Software specifications and Mathematical Proofs in Natural Languages

Muhammad Humayoun

Laboratoire de Mathématiques (LAMA)
Université de Savoie
mhuma@univ-savoie.fr

Starting date: Feb 2007

Co-Supervised by

Christophe Raffalli
Asistant Professor
Laboratoire de Mathématiques (LAMA)
Université de Savoie
France

Aarne Ranta
Professor
Department of Computer Science
Chalmers University of Technology
Sweden
Outline

1. Introduction

2. Mathematical Proofs and specifications
   - Software Specifications
   - The Mathematical Proofs

3. Implementation Details

4. Future Work
An attempt to make a *connection* between natural and formal languages

Aim: to develop resources capable of translating between natural & formal languages

The main goal
- Finding a subset of NL to write Proofs and specifications interactively
- Grammatically correct
- Large enough

We consider:

1. **Parsing** and *translation* of Mathematical Proofs
   - Grammar
   - Lexicon
     - *type-theoretic* containing constants, types, & definitions
   - Ongoing; First prototype: Jan 2008

2. **Checking** interactively, for the abstract syntaxes of both formal & natural Software specifications
An attempt to make a *connection* between natural and formal languages

Aim: to develop resources capable of translating between natural & formal languages

**The main goal**
- Finding a subset of NL to write Proofs and specifications interactively
- Grammatically correct
- large enough

We consider:

*Parsing* and *translation* of Mathematical Proofs

- Grammar
- Lexicon
  - type-theoretic containing constants, types, & definitions
- Ongoing; First prototype: Jan 2008

*Checking* interactively, for the abstract syntaxes of both formal & natural Software specifications
An attempt to make a connection between natural and formal languages

Aim: to develop resources capable of translating between natural & formal languages

The main goal
- Finding a subset of NL to write Proofs and specifications interactively
- Grammatically correct
- large enough

We consider:

1. Parsing and translation of Mathematical Proofs
- Grammar
- Lexicon
  - type-theoretic containing constants, types, & definitions
- Ongoing; First prototype: Jan 2008

2. Checking interactively, for the abstract syntaxes of both formal & natural Software specifications
An attempt to make a connection between natural and formal languages

Aim: to develop resources capable of translating between natural & formal languages

The main goal
- Finding a subset of NL to write Proofs and specifications interactively
- Grammatically correct
- Large enough

We consider:

1. Parsing and translation of Mathematical Proofs
   - Grammar
   - Lexicon
     type-theoretic containing constants, types, & definitions
   - Ongoing; First prototype: Jan 2008

2. Checking interactively, for the abstract syntaxes of both formal & natural Software specifications
Introduction

- An attempt to make a *connection* between natural and formal languages
- Aim: to develop resources capable of translating between natural & formal languages
- The main goal
  - Finding a subset of NL to write Proofs and specifications interactively
  - Grammatically correct
  - large enough

We consider:

1. *Parsing* and *translation* of Mathematical Proofs
   - Grammar
   - Lexicon
     - type-theoretic containing constants, types, & definitions
   - **Ongoing**; First prototype: Jan 2008

2. **Checking** interactively, for the abstract syntaxes of both formal & natural Software specifications
An attempt to make a *connection* between natural and formal languages

Aim: to develop resources capable of translating between natural & formal languages

The main goal

- Finding a subset of NL to write Proofs and specifications interactively
- Grammatically correct
- Large enough

We consider:

1. *Parsing* and *translation* of Mathematical Proofs
   - Grammar
   - Lexicon
     - type-theoretic containing constants, types, & definitions
   - Ongoing; First prototype: Jan 2008

2. *Checking* interactively, for the abstract syntaxes of both formal & natural software specifications
Some terminology

NL = Natural Language
FL = Formal Language
NLP = Natural language processing

Anaphora = reference back to the text. e.g. the man smashes the fly. He was annoyed.

