Hilbert II

Presentation of
Formal Correct
Mathematical Knowledge

Logical Language

Michael Meyling

July 30, 2011

The source for this document can be found here:

`http://www.qedeq.org/0_04_04/doc/project/qedeq_logic_language.xml`

If you have any questions, suggestions or want to add something to the list of modules that use this one, please send an email to the address mime@qedeq.org

The authors of this document are: Michael Meyling michael@meyling.com

Contents

Description

The project **Hilbert II** includes formal correct mathematical knowledge. Here we introduce the underlying formal language for the mathematical formulas. This is done in an informal way. Important theorems (e.g.: universal decomposition, and any proofs) are left out.

All we will do is manipulate symbols. We build lists of symbol strings and use certain simple rules to get new lists. So by starting with a few basic lists we create a whole universe of derived symbol lists. It turns out that these lists could be interpreted as a view to the incredible world of mathematics.

Chapter 1

Entities

To describe the logical language we firstly deal with a more basic notation. This notation enables us to formulate the syntax of formulas and terms later on.

1.1 Elements, Atoms and Lists

The basic structure we have to deal with is an element. An element is either an atom or a list.

An atom carries textual data, atoms are just strings.

Each list has an operator and can contain elements again. An operator is also nothing more than a simple string. A list has a size: the number of elements it contains. Their elements can be accessed by their position number. An atom has no operator, no size and no subelements in the previous sense.

1.2 List Notation

Lists and atoms can be written in the following manner. We write down string atoms quoted with `"` and the lists as the contents of the operator string followed by `(` and a comma separated list of elements and an closing `)`.

1.3 Examples

In this syntax we can write down the following element examples.

```
"I am a string atom"

EMPTY_LIST()

THIS_LIST("contains", "three", "atoms")

OPERATOR("argument 1", "argument 2")

FUNCTION_A(FUNCTION_B("1", "2"), "3")
```

In the last example we have a list that has the operator `FUNCTION_A` and contains two elements. The first element is `FUNCTION_B("1", "2")` which is a list too. The second element is the atom `"3"`.

Chapter 2

Logical Language

There are different basic things we have to do with. These are predicates, functions, subject variables and logical connectives. In the following all of them are named and described.

2.1 Logical Operator Overview

Lists are categorized according to their operators. Before we introduce the formal language in detail the used operators are briefly listed.

logical

AND	logical conjunction operator	$\land$
OR	logical disjunction operator	$\lor$
IMPL	logical implication operator	$\rightarrow$
EQUI	logical biconditional operator	$\leftrightarrow$
NOT	logical negation operator	$\neg$

logical quantifiers

FORLL	universal quantifier	$\forall$
EXISTS	existential quantifier	$\exists$
EXISTSU	unique existential quantifier	$\exists!$

variables

VAR	subject variables	$x, y, z, \ldots$
PREDVAR	predicate variables	$A, B, R, \ldots$
FUNVAR	function variables	$f, g, h, \ldots$

constants

PREDCON	predicate constants	$=, \in, \subseteq, \ldots$
FUNCON	function constants	$\emptyset, \mathfrak{P}, \ldots$
CLASS	class term	$\{x \vert \phi(x)\}$

2.2 Terms and Formulas

Now we define recursivly our formal language. We call some elements *subject variables*, *terms* and some other *formulas*. We also define the relations a subject variable *is free in* and *is bound in* a term or a formula. If something is not according to the formal rules errors occur. The error codes are also described.

2.2.1 General Error Codes

The atoms and lists that build up a formula or term are subject to restrictions. The following errors occur if an atom has no content or has content with length of 0 or an list has no operator or one of its sub-elements does not exist. These are mainly technical error codes, only the error code 30470 shows an semantical error.

