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

Outline

Problem 1

- Specifications & standards written in natural languages (RFCs, ISO, ANSI, patents etc)
- Often incomplete or imprecise[1]
- 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

V. Paxson. Automated packet trace analysis of TCP implementations. In Proc. SIGCOMM 97.

Problem 1

- Specifications & standards written in natural languages (RFCs, ISO, ANSI, patents etc)
- Often incomplete or imprecise[1]
- 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

V. Paxson. Automated packet trace analysis of TCP implementations. In Proc. SIGCOMM 97.

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?

Math & Specifications: 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 ... 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 ...
- For all x, if x is an even integer then ...
- There exists p such that the running process of ss_p 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)

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

But even it fails:

- Could be the first step to implement full NL grammar
- Problems: complexity issues

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.

Supporting Definitions

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
$(queues, counter, running\_process, running\_queue\_no)$ where

1. *queues* is list of a queue of size 3.
2. *counter* is a natural number less than 6.
3. *running_process* is a process id.
4. *running_queue_no* is a natural number less than 3.

Definition 4. A scheduler state is correct if the head of its *queues* numbered *running_queue_no* is equal to its *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 *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.

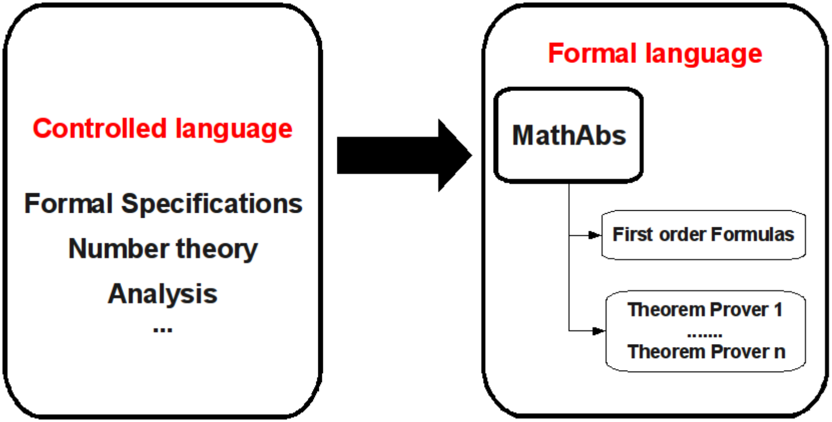
An Appetiser (2)

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.

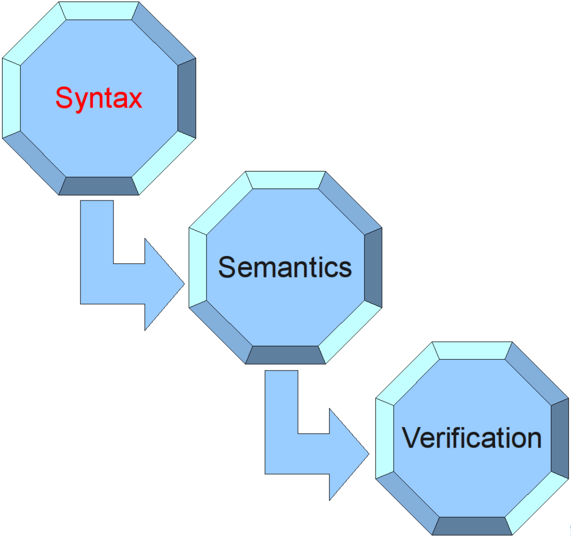
Outline

1. Motivation
2. Examples
3. Implementation details

Overall picture of the project



Automatic formalisation in three steps



1. Syntax

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

cat Prop, Type, Property, Quant
fun
MkProp : Subj → Quant → [Property] → Type → Prop
Two : Quant
Even : Property
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

Number = Sg | Pl
Type = {s : Number => Str}
Integer = table {Sg => "integer" ; Pl => "integers" }

Property, Quant, Prop = {s : Str}
Two = "two"
Even = "even"

