A Controlled Language
for Software Specifications & Mathematical text

Muhammad Humayoun

Department of Mathematics (LAMA)
University of Savoie
mhuma@univ-savoie.fr

2007 – 2010
23.10.2009
Outline

1. Motivation
2. Examples
3. Implementation details
Outline

1. Motivation
2. Examples
3. Implementation details
Problem 1

- Specifications & standards written in natural languages (RFCs, ISO, ANSI, patents etc)
- Often **incomplete** or **imprecise**
- Considerable room for **errors** and **misunderstandings**

---

1 A. Biltcliffe et al. 2006. Rigorous Protocol Design in Practice: An Optical Packet-Switch MAC in HOL. The 14th IEEE Int. Conf. on Network Protocols
Problem 1

- Specifications & standards written in natural languages (RFCs, ISO, ANSI, patents etc)

- Often **incomplete** or **imprecise**

- Considerable room for **errors** and **misunderstandings**

- Formal methods (**formal specs**) not readily accepted

- Formalisation is **hard**
  - Hard to understand & difficult to relate
  - Learning time - model checkers, theorem provers

---

1 A. Biltcliffe et al. 2006. Rigorous Protocol Design in Practice: An Optical Packet-Switch MAC in HOL. The 14th IEEE Int. Conf. on Network Protocols

Problem 2

- Mathematical text written in natural languages
- Well understood by mathematicians
Problem 2

- Mathematical text written in natural languages
- Well understood by mathematicians
- Often not precise enough for formalisation
- Reasoning gaps in mathematical proofs
Problem 2

- Mathematical text written in natural languages
- Well understood by mathematicians
- Often not precise enough for formalisation
- Reasoning gaps in mathematical proofs
- Formalised mathematics: formalisation is hard
- Theorem Provers not readily accepted by mathematicians
Problem 2

- Mathematical text written in natural languages
- Well understood by mathematicians
- Often not precise enough for formalisation
- Reasoning gaps in mathematical proofs
- Formalised mathematics: formalisation is hard
- Theorem Provers not readily accepted by mathematicians

Both problems are quite similar
Can we solve these similar problems together?
Sentence level: both have similar structure

Natural language + formal expressions & notations
Math & Specifications: **Together?**

Sentence level: both have similar structure
Natural language + formal expressions & notations

- If \( s \) is a scheduler state and \( \ldots \) then the scheduling algorithm \( S \) is non-starving.
- If \( y \) is even then \( 2y \) is also even.
- \( q_0 \) and \( q_1 \) are empty.
- \( x \) and \( y \) are positive.
- For all \( s \), if \( s \) is a valid scheduler state then \( \ldots \)
- For all \( x \), if \( x \) is an even integer then \( \ldots \)
- There exists \( p \) such that the running process of \( ssp \) is \( i \).
- There exist two distinct projections of a set \( S \).
- \( ss_0 = s \) and \( ss_{n+1} = S(ts(n), ss_n) \)
- \( F_n = F_{n-1} + F_{n-2} \).
Math & Specifications: Together?

- Proving specifications as a mathematical activity
  - Any algorithm could be developed within the framework of an axiomatic theory
  - Each program statement obeys a formal definition.
Math & Specifications: Together?

- **Proving specifications as a mathematical activity**
  - Any algorithm could be developed within the framework of an axiomatic theory
  - Each program statement obeys a formal definition.

- **Notion of time**: every variable is a function of time
  - Imperative programming
    - e.g. \( x = x + y \) means \( x(t + 1) = x(t) + y(t) \)
  - Differential equations
    - e.g. \( x' = ax + c \) means \( x'(t) = ax(t) + c \)
Math & Specifications: Together?