30400	no element	an element doesn't exist - it is null
30410	no atom	an atom doesn't exist - it is null
30420	no list	a list doesn't exist - it is null
30430	no atom content	an atom has no content - it is null
30440	atom content empty	an atom has content with 0 length
30450	no operator	a list has no operator - it is null
30460	operator empty	a list has an operator with 0 length
30470	list expected	list element expected but not found

2.2.2 Subject Variable

We call an element *subject variable* iff it has the operator *VAR* and its list size is 1 with an atom as its only argument.

Each subject variable is also called a *term*. Only the subject variable itself is free in itself. No subject variable is bound in a subject variable.

30710	not exactly one argument	list has not exactly one element
30730	atom element expected	the first and only list element must be an atom

2.2.3 Function Term

If an element has the operator *FUNVAR* or *FUNCON* and its list size is greater than or equal to 1 with an atom as its first argument and the remaining arguments are all terms then it is called a term too.

Iff a subject variable is free in any sub-element it is also free in the new term. No other subject variables are free. Analogous for bound subject variables.

30720	argument(s) missing	if operator is *FUNCON* the list must have at least one element
30730	atom element expected	the first list element must be an atom
30740	argument(s) missing	if operator is *FUNVAR* the list must have more than one element
30770	free bound mixed	found a bound subject variable that is already free in a previous list element
30780	free bound mixed	found a free subject variable that is already bound in a previous list element
30690	undefined constant	the operator is *FUNCON* and this function constant has not been defined for this argument number

Any other error for term checks may occur due to the fact that all (but the first) sub-elements must be terms too.

2.2.4 Predicate Formula

If an element has the operator *PREDVAR* or *PREDCON* and its list size is greater than or equal to 1 with an atom as its first argument and the remaining arguments are all terms and no errors occur then it is called a *formula*.

Iff a subject variable is free in any sub-element it is also free in the new formula. No other subject variables are free. Analogous for bound subject variables.

30720	argument(s) missing	list must have at least one element
30730	atom element expected	the first list element must be an atom
30770	free bound mixed	found a bound subject variable that is already free in a previous list element
30780	free bound mixed	found a free subject variable that is already bound in a previous list element
30590	undefined constant	the operator is *PREDCON* and this predicate constant has not been defined for this argument number

Any other error for formula checks may occur due to the fact that all (but the first) sub-elements must be terms.

2.2.5 Logical Connectives

If an element has the operator *AND*, *OR*, *IMPL* or *EQUI* and its list size is greater than or equal to 2 and the remaining arguments are all formulas and no errors occur then it is called a formula too.

Iff a subject variable is free in any sub-element it is also free in the new formula. No other subject variables are free. Analogous for bound subject variables.

30740	argument(s) missing	list must have more than one element
30760	exactly 2 elements expected	the operator is *IMPL* and this list size is not equal to 2
30770	free bound mixed	found a bound subject variable that is already free in a previous list element
30780	free bound mixed	found a free subject variable that is already bound in a previous list element

Any other error for formula checks may occur due to the fact that all sub-elements must be formulas.

2.2.6 Negation

If an element has the operator *NOT*, its list size is exactly 1 and its only sub-element arguments is a formula then it is called a formula too.

Iff a subject variable is free in the sub-element it is also free in the new formula. No other subject variables are free. Analogous for bound subject variables.

30710	exactly 1 argument expected	list must have exactly than one element

Any other error for formula checks may occur due to the fact that the sub-element must be a formula.

2.2.7 Quantifiers

If an element has the operator *FORALL*, *EXISTS* or *EXISTSU* its first sub-element is a subject variable and its second and perhaps its third sub-element is a formula then the element is called a *formula* too.

Iff a subject variable is free in the sub-element it is also free in the new formula. No other subject variables are free. Analogous for bound subject variables.

30760	2 or 3 arguments expected	list must have exactly 2 or 3 elements
30540	subject variable expected	first sub-element must be a subject variable
30550	already bound	subject variable already bound in second or third sub-element
30770	free bound mixed	found a bound subject variable that is already free in a previous list element
30780	free bound mixed	found a free subject variable that is already bound in a previous list element

Any other error for formula checks may occur due to the fact that the sub-element must be a formula.

