The ND Proof Assistant

1. Overview

The ND-ProofAssistant (Natural Deduction Proof Assistant) of LogicPalet helps students to construct correct formal proofs of mathematical statements in the context of predicate logic. The constructed proofs are guaranteed to be correct and are formulated in a highly intuitive way, very similar to the proofs one finds in courses on mathematical topics other than logic.

The proof assistant deals with only one theorem at a time. The theorem may depend on lemmas and claims that the students have to state and proof separately. Lemmas are linearly ordered and each lemma can use the other lemmas that appear after it in the ordering. Unlike lemmas, a claim inherits the hypotheses of the theorem, lemma, or claims that enclose it. The proof of a claim can use other claims that are stated within its proof, and so on. In this way, a claim and the claims used to prove it form a tree.

The statement of a theorem or lemma consists of hypotheses and a conclusion which are formulas in a predicate logic language. These formulas may contain free variables. The purpose is to prove that the conclusion is a logical consequence of the hypotheses: This means that for any structure (with signature compatible with the language) and for any values of the free variables, the conclusion is true if the hypotheses are true.

For simplicity of the deduction rules, the ND-ProofAssistant does not support using formulas that contain an inequality ≠. This does not hurt because an inequality can always be replaced by the negation of an equality.

To start the proof of a new theorem, click on the toolbar item "New" and follow the instructions. Always pay attention to the extensive tooltips on buttons, menus and toolbar items.

At any moment during the construction of a formal proof using the ND-ProofAssistant, exactly one proof is "activated". This is done by first selecting the header of the theorem, lemma, or claim whose proof one wants to activate and afterwards clicking on the toolbar item "Activate" (or pressing Ctrl+P). The header of the activated proof is indicated by preceding it with a magenta coloured icon and by appending " ⚡ (activated)".

With an "assertion" we mean a theorem, lemma or claim. The "formulation list" of an assertion is a linearly ordered list with the following items (in order): The header of the assertion (including an ordinal number), the declaration of constants (if there are any new ones), the hypotheses header (if there are at least two hypotheses), the hypotheses (with globally unique ordinal numbers)(if there are any), the (proposed) conclusion, and the proof header of the assertion (to indicate the end of the enumeration of hypotheses).

The "proof list" of an assertion, is an ordered list of items that are formulas, claim headers, or comments (e.g. to indicate the end of a claim). To each formula in the list is associated a (globally) unique ordinal number, and a justification. Each formula in the list must be obtained by one of the ND-deduction rules (see section 4 below) using some formulas, lemmas and/or claims that must satisfy several requirements (see section 3 below). The justification consists of

the name of the used ND-deduction rule and the ordinal numbers of the used formulas, lemmas and claims.

In the ND-ProofAssistant window there is a button for each ND-deduction rule. First select the items that the rule has to use and click on the button to append the deduced formula to the "proof list" of the assertion whose proof is activated.

We say that an assertion (theorem, lemma or claim) is "closed" if the last item of its "proof list" is the same as the proposed conclusion. Then we also call its proof header "closed". We say that an assertion is "all closed" if it is closed and all the claims that are "enclosed" by the assertion are closed (see section 2 below for the notion "enclosed"). We say that the theorem is "completed" when it is "closed' and all the lemmas and claims are closed (then the theorem and all lemmas and claims are fully proven).

2. The deduction tree

At any moment during the construction of a formal proof using the ND-ProofAssistant we see on the screen an ordered tree, except for its root which is hidden. We recall that an ordered tree is a rooted tree with the set of children of every node linearly ordered. It is easy to give a detailed description of the ordered tree we see on the screen. The nodes are the items of the "statement lists" and the "proof lists", together with the hidden root node. Only assertion headers, proof headers, and the root node have children. The set of children of an assertion header is the "statement list" of the assertion (with its header removed). The set of children of the proof header of an assertion is the "proof list" of the assertion. The first child of the root node is the header of the theorem, and the other children are the headers of the lemmas ordered by their ordinal numbers.

Let A be an assertion header or a proof header. Let F be a deduced formula, a hypothesis, or conclusion (resp. an assertion). We say that F is "enclosed" by A (or that A "encloses" F) if F (resp. the header of assertion F) is a descendant of A in the above mentioned ordered tree. (Recall that a child is a descendant and a child of a descendant is also a descendant.) When B is an assertion and F is as above, then we say that F is "enclosed by B" (or that B "encloses" F) if F is "enclosed" by the assertion header of B.

To each ordered tree is associated a binary rooted tree, called the "associated binary tree". The root and the nodes of the associated tree are the same as those of the original tree, but the relation 'x is a child of y' is now given by:

> x and y have the same parent z, and x is a direct successor of y in the linear ordering on the children of z, or x is the first child of y. (Here parent and child are meant with respect to the original tree).

The binary tree associated to the ordered tree on the screen (defined above) is called the "deduction tree" (at any moment during the construction of the formal proof). When we talk about a "path" (in the context of the ND-ProofAssistant) we will always mean a path in the deduction tree.

3. Requirements for items used by deduction rules

1/ The formulas used by a deduction rule must belong to the "path" of the deduction tree that starts at its root and ends at the parent of the deduced formula, excluding from this path the conclusions in the statement lists.

2/ The claims used by a deduction rule must belong to the "proof list" that contains the deduced formula, and these claims must precede (in this list) the deduced formula.

3/ The lemmas used by a deduction rule must be descendants in the deduction tree of the theorem or lemma that encloses the deduced formula.

These requirements ensure that there is no circularity in the reasoning. To help the students, the formulas and claims that can be used (by the deduction rules) in the activated proof are preceded by a blue icon.