- Proving specifications as a mathematical activity
  - Any algorithm could be developed within the framework of an axiomatic theory
  - Each program statement obeys a formal definition.
- Notion of time: every variable is a function of time
  - Imperative programming
    - e.g. $x = x + y$ means $x(t+1) = x(t) + y(t)$
  - Differential equations
    - e.g. $x' = ax + c$ means $x'(t) = ax(t) + c$
- A generic technology built for math could be extended for specs
  - With some exceptions
    - e.g. records in computer science vs. complex numbers in mathematics
Proposed Solution (1)

- A controlled language for formal specifications and mathematics
- Precisely defined subset of English
- Slightly restricted grammar & dictionary
Proposed Solution(1)

- A controlled language for formal specifications and mathematics
- Precisely defined subset of English
- Slightly restricted grammar & dictionary
- Support some complex linguistic features:
  - Basic anaphoric resolution e.g. it, they
  - References e.g. the last statement
  - Collective vs. distributive readings e.g. positive vs. equal
Proposed Solution(1)

- A controlled language for formal specifications and mathematics
- Precisely defined subset of English
- Slightly restricted grammar & dictionary
- Support some complex linguistic features:
  - Basic anaphoric resolution e.g. it, they
  - References e.g. the last statement
  - Collective vs. distributive readings e.g. positive vs. equal
- Automatic translation into a formal language such as first-order logic
- Human acceptable & machine understandable
- Combines natural language with formal methods
- Easier to read than a formal language
Proposed Solution(2)

Easier to write?

- No, writer may go out of grammar very quickly
Proposed Solution (2)

Easier to write?

- No, writer may go out of grammar very quickly

But may be possible if:

- Controlled language is NOT so restricted
- Reasonably big enough
- Tools such as word completion are available
**Proposed Solution (2)**

<table>
<thead>
<tr>
<th>Easier to write?</th>
</tr>
</thead>
<tbody>
<tr>
<td>No, writer may go out of grammar very quickly</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>But may be possible if:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controlled language is <strong>NOT</strong> so restricted</td>
</tr>
<tr>
<td>Reasonably <strong>big enough</strong></td>
</tr>
<tr>
<td>Tools such as <strong>word completion</strong> are available</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>But even it fails:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Could be the first step to implement full NL grammar</td>
</tr>
<tr>
<td>Problems: complexity issues</td>
</tr>
</tbody>
</table>
What this work is **NOT** about

Parsing existing text (RFCs, Protocols, Math textbooks)
What this work is NOT about
Parsing existing text (RFCs, Protocols, Math textbooks)

What this work is ABOUT
- Distinct proofs from different parts of maths – quite mature
- Some case studies from formal specifications – in progress
What this work is NOT about

Parsing existing text (RFCs, Protocols, Math textbooks)

What this work is ABOUT

- Distinct proofs from different parts of maths – quite mature
- Some case studies from formal specifications – in progress

In this talk

- A Proof from elementary number theory – working prototype
- Formal Specifications of a simple scheduler – in progress
- Mostly linguistic issues
Outline

1. Motivation
2. Examples
3. Implementation details
An Appetiser (1), In progress . . .

Definition.

A scheduling algorithm $S$ is non starving if

1. $s$ is a scheduler state.
2. $i$ is a process id in some queue of $s$.
3. For any $ts$, a sequence of termination state, we define the sequence $ss$ by
   1. $ss_0 = s$
   2. $ss_{n+1} = S(ts(n), ss_n)$

Then there exists $p$ such that the `running_process` of $ss_p$ is $i$. 
Definition 1. A process id is a natural number.
Definition 2. A queue is a list of process ids.
Definition 3. A scheduler state is a tuple
\((\text{queues}, \text{counter}, \text{running\_process}, \text{running\_queue\_no})\)
where
1. \text{queues} is list of a queue of size 3.
2. \text{counter} is a natural number less than 6.
3. \text{running\_process} is a process id.
4. \text{running\_queue\_no} is a natural number less than 3.

Definition 4. A scheduler state is correct if the head of its \text{queues} numbered \text{running\_queue\_no} is equal to its \text{running\_process}.
Definition 5. A termination state is an integer which is either TERMINATED, PREEMPTED or YIELD.
Definition 6. A scheduler state is empty if all the queues of its \text{queues} are empty.
Definition 7. A scheduling algorithm is a function taking a correct scheduler state and a termination state, returning a correct or empty scheduler state.
Theorem.
If $x$ and $y$ are two even integers then $x + y$ is even.

Proof.
Suppose that $x$ and $y$ are two even integers.
By the definition of even numbers, $x + y = 2a + 2b$ holds.
We deduce that $x + y = 2(a + b)$ by the last statement.
Thus $x + y$ is an even integer because it is a multiple of 2.
Overall picture of the project

Controlled language

MathAbs

First order Formulas
Theorem Prover 1
Theorem Prover n
Automatic formalisation in three steps

Syntax → Semantics → Verification
Sentence level grammar (without context)

- Natural language + formal expressions & notations
- Type theoretic controlled grammar in Grammatical Framework
- Programming language to define NL grammars
- Our grammar: mostly BNF + dependant records

... 