2.2.8 Class Term

An list element with the operator *CLASS*, containing an subject variable and an formula is a term.

Iff a subject variable is free in the formula and is not equal to the first sub-element (which is a subject variable) it is also free in the new term. No other subject variables are free. If a subject variable is bound in the formula it is bound in the new term. Also the first sub-element is bound. No other subject variables are bound.

30760	2 arguments expected	the list must contain exactly two arguments
30540	subject variable expected	the first sub-element must be a subject variable
30550	already bound	the subject variable is already bound in the formula
30680	undefined class operator	the class operator is still unknown

Any other error for formula checks may occur due to the fact that the second sub-element must be a formula.

2.2.9 Term

When checking an element for beeing a term the element must have the operator for a *Subject Variable*, *Function Term* or *Class Term*.

30620	unknown term operator	element has no operator that is known as a term operator

Any other error for the accordant operator checks may occur.

2.2.10 Formula

When checking an element for beeing a formul the element must have the operator for a *Predicate Formula*, *Logical Connective*, *Negation* or *Quantifier*.

30530	unknown logical operator	element has no known logical operator

Any other error for the accordant operator checks may occur.

Chapter 3

Representations

The representation of elements differ according to the viewpoint. Lets take the following formula for example.

$$y \ = \ \{x \mid \phi(x)\} \ \leftrightarrow \ \forall z \ (z \in y \ \leftrightarrow \ z \in \{x \mid \phi(x)\})$$

The predicate constant $\in$ must have been defined in previous sections.

3.1 List Notation

In list notation (see **??**) the above formula looks like the following.

```
EQUI(
  PREDCON(
    "equal",
    VAR("y"),
    CLASS(
      VAR("x"),
      PREDVAR(
        "\phi",
        VAR("x")
      )
    )
  ),
  FORALL(
    VAR("z"),
    EQUI(
      PREDCON(
        "in",
        VAR("z"),
        VAR("y")
      ),
      PREDCON(
        "in",
        VAR("z"),
        CLASS(
          VAR("x"),
          PREDVAR(
            "\phi",
            VAR("x")
```

```
          )
        )
      )
    )
  )
)
```

3.2 Java

The list notation leads directly to the following Java code.

```
Element el = new ElementListImpl("EQUI", new Element[] {
    new ElementListImpl("PREDCON", new Element[] {
        new AtomImpl("equal"),
        new ElementListImpl("VAR", new Element[] {
            new AtomImpl("y"),
        }),
        new ElementListImpl("CLASS", new Element[] {
            new ElementListImpl("VAR", new Element[] {
                new AtomImpl("x"),
            }),
            new ElementListImpl("PREDVAR", new Element[] {
                new AtomImpl("\\phi"),
                new ElementListImpl("VAR", new Element[] {
                    new AtomImpl("x"),
                })
            })
        })
    }),
    new ElementListImpl("FORALL", new Element[] {
        new ElementListImpl("VAR", new Element[] {
            new AtomImpl("z"),
        }),
        new ElementListImpl("EQUI", new Element[] {
            new ElementListImpl("PREDCON", new Element[] {
                new AtomImpl("in"),
                new ElementListImpl("VAR", new Element[] {
                    new AtomImpl("z"),
                }),
                new ElementListImpl("VAR", new Element[] {
                    new AtomImpl("y"),
                })
            }),
            new ElementListImpl("PREDCON", new Element[] {
                new AtomImpl("in"),
                new ElementListImpl("VAR", new Element[] {
                    new AtomImpl("z"),
                }),
                new ElementListImpl("CLASS", new Element[] {
                    new ElementListImpl("VAR", new Element[] {
                        new AtomImpl("x"),
                    }),
                    new ElementListImpl("PREDVAR", new Element[] {
                        new AtomImpl("\\phi"),
                        new ElementListImpl("VAR", new Element[] {
```

```
                        new AtomImpl("x"),
                    })
                })
            })
        })
    })
})
});
```

