y st 1 <= i & i < n & Suc(f) = p.(x,y) & not y
in still_not-bound_in (Ant(f)) & not y in still_not-bound_in All(x,p) & f = (PR
  .i)‘1 & Ant(f)^<*All(x,p)*> = (PR.n)‘1 if (PR.n)‘2 = 9;
end;

definition
  let Al;
  let PR;
  attr PR is a_proof means
:: CALCL1_2:def 8

  PR <> {} & for n being Nat st 1 <= n & n <=
  len PR holds PR,n is_a_correct_step;
end;

definition
  let Al;
  let f;
  pred |- f means
:: CALCL1_2:def 9

  ex PR st PR is a_proof & f = (PR.(len PR))‘1;
end;

definition
  let Al;
  let p,X;
  pred p is_formal_provable_from X means
:: CALCL1_2:def 10

  ex f st rng Ant(f) c= X & Suc (f) = p & |- f;
end;

definition
  let Al;
  let X;
  let A;
  let J;
  let v;
  pred J,v |= X means
:: CALCL1_2:def 11

  p in X implies J,v |= p;
end;

definition
  let Al;
  let X,p;
  pred X |= p means
:: CALCL1_2:def 12
  J,v |= X implies J,v |= p;
end;

definition
  let Al;
  let p;
  pred |= p means
:: CALCL1_2:def 13
  {}(CFO-WFF(Al)) |= p;
end;

definition
  let Al;
  let f, A, J, v;
  pred J,v |= f means
:: CALCL1_2:def 14

  J,v |= rng(f);
end;
```

```
definition
  let Al;
  let f, p;
  pred f |= p means
:: CALCL1_2:def 15

  J,v |= f implies J,v |= p;
end;

theorem :: CALCL1_2:15
  Suc(f) is_tail_of Ant(f) implies Ant(f) |= Suc(f);

theorem :: CALCL1_2:16
  Ant(f) is_Subsequence_of Ant(g) & Suc(f) = Suc(g) & Ant(f) |=
  Suc(f) implies Ant(g) |= Suc(g);

theorem :: CALCL1_2:17
  len f > 0 implies (J,v |= Ant(f) & J,v |= Suc(f) iff J,v |= f);

theorem :: CALCL1_2:18
  len f > 1 & len g > 1 & Ant(Ant(f)) = Ant(Ant(g)) & 'not' Suc(
  Ant(f)) = Suc(Ant(g)) & Suc(f) = Suc(g) & Ant(f) |= Suc(f) & Ant(g) |= Suc(g)
  implies Ant(Ant(f)) |= Suc(f);

theorem :: CALCL1_2:19
  len f > 1 & Ant(f) = Ant(g) & 'not' p = Suc(Ant(f)) & 'not' Suc(
  f) = Suc(g) & Ant(f) |= Suc(f) & Ant(g) |= Suc(g) implies Ant(Ant(f)) |= p;

theorem :: CALCL1_2:20
  Ant(f) = Ant(g) & Ant(f) |= Suc(f) & Ant(g) |= Suc(g) implies
  Ant(f) |= (Suc(f)) '&' (Suc(g));

theorem :: CALCL1_2:21
  Ant(f) |= p '&' q implies Ant(f) |= p;

theorem :: CALCL1_2:22
  Ant(f) |= p '&' q implies Ant(f) |= q;

theorem :: CALCL1_2:23
  J,v |= [p,Sub] iff J,v |= p;

reserve a for Element of A;

theorem :: CALCL1_2:24
  J,v |= p.(x,y) iff ex a st v.y = a & J,v.(x|a) |= p;

theorem :: CALCL1_2:25
  Suc(f) = All(x,p) & Ant(f) |= Suc(f) implies for y holds Ant(f) |= p.(x,y);

theorem :: CALCL1_2:26
  for X being set st X c= bound_FO-variables(Al) holds not x in X
  implies v.(x|a)|X = v|X;

theorem :: CALCL1_2:27
  for v,w holds v|still_not-bound_in f = w|still_not-bound_in f
  implies (J,v |= f implies J,w |= f);

theorem :: CALCL1_2:28
  not y in still_not-bound_in All(x,p) implies v.(y|a).(x|a)|
  still_not-bound_in p = v.(x|a)|still_not-bound_in p;

theorem :: CALCL1_2:29
  Suc(f) = p.(x,y) & Ant(f) |= Suc(f) & not y in
still_not-bound_in Ant(f) & not y in still_not-bound_in All(x,p) implies Ant(f)
  |= All(x,p);

theorem :: CALCL1_2:30
  Ant(f^<*VERUM(Al)*>) |= Suc(f^<*VERUM(Al)*>);

theorem :: CALCL1_2:31
  for n being Nat holds 1 <= n & n <= len PR implies (PR.n)`2 = 0
or (PR.n)`2 = 1 or (PR.n)`2 = 2 or (PR.n)`2 = 3 or (PR.n)`2 = 4 or (PR.n)`2 = 5
  or (PR.n)`2 = 6 or (PR.n)`2 = 7 or (PR.n)`2 = 8 or (PR.n)`2 = 9;