- We deduce that $x + y = 2(a + b)$ by the last statement.
- Thus $x + y$ is an even integer because it is a multiple of 2.
1. Syntax

Example: \( x \) and \( y \) are two even integers.

**Abstract Syntax**

Syntactic structure/tree

\[
\text{cat Prop, Type, Property, Quant} \\
\text{fun} \\
MkProp : \text{Subj} \rightarrow \text{Quant} \rightarrow [\text{Property}] \rightarrow \text{Type} \rightarrow \text{Prop} \\
\text{Two : Quant} \\
\text{Even : Property} \\
\text{Integer : Type}
\]

We deduce that \( x + y = 2(a + b) \) by the last statement.

Thus \( x + y \) is an even integer because it is a multiple of \( 2 \).

**Concrete Syntax**

Linearization rules to each function

\[
\text{Number} = \text{Sg} \mid \text{Pl} \\
\text{Type} = \{s : \text{Number} \Rightarrow \text{Str}\} \\
\text{Integer} = \text{table } \{\text{Sg} \Rightarrow \text{“integer”} ; \text{Pl} \Rightarrow \text{“integers”}\} \\
\text{MkProp subj quant props type} = \text{subj.s} \text{ ++ be.s!subj.n} \text{ ++ quant.s} \text{ ++ props.s} \text{ ++ type.s!subj.n}
\]

\[
\text{Property, Quant, Prop} = \{s : \text{Str}\} \\
\text{Two} = \text{“two”} \\
\text{Even} = \text{“even”}
\]
1. Syntax

Modular structure of the grammar allows to **rephrase** sentences

- We deduce that $x + y = 2(a + b)$ by the last statement.
- By the last statement, we deduce that $x + y = 2(a + b)$.
- By the last statement, $x + y = 2(a + b)$ holds.
- By the last statement, $x + y = 2(a + b)$.
- $x + y = 2(a + b)$ by the last statement.
Automatic formalisation in three steps

Syntax → Semantics → Verification
2. Semantics

- Building discourse

**Theorem.** If \( x \) and \( y \) are two even integers then \( x + y \) is even.

**Proof.** Suppose that \( x \) and \( y \) are two even integers. By the definition of even numbers, \( x + y = 2a + 2b \) holds. We deduce that \( x + y = 2(a + b) \) by the last statement. Thus \( x + y \) is an even integer because it is a multiple of 2.

<table>
<thead>
<tr>
<th>Variables</th>
<th>Type</th>
<th>Gender</th>
<th>Number</th>
<th>How Decl</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>(NoType)</td>
<td>Neut</td>
<td>Sg</td>
<td>DeclMan</td>
</tr>
<tr>
<td>it</td>
<td>(?)</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>( x + y )</td>
<td>(Int)</td>
<td>Neut</td>
<td>Sg</td>
<td>DeclMan</td>
</tr>
<tr>
<td>( x, y )</td>
<td>(Int)</td>
<td>Neut</td>
<td>Pl</td>
<td>DeclMan</td>
</tr>
<tr>
<td>( a, b )</td>
<td>(NoType)</td>
<td>Neut</td>
<td>Pl</td>
<td>DeclAuto</td>
</tr>
<tr>
<td>( x, y )</td>
<td>(Int)</td>
<td>Neut</td>
<td>Pl</td>
<td>DeclMan</td>
</tr>
<tr>
<td>( x, y )</td>
<td>(Int)</td>
<td>Neut</td>
<td>Pl</td>
<td>DeclMan</td>
</tr>
<tr>
<td>( x + y )</td>
<td>(NoType)</td>
<td>Neut</td>
<td>Sg</td>
<td>Prv</td>
</tr>
<tr>
<td>( x, y )</td>
<td>(Int)</td>
<td>Neut</td>
<td>Pl</td>
<td>Prv</td>
</tr>
</tbody>
</table>
2. Semantics

- Solving basic anaphora e.g. It and They

**Theorem.** If $x$ and $y$ are two even integers then $x + y$ is even.

**Proof.** Suppose that $x$ and $y$ are two even integers. By the definition of even numbers, $x + y = 2a + 2b$ holds. We deduce that $x + y = 2(a + b)$ by the last statement. Thus $x + y$ is an even integer because it is a multiple of 2.