3.3 XML

The XML representation within an QEDEQ module looks a little bit different. Here all first list atoms are represented as the attribute `ref` or `id`. So the above formula may look like the following.

```
<EQUI>
  <PREDCON ref="equal">
    <VAR id="y"/>
    <CLASS>
      <VAR id="x"/>
      <PREDVAR id="\phi">
        <VAR id="x"/>
      </PREDVAR>
    </CLASS>
  </PREDCON>
  <FORALL>
    <VAR id="z"/>
    <EQUI>
      <PREDCON ref="in">
        <VAR id="z"/>
        <VAR id="y"/>
      </PREDCON>
      <PREDCON ref="in">
        <VAR id="z"/>
        <CLASS>
          <VAR id="x"/>
          <PREDVAR id="\phi">
            <VAR id="x"/>
          </PREDVAR>
        </CLASS>
      </PREDCON>
    </EQUI>
  </FORALL>
</EQUI>
```

Due to XSD restrictions for the XML document some error codes listed in Chapter 1 will not occur. Instead the XML will be classified as invalid.

Chapter 4

Document structure

In this chapter we make some remarks about the QEDEQ XML format.

4.1 Basic structure

The mathematical knowledge of this project is organized in so called QEDEQ modules. Such a module can be read and edited with a simple text editor. It could contain references to other QEDEQ modules which lay anywhere in the world wide web.

A QEDEQ module is build like a mathematical text book. It's main structure looks like an LaTeX book file. It contains chapters which are composed of sections and sections are composed of subsections. A subsection may be pure text or an so called *node*. A node is either an axiom, definition, proposition or rule. Every node has an id and could be referenced by that. Essential formal elements of a node are formulas.

The formal definition of an QEDEQ XML document can be found here: `http://www.qedeq.org/0_04_04/xml/qedeq/noNamespace/element/QEDEQ.html`.

4.2 References

In QEDEQ documents reference links are used very often. There exist four goals for references: modules, nodes, sub formulas and proof lines.

If you want to address an external module you have to know its import *label*. See `http://www.qedeq.org/0_04_04/xml/qedeq/noNamespace/element/QEDEQ.HEADER.IMPORTS.IMPORT.html`.

A reference to a node needs the *id* of that node. See `http://www.qedeq.org/0_04_04/xml/qedeq/noNamespace/element/NODE.html`.

In certain cases it is also possible to reference a subformula of a proposition formula. This is only possible if the proposition formula is a conjunction (e.g. the top level logical operation is a conjunction). For each parameter a label is automatically generated. If the number of conjunction parameters is below 27 the label is simply the n'th alphabet character. If the number is greater 26 the label is written in the 26 system with alphabet characters as digits. To reference to a subformula of an external node the syntax is `importLabel.nodeId/subRef`.

You can also reference to a fromal proof line *label*, see `http://www.qedeq.org/0_04_04/xml/qedeq/noNamespace/element/L.html`. Within the node you just need to link to the label. Outside the node context (but within the same module) the syntax is `nodeId!lineLabel`.

Here follows a reference summary.

external module	`importLabel`
(external) node reference	`[importLabel.]nodeId`
(external) node sub formula ref.	`[importLabel.]nodeId/subRef`
(external) node proof line ref.	`[[importLabel].nodeId!]lineLabel`

Chapter 5

Basic Rules of Inference

To get new formulas from already proven or given ones we introduce *proof rules*. We can call a formula *proposition* if we can write down a sequence of formulas that derive it from axioms, definitions and propositions by applying proof methods. Such a sequence is called a *proof*. It is made of *proof lines*. A proof line is a formula and a proof rule usage with its parameters. Each proof line has a label. The last formula of a proof must be the proposition formula itself.

We will introduce the following proof rules.