:: Theorem on the Correctness (Ebb et al, Chapter IV, Theorem 6.2)

theorem :: CALCL1_2:32
  p is_formal_provable_from X implies X |= p;

begin :: Derived Rules

theorem :: CALCL1_2:33
  Suc(f) is_tail_of Ant(f) implies |- f;
```

```
theorem :: CALCL1_2:34
  for n being Nat holds 1 <= n & n <= len PR implies (PR,n
  is_a_correct_step iff PR^PR1,n is_a_correct_step);

theorem :: CALCL1_2:35
  1 <= n & n <= len PR1 & PR1,n is_a_correct_step implies (PR^PR1)
  ,(n+len PR) is_a_correct_step;

theorem :: CALCL1_2:36
  Ant(f) is_Subsequence_of Ant(g) & Suc(f) = Suc(g) & |- f implies |- g;

theorem :: CALCL1_2:37
  1 < len f & 1 < len g & Ant(Ant(f)) = Ant(Ant(g)) & 'not' Suc(
Ant(f)) = Suc(Ant(g)) & Suc(f) = Suc(g) & |- f & |- g implies |- Ant(Ant(f))^<*
  Suc(f)*>;

theorem :: CALCL1_2:38
  len f > 1 & Ant(f) = Ant(g) & Suc(Ant(f)) = 'not' p & 'not' Suc(
  f) = Suc(g) & |- f & |- g implies |- Ant(Ant(f))^<*p*>;

theorem :: CALCL1_2:39
  Ant(f) = Ant(g) & |- f & |- g implies |- Ant(f)^<*(Suc(f)) '&' ( Suc(g))*>;

theorem :: CALCL1_2:40
  p '&' q = Suc(f) & |- f implies |- Ant(f)^<*p*>;

theorem :: CALCL1_2:41
  p '&' q = Suc(f) & |- f implies |- Ant(f)^<*q*>;

theorem :: CALCL1_2:42
  Suc(f) = All(x,p) & |- f implies |- Ant(f)^<*p.(x,y)*>;

theorem :: CALCL1_2:43
  Suc(f) = p.(x,y) & not y in still_not-bound_in Ant(f) & not y in
  still_not-bound_in All(x,p) & |- f implies |- Ant(f)^<*All(x,p)*>;

theorem :: CALCL1_2:44
  |- f & |- Ant(f)^<*'not' Suc(f)*> implies |- Ant(f)^<*p*>;

theorem :: CALCL1_2:45
  1 <= len f & |- f & |- f^<*p*> implies |- Ant(f)^<*p*>;

theorem :: CALCL1_2:46
  |- f^<*p*>^<*q*> implies |- f^<*'not' q*>^<*'not' p*>;

theorem :: CALCL1_2:47
  |- f^<*'not' p*>^<*'not' q*> implies |- f^<*q*>^<*p*>;

theorem :: CALCL1_2:48
  |- f^<*'not' p*>^<*q*> implies |- f^<*'not' q*>^<*p*>;

theorem :: CALCL1_2:49
  |- f^<*p*>^<*'not' q*> implies |- f^<*q*>^<*'not' p*>;

theorem :: CALCL1_2:50
  |- f^<*p*>^<*r*> & |- f^<*q*>^<*r*> implies |- f^<*p 'or' q*>^<*r*>;

theorem :: CALCL1_2:51
  |- f^<*p*> implies |- f^<*p 'or' q*>;

theorem :: CALCL1_2:52
  |- f^<*q*> implies |- f^<*p 'or' q*>;

theorem :: CALCL1_2:53
  |- f^<*p*>^<*r*> & |- f^<*q*>^<*r*> implies |- f^<*p 'or' q*>^<* r*>;

theorem :: CALCL1_2:54
  |- f^<*p*> implies |- f^<*'not' 'not' p*>;

theorem :: CALCL1_2:55
  |- f^<*'not' 'not' p*> implies |- f^<*p*>;

theorem :: CALCL1_2:56
  |- f^<*p => q*> & |- f^<*p*> implies |- f^<*q*>;

theorem :: CALCL1_2:57
  ('not' p).(x,y) = 'not' (p.(x,y));

theorem :: CALCL1_2:58
  (ex y st |- f^<*p.(x,y)*>) implies |- f^<*Ex(x,p)*>;

theorem :: CALCL1_2:59
  still_not-bound_in (f^g) = still_not-bound_in f \/ still_not-bound_in g;
```

```
theorem :: CALCL1_2:60
  still_not-bound_in <*p*> = still_not-bound_in p;

theorem :: CALCL1_2:61
  |- f^<*p.(x,y)*>^<*q*> & not y in still_not-bound_in (f^<*Ex(x,p)*>^<*
  q*>) implies |- f^<*Ex(x,p)*>^<*q*>;

theorem :: CALCL1_2:62
  still_not-bound_in f = union {still_not-bound_in p : ex i st i
  in dom f & p = f.i};

theorem :: CALCL1_2:63
  still_not-bound_in f is finite;

theorem :: CALCL1_2:64
  card bound_FO-variables(Al) = card FO-symbols(Al) &
  not bound_FO-variables(Al) is finite;

theorem :: CALCL1_2:65
  ex x st not x in still_not-bound_in f;

theorem :: CALCL1_2:66
  |- f^<*All(x,p)*> implies |- f^<*All(x,'not' 'not' p)*>;

theorem :: CALCL1_2:67
  |- f^<*All(x,'not' 'not' p)*> implies |- f^<*All(x,p)*>;

theorem :: CALCL1_2:68
  |- f^<*All(x,p)*> iff |- f^<*'not' Ex(x,'not' p)*>;

definition
  let f be FinSequence, p be set;
  redefine pred p is_tail_of f means
:: CALCL1_2:def 16
  ex i being Element of NAT st i in dom f & f.i =  p;
end;
```

9.12 CALCL2_2