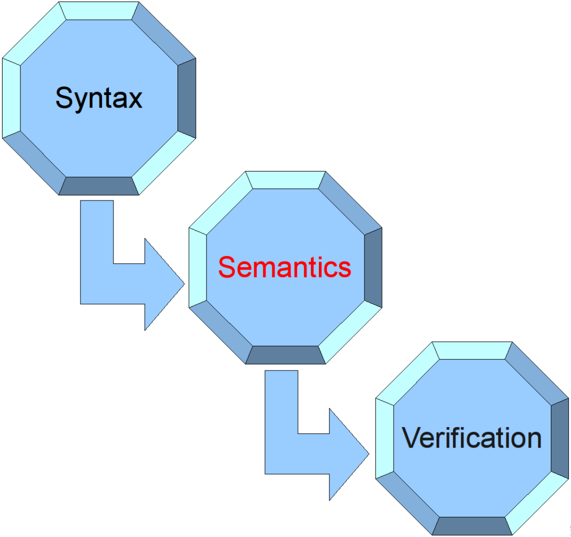
MkProp subj quant props type = subj.s ++ be.s!subj.n ++ quant.s ++ props.s ++ type.s!subj.n

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



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.

Variables	Type	Gender	Number	How Decl
2	(NoType	Neut	Sg	DeclMan)
it	(?	?	?	?)
$x+y$	(Int	Neut	Sg	DeclMan)
x, y	(Int	Neut	Pl	DeclMan)
a, b	(NoType	Neut	Pl	DeclAuto)
x, y	(Int	Neut	Pl	DeclMan)
x, y	(Int	Neut	Pl	DeclMan)
$x+y$	(NoType	Neut	Sg	Prv)
x, y	(Int	Neut	Pl	Prv)

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.

Variables	Type	Gender	Number	How Decl
2	(NoType	Neut	Sg	DeclMan)
$x + y$	(Int	Neut	Sg	DeclMan)
$x + y$	(Int	Neut	Sg	DeclMan)
x, y	(Int	Neut	Pl	DeclMan)
a, b	(NoType	Neut	Pl	DeclAuto)
x, y	(Int	Neut	Pl	DeclMan)
x, y	(Int	Neut	Pl	DeclMan)
$x + y$	(NoType	Neut	Sg	Prv)
x, y	(Int	Neut	Pl	Prv)

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.

Formulas	Type
$x + y \in Z$ & even $(x + y)$	Deduction
$x + y = 2(a + b)$	Deduction
$x + y = 2a + 2b$	Deduction
$x, y \in Z$ & (even (x) & even (y))	Hypothesis
$\forall_{x,y}(x, y \in Z$ & even (x) & even $(y) \Rightarrow$ even $(x + y))$	Goal

2. Semantics

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

2. Semantics

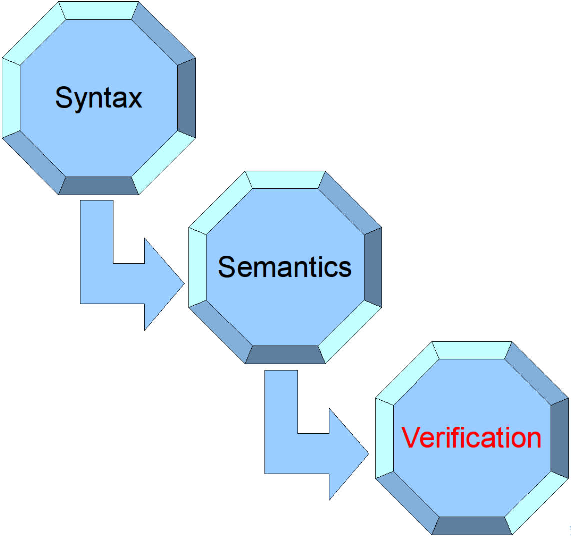
- Distributive vs. Collective readings
  e.g. *x* and *y* are positive vs. *x* and *y* are equal
  positive(x) & positive(y) vs. equal(x,y)

2. Semantics

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

- Translation of controlled language to an abstact mathematical langauge MathAbs

Automatic formalisation in three steps



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

Summary

- Specifications and Math text are similar problems
- A controlled language could work for both
- A long term working project, but feasible

Questions

?