Add	add already proven formula
MP	modus ponens
Rename	rename bound subject variable
SubstFree	substitute free subject variable by term
SubstFun	substitute function variable by term
SubstPred	substitute predicate variable by formula
Universal	universal generalization
Existential	existential generalization

These basic rules get the rule version number 0.01.00. The rules might get extended in higher rule versions.[1]

TODO 20110612 m31: add error code description for rules

5.1 Addition

Addition of an axiom, definition or already proven formula. We have to reference to the location of a true formula.

name	`Add`	name of proof rule
parameter 1	`ref`	reference to axiom, definition or proposition

See `http://www.qedeq.org/0_04_04/xml/qedeq/noNamespace/element/ADD.html`.

5.2 Modus Ponens

Modus Ponens (Conditional Elimination)

[1]For example we want to allow modus ponens also with a formula like $A \leftrightarrow B$.

$$\frac{\begin{array}{c}A\\ A \rightarrow B\end{array}}{B}$$

This rule states that if each of A and $A \rightarrow B$ are already true formulas then B is also a true formula.

name	`MP`	name of proof rule
parameter 1	`ref1`	reference to a proof line label with a formula like A
parameter 2	`ref2`	reference to a another proof line label with a formula like $A \rightarrow B$

See http://www.qedeq.org/0_04_04/xml/qedeq/noNamespace/element/MP.html.

5.3 Rename bound subject variable

We may replace a bound subject variable occurring in a formula by any other subject variable, provided that the new variable occurs not free in the original formula. If the variable to be replaced occurs in more than one scope, then the replacement needs to be made in one scope only. For example in this case we replace x by y at the first occurrence.

$$\frac{\ldots \forall x A(x) \ldots}{\ldots \forall y A(y) \ldots}$$

name	`Rename`	name of proof rule
parameter 1	`ref`	reference to a proof line label
parameter 2	`original`	bound subject variable that should be renamed
parameter 3	`replacement`	new name for subject variable
parameter 4	`occurrence`	bound occurence where we want to replace

5.4 Substitute free subject variable by term.

A free subject variable may be replaced by an arbitrary term, provided that the substituted term contains no subject variable that have a bound occurrence in the original formula. All occurrences of the free variable must be simultaneously replaced.

$$\frac{A(x)}{A(t)}$$

name	`SubstFree`	name of proof rule
parameter 1	`ref`	reference to a proof line label
parameter 2	`original`	free subject variable that should be replaced
parameter 3	`replacement`	replacement term

See http://www.qedeq.org/0_04_04/xml/qedeq/noNamespace/element/SUBST_FREE.html.

5.5 Substitute predicate variable by formula

Let α be a true formula that contains a predicate variable p of arity n, let x_1, $\ldots$, x_n be pairwise different subject variables and let $\beta(x_1, \ldots, x_n)$ be a formula where $x_1, \ldots, x_n$ are not bound. The formula $\beta(x_1, \ldots, x_n)$ must not contain all $x_1, \ldots, x_n$ as free subject variables. Furthermore it can also have other subject variables either free or bound.

If the following conditions are fulfilled, then a replacement of all occurrences of $p(t_1, \ldots, t_n)$ each with appropriate terms $t_1, \ldots, t_n$ in α by $\beta(t_1, \ldots, t_n)$ results in another true formula.

- the free variables of $\beta(x_1, \ldots, x_n)$ without $x_1, \ldots, x_n$ do not occur as bound variables in α
- each occurrence of $p(t_1, \ldots, t_n)$ in α contains no bound variable of $\beta(x_1, \ldots, x_n)$
- the result of the substitution is a well-formed formula

$$\frac{A(\sigma)}{A(\tau)}$$

name	`SubstPred`	name of proof rule
parameter 1	`ref`	reference to a proof line label
parameter 2	`original`	predicate variable that should be replaced
parameter 3	`replacement`	replacement formula

See http://www.qedeq.org/0_04_04/xml/qedeq/noNamespace/element/SUBST_PREDVAR.html.