<table>
<thead>
<tr>
<th>Variables</th>
<th>Type</th>
<th>Gender</th>
<th>Number</th>
<th>How Decl</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>(NoType)</td>
<td>Neut</td>
<td>Sg</td>
<td>DeclMan</td>
</tr>
<tr>
<td>$x + y$</td>
<td>(Int)</td>
<td>Neut</td>
<td>Sg</td>
<td>DeclMan</td>
</tr>
<tr>
<td>$x + y$</td>
<td>(Int)</td>
<td>Neut</td>
<td>Sg</td>
<td>DeclMan</td>
</tr>
<tr>
<td>$x, y$</td>
<td>(Int)</td>
<td>Neut</td>
<td>Pl</td>
<td>DeclMan</td>
</tr>
<tr>
<td>$a, b$</td>
<td>(NoType)</td>
<td>Neut</td>
<td>Pl</td>
<td>DeclAuto</td>
</tr>
<tr>
<td>$x, y$</td>
<td>(Int)</td>
<td>Neut</td>
<td>Pl</td>
<td>DeclMan</td>
</tr>
<tr>
<td>$x, y$</td>
<td>(Int)</td>
<td>Neut</td>
<td>Pl</td>
<td>DeclMan</td>
</tr>
<tr>
<td>$x + y$</td>
<td>(NoType)</td>
<td>Neut</td>
<td>Sg</td>
<td>Prv</td>
</tr>
<tr>
<td>$x, y$</td>
<td>(Int)</td>
<td>Neut</td>
<td>Pl</td>
<td>Prv</td>
</tr>
</tbody>
</table>
2. Semantics

- Solving basic anaphora e.g. references

**Theorem.** If $x$ and $y$ are two even integers then $x + y$ is even.

**Proof.** Suppose that $x$ and $y$ are two even integers. By the definition of even numbers, $x + y = 2a + 2b$ holds. We deduce that $x + y = 2(a + b)$ by the last statement. Thus $x + y$ is an even integer because it is a multiple of 2.

<table>
<thead>
<tr>
<th>Formulas</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x + y \in \mathbb{Z} &amp; \text{even } (x + y)$</td>
<td>Deduction</td>
</tr>
<tr>
<td>$x + y = 2(a + b)$</td>
<td>Deduction</td>
</tr>
<tr>
<td>$x + y = 2a + 2b$</td>
<td>Deduction</td>
</tr>
<tr>
<td>$x, y \in \mathbb{Z} &amp; (\text{even } (x) &amp; \text{even } (y))$</td>
<td>Hypothesis</td>
</tr>
<tr>
<td>$\forall_{x,y}(x, y \in \mathbb{Z} &amp; \text{even } (x) &amp; \text{even } (y) \Rightarrow \text{even } (x + y))$</td>
<td>Goal</td>
</tr>
</tbody>
</table>
2. Semantics

- Distributive vs. Collective readings
  e.g. $x$ and $y$ are positive vs. $x$ and $y$ are equal
2. Semantics

- Distributive vs. Collective readings
  e.g. $x$ and $y$ are positive vs. $x$ and $y$ are equal
  positive($x$) & positive($y$) vs. equal($x$,$y$)
2. Semantics

- Distributive vs. Collective readings
  - e.g. $x$ and $y$ are positive vs. $x$ and $y$ are equal
  - $\text{positive}(x) \& \text{positive}(y)$ vs. $\text{equal}(x,y)$

- Translation of controlled language to an abstract mathematical language $\text{MathAbs}$
Automatic formalisation in three steps

Syntax

Semantics

Verification
3. Verification

- MathAbs to first order formulas in progress
3. Verification

- MathAbs to first order formulas **in progress** . . .
- MathAbs to a prover specific formalism **after PhD**
  - Logical types and linguistic types are not the same
  - Need a theorem prover that could deals with types as predicates
  - e.g. PML\(^a\)

\(^a\)http://www.lama.univ-savoie.fr/tracpml
Specifications and Math text are similar problems
A controlled language could work for both
A long term working project, but feasible
Questions