```
:: Consequences of the Sequent Calculus
::  by Patrick Braselmann and Peter Koepke
::
:: Received September 25, 2004
:: Copyright (c) 2004-2011 Association of Mizar Users
::           (Stowarzyszenie Uzytkownikow Mizara, Bialystok, Poland).
:: This code can be distributed under the GNU General Public Licence
:: version 3.0 or later, or the Creative Commons Attribution-ShareAlike
:: License version 3.0 or later, subject to the binding interpretation
:: detailed in file COPYING.interpretation.
:: See COPYING.GPL and COPYING.CC-BY-SA for the full text of these
:: licenses, or see http://www.gnu.org/licenses/gpl.html and
:: http://creativecommons.org/licenses/by-sa/3.0/.

environ

 vocabularies NUMBERS, SUBSET_1, CFO_LANG, FINSEQ_1, ORDINAL1, ARYTM_3,
      XXREAL_0, TARSKI, CARD_1, XBOOLE_0, NAT_1, FINSET_1, RELAT_1, ORDINAL4,
      FUNCT_1, CALCL1_2, FUNCT_2, CFO_THE1, FO_LANG1, XBOOLEAN, FINSEQ_5,
      ARYTM_1, FINSEQ_2, CALCL2_2;
 notations TARSKI, XBOOLE_0, SUBSET_1, ORDINAL1, XCMPLX_0, CARD_1, NUMBERS,
      XXREAL_0, NAT_1, RELAT_1, FUNCT_1, FINSEQ_1, FO_LANG1, CFO_LANG,
      FINSET_1,FINSEQ_5, FINSEQ_2, RELSET_1, FUNCT_2, WELLORD2, CALCL1_2;
 constructors PARTFUN1, WELLORD2, XXREAL_0, REAL_1, NAT_1, INT_1, FINSEQ_2,
      FINSEQ_5, CALCL1_2, RELSET_1, FO_LANG1;
 registrations FUNCT_1, ORDINAL1, RELSET_1, XXREAL_0, XREAL_0, NAT_1, INT_1,
      FINSEQ_1, FO_LANG1, CFO_LANG, FUNCT_2, FINSEQ_2, CARD_1;
 requirements REAL, NUMERALS, SUBSET, BOOLE, ARITHM;
 definitions TARSKI, XBOOLE_0, FINSEQ_2;
 theorems TARSKI, FINSEQ_1, FINSEQ_3, FUNCT_1, XBOOLE_0, FINSEQ_2, RELAT_1,
      NAT_1, XBOOLE_1, FUNCT_2, CALCL1_2, FO_LANG2, CARD_1, ORDINAL1, FUNCT_4,
      FINSEQ_5, INT_1, XREAL_1, XXREAL_0, FUNCOP_1, CFO_LANG;
 schemes NAT_1, RECDEF_1;

begin :: f is Subsequence of g^f

reserve Al for FO-alphabet;

reserve p,q,p1,p2,q1 for Element of CFO-WFF(Al),
  k,m,n,i for Element of NAT,
  f, f1,f2,g for FinSequence of CFO-WFF(Al),
  a,b,b1,b2,c for natural number;
```

```
definition
  let m,n be natural number;
  func seq(m,n) -> set equals
:: CALCL2_2:def 1
  {k : 1+m <= k & k <= n+m };
end;

definition
  let m,n be natural number;
  redefine func seq(m,n) -> Subset of NAT;
end;

theorem :: CALCL2_2:1
  c in seq(a,b) iff 1+a <= c & c <= b+a;

theorem :: CALCL2_2:2
  seq(a,0) = {};

theorem :: CALCL2_2:3
  b = 0 or b+a in seq(a,b);

theorem :: CALCL2_2:4
  b1 <= b2 iff seq(a,b1) c= seq(a,b2);

theorem :: CALCL2_2:5
  seq(a,b) \/ {a+b+1} = seq(a,b+1);

theorem :: CALCL2_2:6
  seq(m,n),n are_equipotent;

registration
  let m,n;
  cluster seq(m,n) -> finite;
end;

registration
  let Al;
  let f;
  cluster len f -> finite;
end;

theorem :: CALCL2_2:7
  seq(m,n) c= Seg (m+n);

theorem :: CALCL2_2:8
  Seg n misses seq(n,m);

theorem :: CALCL2_2:9
  for f,g be FinSequence holds dom(f^g) = dom f \/ seq(len f,len g);