Finaly there is one other requirement (but not related to what deduction rules use):

4/ All formulas appearing in the deduction tree must all belong to a same suitable predicate logic language. (For example, if R appears in two lemmas and in the first lemma it is a relation identifier with two arguments, then in the second lemma it must also be a relation with two arguments.)

4. The deduction rules

To state the Tautological Rule we first need some terminology:

Definition 1. Let $A_1, A_2, \dots, A_n$ and B be formulas of some predicate logic language. We say that B is a "tautological consequence" of $A_1, A_2, \dots, A_n$ if there exist formulas $P_1, P_2, \dots, P_n$ and Q in the language of propositional logic such that Q is a logical consequence of $P_1, P_2, \dots, P_n$ and such that the formulas $A_1, A_2, \dots, A_n$, B can be obtained from the formulas $P_1, P_2, \dots, P_n$, Q by substituting formulas of the predicate logic language for the propositional variables.

There is a straightforward algorithm using truth tables to determine whether or not B is a "tautological consequence" of $A_1, A_2, \dots, A_n$. Moreover this algorithm yields a proof whenever the answer is yes. This contrasts with the well known fact that there does not exist an algorithm to decide whether or not B is a logical consequence of $A_1, A_2, \dots, A_n$.

Definition 2. The "mono-reformulation" of an "assertion" (theorem, lemma, or claim) is the single formula equal to the implication with consequent the conclusion of the assertion and with antecedent the conjunction of the hypotheses (in case of a claim we mean the 'new' hypotheses, not these of the enclosing assertions).

We now state the Tautological Rule, under the assumption that all conditions outlined in Section 3 are met.

Tautological Rule

Using formulas $A_1, A_2, \dots, A_n$ and "assertions" $B_1, B_2, \dots, B_m$, one can deduce any formula that is a "tautological consequence" of $A_1, \dots, A_n$ and the mono-reformulations of the "assertions" $B_1, B_2, \dots, B_m$.

Remark: Using an assertion with hypothesis A and conclusion B one can deduce $A \Rightarrow B$ by the Tautological Rule.

Remark: If the deduction tree contains an assertion with no hypotheses, then to use the conclusion of the assertion one can apply the Tautological Rule.

Remark: Using a formula A one can deduce the same formula A by the Tautological Rule. In this way you can copy an already deduced formula to some other locations.

To state the next rules we need the following terminology:

Definition 3. Let A be a formula, x be a variable, and t be a term (in a predicate logic language). The formula obtained from A by replacing each free occurrence of x by t is denoted by A[t|x]. It is called the formula obtained from A by substituting t for x. We say that the substitution A[t|x] is "permitted" if no variable in the term t gets bounded in A[t|x] at the places where x is replaced by t in A.

Next we state the six rules about quantifiers, always under the assumption that all conditions outlined in Section 3 are met. Furthermore, formulas and terms are meant in the sense of predicate logic.

Universal Quantifier Elimination Rule

Let A be a formula, x be a variable, and t be a term. Using the formula ∀x: A, one can deduce the formula A[t|x], provided that the substitution A[t|x] is “permitted”.

Existential Quantifier Elimination Rule

Let A be a formula and x be a variable. Using the formula ∃x: A one can deduce the formula A[s(v1, v2, ...)|x] where s is a function identifier that does not yet occur in any formula that is "enclosed" by the theorem or lemma that “encloses” the activated proof header, and where v_1, v_2, ... is a list of all free variables of the formula ∃x: A. Moreover it is required that the substitution A[s(v_1, v_2, ...)|x] is "permitted". If the list v_1, v_2, ... is empty then s must be a constant identifier and the deduced formula is A[s|x].

Universal Quantifier Introduction Rule

Let A be a formula and x be a variable. Using A one can deduce the formula ∀x: A, provided that x does not occur free anywhere in the hypotheses of assertions that are "enclosed" by the theorem or lemma that encloses the activated proof header. (Note that x is allowed to occur free in the hypotheses of assertions that are not “enclosed” by the theorem or lemma that “encloses” the activated proof header.)

Existential Quantifier Introduction Rule

Let A be a formula, x be a variable, and t be a term. Using the formula A[t|x] one can deduce the formula ∃x: A, provided that the substitution A[t|x] is “permitted”.

Quantifier Renaming Rule

Let A be a formula, and x, y be variables. Let B be a formula that contains the formula ∃x: A as a subformula at a certain location. Let B' be the formula obtained from B by replacing the subformula ∃x: A at that location by the formula ∃y: A[y|x]. Using B one can deduce B', provided that the substitution A[y|x] is permissible and that y does not occur free anywhere in A. The same also applies to the formula ∀x: A instead of ∃x: A.

Quantifier Negation Rule

Let A be a formula and x be a variable. Using the formula ¬∀x: A one can deduce the formula ∃x: ¬A. Using the formula ¬∃x: A one can deduce the formula ∀x: ¬A.

Actually the Quantifier Negation Rule can be easily proven from the other deduction rules. But we include it for efficiency.

Equality Rule

Using no formulas one can deduce any equality of the form t = t, where t is any term. Using any formula A one can deduce any formula B obtained from A by replacing (possibly several times) any subformula of the form t = u by u = t, where t and u are terms. Using any formula A and any formula of the form $t_1 = t_2$, with t_1 and t_2 terms, one deduces any formula B obtained from A by replacing (possibly several times) t_1 by t_2 and/or t_2 by t_1, and/or replacing (possibly several times) any subformula of the form t = u by u = t, where t and u are terms.

Soundness and Completeness

For the proofs of soundness and completeness of the deduction rules we refer to http://www.logicpalet.be/ND-ProofAssistant-SoundAndComplete.pdf