5.6 Substitute function variable by term

Let α be an already proved formula that contains a function variable σ of arity n, let $x_1, \ldots, x_n$ be pairwise different subject variables and let $\tau(x_1, \ldots, x_n)$ be an arbitrary term where $x_1, \ldots, x_n$ are not bound. The term $\tau(x_1, \ldots, x_n)$ must not contain all $x_1, \ldots, x_n$ as free subject variables. Furthermore it can also have other subject variables either free or bound.

If the following conditions are fulfilled we can obtain a new true formula by replacing each occurrence of $\sigma(t_1, \ldots, t_n)$ with appropriate terms $t_1, \ldots, t_n$ in α by $\tau(t_1, \ldots, t_n)$.

- the free variables of $\tau(x_1, \ldots, x_n)$ without $x_1, \ldots, x_n$ do not occur as bound variables in α
- each occurrence of $\sigma(t_1, \ldots, t_n)$ in α contains no bound variable of $\tau(x_1, \ldots, x_n)$
- the result of the substitution is a well-formed formula

$$\frac{A(\sigma)}{A(\tau)}$$

name	`SubstFun`	name of proof rule
parameter 1	`ref`	reference to a proof line label
parameter 2	`original`	function variable that should be replaced
parameter 3	`replacement`	replacement term

See http://www.qedeq.org/0_04_04/xml/qedeq/noNamespace/element/SUBST_FUNVAR.html.

5.7 Universal Generalization

If $\alpha \rightarrow \beta(x_1)$ is a true formula and α does not contain the subject variable x_1, then $\alpha \rightarrow (\forall x_1\ (\beta(x_1)))$ is a true formula too.

$$\frac{\alpha \rightarrow \beta(x_1)}{\alpha \rightarrow (\forall x_1\ (\beta(x_1)))}$$

name	`Universal`	name of proof rule
parameter 1	`ref`	reference to a proof line label
parameter 2	`var`	subject variable we want to quantify with

See http://www.qedeq.org/0_04_04/xml/qedeq/noNamespace/element/UNIVERSAL.html.

5.8 Existential Generalization

If $\alpha(x_1) \rightarrow \beta$ is already proved to be true and β does not contain the subject variable x_1, then $(\exists x_1\ \alpha(x_1)) \rightarrow \beta$ is also a true formula.

$$\frac{\alpha(x_1) \rightarrow \beta}{(\exists x_1\ \alpha(x_1)) \rightarrow \beta}$$

name	`Existential`	name of proof rule
parameter 1	`ref`	reference to a proof line label
parameter 2	`var`	subject variable we want to quantify with

See http://www.qedeq.org/0_04_04/xml/qedeq/noNamespace/element/EXISTENTIAL.html.

Chapter 6

Derived Rules

We can use derived rules that can be completely replaced by the old rules but enable us shorter proofs. We introduce a new rule that allows us to make an assumption and derive from that hypothesis. All previous rules get also slightly modified.

CP	conditional proof

These basic rules get the rule version number 0.02.00.

6.1 Conditional Proof

We have the well-formed formula α and add it as a new proof line. We assume this formula as hypothesis. Now we modify the existing inference rules. We can add a further proof line β if $\alpha \rightarrow \beta$ is a well-formed formula and the usage of a previous inference rule with the following restrictions justifies the addition: any substitution of a free subject variable, a predicate variable or a function variable is only allowed, if the variable doesn't occur in α.

This rule can be used recursive. The conjunction of all hypothesis formulas is called a *condition* for the proof line we want to check.

name	`CP`	name of proof rule
parameter 1	`HYPOTHESIS`	hypothesis
parameter 2	`LINES`	formal proof that uses the hypothesis
parameter 2	`CONCLUSION`	implication with hypothesis and last proof line

See http://www.qedeq.org/0_04_04/xml/qedeq/noNamespace/element/CP.html.

Index