theorem :: CALCL2_2:10
  len Sgm(seq(len g,len f)) = len f;

theorem :: CALCL2_2:11
  dom Sgm(seq(len g,len f)) = dom f;

theorem :: CALCL2_2:12
  rng Sgm(seq(len g,len f)) = seq(len g,len f);

theorem :: CALCL2_2:13
  i in dom Sgm(seq(len g,len f)) implies Sgm(seq(len g,len f)).i = len g+i;

theorem :: CALCL2_2:14
  seq(len g,len f) c= dom (g^f);

theorem :: CALCL2_2:15
  dom((g^f)|seq(len g,len f)) = seq(len g,len f);

theorem :: CALCL2_2:16
  Seq((g^f)|seq(len g,len f)) = Sgm(seq(len g,len f)) * (g^f);

theorem :: CALCL2_2:17
  dom Seq((g^f)|seq(len g,len f)) = dom f;

theorem :: CALCL2_2:18
  f is_Subsequence_of g^f;

definition
  let D be non empty set, f be FinSequence of D;
  let P be Permutation of dom f;
  func Per(f,P) -> FinSequence of D equals
:: CALCL2_2:def 2
  P*f;
end;
```

```
reserve P for Permutation of dom f;

theorem :: CALCL2_2:19
  dom Per(f,P) = dom f;

theorem :: CALCL2_2:20
  |- f^<*p*> implies |- g^f^<*p*>;

begin :: The Ordering of the Antecedent is Irrelevant

definition
  let Al;
  let f;
  func Begin(f) -> Element of CFO-WFF(Al) means
:: CALCL2_2:def 3

  it = f.1 if 1 <= len f otherwise it = VERUM(Al);
end;

definition
  let Al;
  let f;
  assume
 1 <= len f;
  func Impl(f) -> Element of CFO-WFF(Al) means
:: CALCL2_2:def 4

  ex F being FinSequence of
CFO-WFF(Al) st it = F.(len f) & len F = len f & (F.1 = Begin(f) or len f = 0)
 & for n st 1 <= n & n < len f holds ex p,q st p = f.(n+1) &
  q = F.n & F.(n+1) = p => q;
end;

:: Some details about the calculus in CALCL1_2

theorem :: CALCL2_2:21
  |- f^<*p*>^<*p*>;

theorem :: CALCL2_2:22
  |- f^<*p '&' q*> implies |- f^<*p*>;

theorem :: CALCL2_2:23
  |- f^<*p '&' q*> implies |- f^<*q*>;

theorem :: CALCL2_2:24
  |- f^<*p*> & |- f^<*p*>^<*q*> implies |- f^<*q*>;

theorem :: CALCL2_2:25
  |- f^<*p*> & |- f^<*'not' p*> implies |- f^<*q*>;

theorem :: CALCL2_2:26
  |- f^<*p*>^<*q*> & |- f^<*'not' p*>^<*q*> implies |- f^<*q*>;

theorem :: CALCL2_2:27
  |- f^<*p*>^<*q*> implies |- f^<*p => q*>;

theorem :: CALCL2_2:28
  1 <= len g & |- f^g implies |- f^<*Impl(Rev g)*>;

theorem :: CALCL2_2:29
  |- Per(f,P)^<*Impl(Rev (f^<*p*>))*> implies |- Per(f,P)^<*p*>;

theorem :: CALCL2_2:30
  |- f^<*p*> implies |- Per(f,P)^<*p*>;

begin :: Multiple Occurrence in the Antecedent is Irrelevant

notation
  let n;
  let c be set;
  synonym IdFinS(c,n) for n |-> c;
end;

theorem :: CALCL2_2:31
  for c being set st 1 <= n holds rng IdFinS(c,n) = rng <*c*>;

definition
  let D be non empty set, n be Element of NAT, p be Element of D;
  redefine func IdFinS(p,n) -> FinSequence of D;
end;

theorem :: CALCL2_2:32
  1 <= n & |- f^IdFinS(p,n)^<*q*> implies |- f^<*p*>^<*q*>;
```

9.13 HENMOD_2

```
:: Equivalences of Inconsistency and {H}enkin Models
::  by Patrick Braselmann and Peter Koepke
::
:: Received September 25, 2004
:: Copyright (c) 2004-2011 Association of Mizar Users
::           (Stowarzyszenie Uzytkownikow Mizara, Bialystok, Poland).
:: This code can be distributed under the GNU General Public Licence
:: version 3.0 or later, or the Creative Commons Attribution-ShareAlike
:: License version 3.0 or later, subject to the binding interpretation
:: detailed in file COPYING.interpretation.
:: See COPYING.GPL and COPYING.CC-BY-SA for the full text of these
:: licenses, or see http://www.gnu.org/licenses/gpl.html and
:: http://creativecommons.org/licenses/by-sa/3.0/.

environ

 vocabularies NUMBERS, SUBSET_1, CFO_LANG, FO_LANG1, FINSEQ_1, XBOOLE_0,
      FINSET_1, FUNCT_1, ORDINAL1, RELAT_1, ARYTM_3, TARSKI, WELLORD2,
      WELLORD1, CARD_1, NAT_1, CFO_THE1, ORDINAL4, XBOOLEAN, CALCL1_2,
      CALCL2_2, ARYTM_1, INT_1, XXREAL_0, FUNCT_2, ZFMISC_1, FO_VALUA,
      ZF_MODEL, MARGREL1, REALSET1, MCART_1, HENMOD_2;
 notations TARSKI, XBOOLE_0, ZFMISC_1, SUBSET_1, SETFAM_1, CALCL1_2, RELAT_1,
      FUNCT_1, ORDINAL1, XCMPLX_0, XXREAL_0, NAT_1, FINSEQ_1, FO_LANG1,
      CFO_LANG, CFO_THE1, FO_VALUA, FINSET_1, RELSET_1, FUNCT_2, CARD_1,
      MARGREL1, CFO_SIM1, DOMAIN_1, MCART_1, NUMBERS, FINSEQ_3, CALCL2_2,
      INT_1, WELLORD1, WELLORD2;
 constructors SETFAM_1, WELLORD1, WELLORD2, DOMAIN_1, XXREAL_0, NAT_1, INT_1,
      FINSEQ_3, CFO_SIM1, SUBST2_2, CALCL1_2, CALCL2_2, RELSET_1;
 registrations SUBSET_1, FUNCT_1, ORDINAL1, RELSET_1, FINSET_1, XXREAL_0,
      XREAL_0, INT_1, FINSEQ_1, FO_LANG1, CFO_LANG, FINSEQ_2, CARD_1;
 requirements REAL, NUMERALS, SUBSET, BOOLE, ARITHM;
 definitions TARSKI, XBOOLE_0, FUNCT_1, FINSEQ_2;
 theorems TARSKI, FINSEQ_1, FINSEQ_3, FUNCT_1, FO_VALUA, XBOOLE_0, FINSEQ_2,
      ZFMISC_1, RELAT_1, FO_LANG3, XBOOLE_1, NAT_1, MARGREL1, FUNCT_2,
      RELSET_1, FINSET_1, CALCL1_2, CFO_THE1, ORDINAL2, AXIOMS, CALCL2_2,
      CARD_2, INT_1, ORDINAL1, WELLORD1, WELLORD2, MCART_1, SUBST2_2, XREAL_1,
      XXREAL_0, FUNCOP_1, CARD_1;
 schemes FUNCT_1, FINSEQ_1, CLASSES1;

begin :: Preliminaries and Equivalences of Inconsistency

reserve Al for FO-alphabet;

reserve a,a1,a2,b,c,d for set,
  X,Y,Z for Subset of CFO-WFF(Al),
  i,k,m,n for Element of NAT,
  p,q for Element of CFO-WFF(Al),
  P for FO-pred_symbol of k,Al,
  ll for CFO-variable_list of k,Al,
  f,f1,f2,g for FinSequence of CFO-WFF(Al);
reserve A for non empty finite Subset of NAT;

theorem :: HENMOD_2:1
  for f being Function of n,A st ((ex m st succ m = n) & rng f = A
& for n,m st m in dom f & n in dom f & n < m holds f.n in f.m) holds f.(union n
  ) = union rng f;

theorem :: HENMOD_2:2
  union A in A & for a st a in A holds (a in union A or a = union A );

reserve C for non empty set;

theorem :: HENMOD_2:3
  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 holds ex k
  st X c= f.k;

definition
  let Al;
  let X,p;
  pred X |- p means
:: HENMOD_2:def 1

  ex f st rng f c= X & |- f^<*p*>;
end;

definition
  let Al;
  let X;
  attr X is Consistent means
:: HENMOD_2:def 2

  for p holds not (X |- p & X |- 'not' p);
end;

notation
  let Al;
  let X;
  antonym X is Inconsistent for X is Consistent;
end;

definition
  let Al;
  let f be FinSequence of CFO-WFF(Al);
  attr f is Consistent means
:: HENMOD_2:def 3

  for p holds not (|- f^<*p*> & |- f^<*'not' p*>);
end;

notation
  let Al;
  let f be FinSequence of CFO-WFF(Al);
  antonym f is Inconsistent for f is Consistent;
end;

theorem :: HENMOD_2:4
  X is Consistent & rng g c= X implies g is Consistent;

theorem :: HENMOD_2:5
  |- f^<*p*> implies |- f^g^<*p*>;

:: Ebb et al, Chapter IV, Lemma 7.2

theorem :: HENMOD_2:6
  X is Inconsistent iff for p holds X |- p;

:: One direction of Ebb et al, Chapter IV, Lemma 7.4

theorem :: HENMOD_2:7
  X is Inconsistent implies ex Y st Y c= X & Y is finite & Y is Inconsistent;

theorem :: HENMOD_2:8
  X \/ {p} |- q implies ex g st rng g c= X & |- g^<*p*>^<*q*>;

:: Corresponds to Ebb et al, Chapter IV, Lemma 7.6 (a)

theorem :: HENMOD_2:9
  X |- p iff X \/ {'not' p} is Inconsistent;

:: Similar to Ebb et al, Chapter IV, Lemma 7.6 (a)

theorem :: HENMOD_2:10
  X |- 'not' p iff X \/ {p} is Inconsistent;

begin :: Unions of Consistent Sets
:: Ebb et al, Chapter IV, Lemma 7.7

theorem :: HENMOD_2:11
  for f being Function of NAT,bool CFO-WFF(Al) st (for n,m st m in dom f & n
  in dom f & n < m holds f.n is Consistent & f.n c= f.m) holds (union rng f) is
  Consistent;

begin :: Construction of a Henkin Model

reserve A for non empty set,
  v for Element of Valuations_in(Al,A),
  J for interpretation of Al,A;

theorem :: HENMOD_2:12
  X is Inconsistent implies for J,v holds not J,v |= X;

theorem :: HENMOD_2:13
  {VERUM(Al)} is Consistent;

registration
  let Al;
  cluster Consistent for Subset of CFO-WFF(Al);
end;

reserve CX for Consistent Subset of CFO-WFF(Al),
  P9 for Element of FO-pred_symbols(Al);
```

```
definition
  let Al;
  func HCar(Al) -> non empty set equals
:: HENMOD_2:def 4
  bound_FO-variables(Al);
end;

definition
  let Al;
  let P be (Element of FO-pred_symbols(Al)), ll be CFO-variable_list of (
  the_arity_of P), Al;
  redefine func P!ll -> Element of CFO-WFF(Al);
end;

:: The Henkin Model (Ebb et al, Chapter V, §1)

definition
  let Al;
  let CX;
  mode Henkin_interpretation of CX -> interpretation of Al,HCar(Al) means
:: HENMOD_2:def 5

    for
P being (Element of FO-pred_symbols(Al)), r being Element of relations_on
HCar(Al) st it.P = r holds for a holds a in r iff
ex ll being CFO-variable_list of (the_arity_of P), Al st a = ll &
CX |- P!ll;
end;

definition
  let Al;
  func valH(Al) -> Element of Valuations_in (Al,HCar(Al)) equals
:: HENMOD_2:def 6
  id bound_FO-variables(Al);
end;

begin :: Some Properties of the Henkin Model

reserve JH for Henkin_interpretation of CX;

theorem :: HENMOD_2:14
  (valH(Al))*'ll = ll;

theorem :: HENMOD_2:15
  |- f^<*VERUM(Al)*>;

theorem :: HENMOD_2:16
  JH,valH(Al) |= VERUM(Al) iff CX |- VERUM(Al);

theorem :: HENMOD_2:17
  JH,valH(Al) |= P!ll iff CX |- P!ll;
```

9.14 GOEDCP_2

```
:: G{\"o}del's Completeness Theorem
::  by Patrick Braselmann and Peter Koepke
::
:: Received September 25, 2004
:: Copyright (c) 2004-2011 Association of Mizar Users
::           (Stowarzyszenie Uzytkownikow Mizara, Bialystok, Poland).
:: This code can be distributed under the GNU General Public Licence
:: version 3.0 or later, or the Creative Commons Attribution-ShareAlike
:: License version 3.0 or later, subject to the binding interpretation
:: detailed in file COPYING.interpretation.
:: See COPYING.GPL and COPYING.CC-BY-SA for the full text of these
:: licenses, or see http://www.gnu.org/licenses/gpl.html and
:: http://creativecommons.org/licenses/by-sa/3.0/.

environ

 vocabularies NUMBERS, SUBSET_1, CFO_LANG, FO_LANG1, XBOOLE_0, FO_VALUA,
      FINSEQ_1, HENMOD_2, CFO_THE1, XBOOLEAN, BVFUNC_2, FUNCT_1, ORDINAL4,
      CALCL1_2, ARYTM_3, RELAT_1, CARD_1, XXREAL_0, TARSKI, ZF_MODEL, CFO_SIM1,
      REALSET1, SUBST1_2, SUBST2_2, ZF_LANG, ARYTM_1, CARD_3, ZFMISC_1,
      FINSET_1, MCART_1, NAT_1, GOEDCP_2;
 notations TARSKI, XBOOLE_0, ZFMISC_1, SUBSET_1, XCMPLX_0, XXREAL_0, NAT_1,
      CARD_3, FINSEQ_1, FUNCT_1, FO_LANG1, FO_LANG2, FO_LANG3, NUMBERS,
      CFO_LANG, RELAT_1, FINSET_1, FO_VALUA, RELSET_1, FUNCT_2, CFO_SIM1,
      DOMAIN_1, MCART_1, SUBST1_2, SUBLEM_2, SUBST2_2, CALCL1_2, HENMOD_2;
 constructors SETFAM_1, DOMAIN_1, XXREAL_0, NAT_1, NAT_D, FINSEQ_2, FO_LANG3,
      CFO_SIM1, SUBST2_2, CALCL1_2, HENMOD_2, CARD_3, RELSET_1;
 registrations SUBSET_1, RELAT_1, ORDINAL1, XXREAL_0, XREAL_0, CFO_LANG,
      HENMOD_2, FINSEQ_1, FINSET_1, CARD_3, RELSET_1;
 requirements REAL, NUMERALS, SUBSET, BOOLE, ARITHM;
 definitions TARSKI, XBOOLE_0;
 theorems TARSKI, FUNCT_1, MCART_1, XBOOLE_0, XBOOLE_1, CFO_LANG, FO_LANG1,
      ZFMISC_1, RELAT_1, FO_LANG3, FO_LANG2, HENMOD_2, CALCL1_2, SUBLEM_2,
      NAT_1, FINSEQ_1, FO_VALUA, FUNCT_2, SUBST2_2, CFO_SIM1, CARD_4, CALCL2_2,
      SUPINF_2, XREAL_1, XXREAL_0, ORDINAL1;
 schemes XBOOLE_0, NAT_1, FUNCT_1, SUBST2_2, RECDEF_1;

begin :: Henkin‘s Theorem

registration
  cluster countable for FO-alphabet;
end;

reserve Al for FO-alphabet;

reserve b,c,d for set,
  X,Y for Subset of CFO-WFF(Al),
  i,j,k,m,n for Element of NAT,
  p,p1,q,r,s,s1 for Element of CFO-WFF(Al),
  x,x1,x2,y,y1 for bound_FO-variable of Al,
  A for non empty set,
  J for interpretation of Al, A,
  v for Element of Valuations_in(Al,A),
  f1,f2 for FinSequence of CFO-WFF(Al),
  CX,CY,CZ for Consistent Subset of CFO-WFF(Al),
  JH for Henkin_interpretation of CX,
  a for Element of A,
  t,u for FO-symbol of Al;

definition
  let Al;
  let X;
  attr X is negation_faithful means
:: GOEDCP_2:def 1

  X |- p or X |- 'not' p;
end;

definition
  let Al;
  let X;
  attr X is with_examples means
:: GOEDCP_2:def 2

  for x,p holds ex y st X |- ('not' Ex(x,p)) 'or' (p.(x,y));
end;

theorem :: GOEDCP_2:1
  CX is negation_faithful implies (CX |- p iff not CX |- 'not' p);

theorem :: GOEDCP_2:2
  for f being FinSequence of CFO-WFF(Al) holds
  |- f^<*'not' p 'or' q*> & |- f^<*p*> implies |- f^<*q*>;

theorem :: GOEDCP_2:3
  X is with_examples implies (X |- Ex(x,p) iff ex y st X |- p.(x,y));

theorem :: GOEDCP_2:4
  (CX is negation_faithful & CX is with_examples implies
  (JH,valH(Al) |= p iff CX |- p)) implies
  (CX is negation_faithful & CX is with_examples implies
  (JH,valH(Al) |= 'not' p iff CX |- 'not' p));

theorem :: GOEDCP_2:5
  |- f1^<*p*> & |- f1^<*q*> implies |- f1^<*p '&' q*>;

theorem :: GOEDCP_2:6
  X |- p & X |- q iff X |- p '&' q;

theorem :: GOEDCP_2:7
  (CX is negation_faithful & CX is with_examples implies
  (JH,valH(Al) |= p iff CX |- p)) &
  (CX is negation_faithful & CX is with_examples implies
  (JH,valH(Al) |= q iff CX |- q)) implies
  (CX is negation_faithful & CX is with_examples
  implies (JH,valH(Al) |= p '&' q iff CX |- p '&' q));

theorem :: GOEDCP_2:8
  for p st QuantNbr(p) <= 0 holds
  CX is negation_faithful & CX is with_examples implies
  (JH,valH(Al) |= p iff CX |- p);
```

```
theorem :: GOEDCP_2:9
  J,v |= Ex(x,p) iff ex a st J,v.(x|a) |= p;

theorem :: GOEDCP_2:10
  JH,valH(Al) |= Ex(x,p) iff ex y st JH,valH(Al) |= p.(x,y);

theorem :: GOEDCP_2:11
  J,v |= 'not' Ex(x,'not' p) iff J,v |= All(x,p);

theorem :: GOEDCP_2:12
  X |- 'not' Ex(x,'not' p) iff X |- All(x,p);

theorem :: GOEDCP_2:13
  QuantNbr(Ex(x,p)) = QuantNbr(p)+1;

theorem :: GOEDCP_2:14
  QuantNbr(p) = QuantNbr(p.(x,y));

reserve L for PATH of q,p,
  F1,F3 for FO-formula of Al,
  a for set;

theorem :: GOEDCP_2:15
  for p st QuantNbr(p) = 1 holds (CX is negation_faithful & CX is with_examples
  implies (JH,valH(Al) |= p iff CX |- p));

theorem :: GOEDCP_2:16
  for n st for p st QuantNbr(p) <= n holds
  (CX is negation_faithful & CX is with_examples
  implies (JH,valH(Al) |= p iff CX |- p)) holds
  for p st QuantNbr(p) <= n+1 holds
  (CX is negation_faithful & CX is with_examples
  implies (JH,valH(Al) |= p iff CX |- p));

:: Ebb et al, Chapter V, Henkin's Theorem 1.10

theorem :: GOEDCP_2:17
  for p holds (CX is negation_faithful & CX is with_examples
  implies (JH,valH(Al) |= p iff CX |- p));

begin :: Satisfiability of Consistent Sets of Formulas with Finitely Many Free
:: Variables

theorem :: GOEDCP_2:18
  Al is countable implies
  FO-WFF(Al) is countable;

definition
  let Al;
  func ExCl(Al) -> Subset of CFO-WFF(Al) means
:: GOEDCP_2:def 3

  a in it iff ex x,p st a = Ex(x,p);
end;

theorem :: GOEDCP_2:19
  Al is countable implies
  CFO-WFF(Al) is countable;

theorem :: GOEDCP_2:20
  Al is countable implies
  ExCl(Al) is non empty & ExCl(Al) is countable;

definition
  let Al;
  let p be Element of FO-WFF(Al) such that
 p is existential;
  func Ex-bound_in p -> bound_FO-variable of Al means
:: GOEDCP_2:def 4

  ex q being Element of FO-WFF(Al) st p = Ex(it,q);
end;

definition
  let Al;
  let p be Element of CFO-WFF(Al) such that
 p is existential;
  func Ex-the_scope_of p -> Element of CFO-WFF(Al) means
:: GOEDCP_2:def 5

  ex x st p = Ex(x,it);
end;
```

```
definition
  let Al;
  let F be Function of NAT,CFO-WFF(Al),a be Element of NAT;
  func bound_in(F,a) -> bound_FO-variable of Al means
:: GOEDCP_2:def 6

  p = F.a implies it = Ex-bound_in p;
end;

definition
  let Al;
  let F be Function of NAT,CFO-WFF(Al),a be Element of NAT;
  func the_scope_of(F,a) -> Element of CFO-WFF(Al) means
:: GOEDCP_2:def 7

  p = F.a implies it = Ex-the_scope_of p;
end;

definition
  let Al;
  let X;
  func still_not-bound_in X -> Subset of bound_FO-variables(Al) equals
:: GOEDCP_2:def 8
  union {still_not-bound_in p : p in X};
end;

theorem :: GOEDCP_2:21
  p in X implies X |- p;

theorem :: GOEDCP_2:22
  Ex-bound_in Ex(x,p) = x & Ex-the_scope_of Ex(x,p) = p;

theorem :: GOEDCP_2:23
  X |- VERUM(Al);

theorem :: GOEDCP_2:24
  X |- 'not' VERUM(Al) iff X is Inconsistent;

reserve C,D for Element of [:CFO-WFF(Al),bool bound_FO-variables(Al):];
reserve K,L for Subset of bound_FO-variables(Al);

theorem :: GOEDCP_2:25
  for f,g being FinSequence of CFO-WFF(Al) st 0 < len f & |- f^<*p*> holds
  |- Ant(f)^g^<*Suc(f)*>^<*p*>;

theorem :: GOEDCP_2:26
  still_not-bound_in {p} = still_not-bound_in p;

theorem :: GOEDCP_2:27
  still_not-bound_in (X \/ Y) = still_not-bound_in X \/ still_not-bound_in Y;

theorem :: GOEDCP_2:28
  for A being Subset of bound_FO-variables(Al) st A is finite holds
  ex x st not x in A;

theorem :: GOEDCP_2:29
  X c= Y implies still_not-bound_in X c= still_not-bound_in Y;

theorem :: GOEDCP_2:30
  for f being FinSequence of CFO-WFF(Al) holds
  still_not-bound_in rng f = still_not-bound_in f;

:: Ebb et al, Chapter V, Lemma 2.1

definition
  let Al;
  let A be set;
  func Al-one_in A -> FO-symbol of Al means
:: GOEDCP_2:def 9
  it = the Element of A if A is non empty Subset of FO-symbols(Al)
  otherwise it = 0;
end;

theorem :: GOEDCP_2:31
  ( Al is countable &
  still_not-bound_in CX is finite ) implies
  ex CY st CX c= CY & CY is with_examples;

theorem :: GOEDCP_2:32
  X |- p & X c= Y implies Y |- p;

reserve C,D for Subset of CFO-WFF(Al);

:: Ebb et al, Chapter V, Lemma 2.2
```

```
theorem :: GOEDCP_2:33
  ( Al is countable &
  CX is with_examples ) implies ( ex CY st CX c= CY &
  CY is negation_faithful & CY is with_examples );

reserve JH1 for Henkin_interpretation of CZ,
  J for interpretation of Al, A,
  v for Element of Valuations_in(Al,A);

theorem :: GOEDCP_2:34
  (Al is countable & still_not-bound_in CX is finite)
   implies ex CZ,JH1 st JH1,valH(Al) |= CX;

begin :: Goedel's Completeness Theorem,
:: Ebb et al, Chapter V, Completeness Theorem 4.1

theorem :: GOEDCP_2:35
  J,v |= X & Y c= X implies J,v |= Y;

theorem :: GOEDCP_2:36
  still_not-bound_in X is finite implies
  still_not-bound_in (X \/ {p}) is finite;

theorem :: GOEDCP_2:37
  X |= p implies not J,v |= X \/ {'not' p};

::$N Goedel Completeness Theorem
theorem :: GOEDCP_2:38
  ( Al is countable &
  still_not-bound_in X is finite & X |= p ) implies X |- p;
```