- **Generation** ⇒ (FL → NL) **trivial generation - bad text**
  - *Canned text*
    - you have 1 new message(s)
    - you have 100 new message(s)
  - **Another Example:**
    - an\(^1\) exception is not thrown and the AID is equal to the subsequence from offset plus 1 to offset plus length of bArray or a transaction exception is thrown and the reason of the system instance of TransactionException is equal to the BUFFER-FULL reason of TransactionException
Some terminology

NL = Natural Language
FL = Formal Language
NLP = Natural language processing

Anaphora = reference back to the text. e.g. the man smashes the fly. He was annoyed.

- **Generation** $\Rightarrow$ (FL $\rightarrow$ NL) trivial generation - bad text
  - Canned text
    - you have 1 new message(s)
    - you have 100 new message(s)
  - Another Example:
    - an\(^1\) exception is not thrown and the AID is equal to the subsequence from offset plus 1 to offset plus length of bArray or a transaction exception is thrown and the reason of the system instance of TransactionException is equal to the BUFFER-FULL reason of TransactionException

---

\(^1\)"Improving the natural language translation of formal specifications", David A. Bruke, Page 63
Some terminology

NL = Natural Language
FL = Formal Language
NLP = Natural language processing

Anaphora = reference back to the text. e.g. the man smashes the fly. He was annoyed.

- **Generation** ⇒ (FL → NL) trivial generation - bad text
  - **Canned text**
    - you have 1 new message(s)
    - you have 100 new message(s)
  - **Another Example:**
    - an\(^1\) exception is not thrown and the AID is equal to the subsequence from offset plus 1 to offset plus length of bArray or a transaction exception is thrown and the reason of the system instance of TransactionException is equal to the BUFFER-FULL reason of TransactionException

\(^1\)"Improving the natural language translation of formal specifications", David A. Bruke, Page 63
Some terminology

NL = Natural Language  
FL = Formal Language  
NLP = Natural language processing  
Anaphora = reference back to the text. e.g. the man smashes the fly. He was annoyed.

- **Generation** ⇒ (FL → NL) trivial generation - bad text
  - Canned text
    you have 1 new message(s)
you have 100 new message(s)
  - Another Example:
an\(^1\) exception is not thrown and the AID is equal to the subsequence from offset plus 1 to offset plus length of bArray or a transaction exception is thrown and the reason of the system instance of TransactionException is equal to the BUFFER-FULL reason of TransactionException

---

\(^1\) "Improving the natural language translation of formal specifications", David A. Bruke, Page 63
Some terminology

NL = Natural Language
FL = Formal Language
NLP = Natural language processing
Anaphora = reference back to the text. e.g. the man smashes the fly. He was annoyed.

- **Generation ⇒ (FL → NL)** trivial generation - bad text
  - **Canned text**
    - you have 1 new message(s)
    - you have 100 new message(s)
  - **Another Example:**
    - An exception is not thrown and the AID is equal to the subsequence from offset plus 1 to offset plus length of bArray or a transaction exception is thrown and the reason of the system instance of TransactionException is equal to the BUFFER-FULL reason of TransactionException

---

1"Improving the natural language translation of formal specifications", David A. Bruke, Page 63
Some terminology

NL = Natural Language
FL = Formal Language
NLP = Natural language processing

Anaphora = reference back to the text. e.g. the man smashes the fly. He was annoyed.

- **Generation** ⇒ (FL → NL) trivial generation -Bad Text
- **Checking** ⇒ (NL → FL → Boolean) less-ambiguous
- **Parsing** ⇒ (NL → FL) ambiguity
- **Translation** ⇒ (NL → NL) ambiguity, loss of info
- **Generation** ⇒ (FL → NL) very hard

Text in standards such as RFCs, ISO, ANSI, patents

an exception is not thrown and the AID is equal to the subsequence from offset plus 1 to offset plus length of bArray...

- What is the suitable division between literal and non literal(semantic) translation?
- What should be translated and what should not (formulas, …)?
- Complexity grows with the size of statement
Some terminology

NL = Natural Language
FL = Formal Language
NLP = Natural language processing

Anaphora = reference back to the text. e.g. the man smashes the fly. He was annoyed.

- Generation ⇒ (FL → NL) trivial generation -Bad Text
- Checking ⇒ (NL → FL → Boolean) less-ambiguous
- Parsing ⇒ (NL → FL) ambiguity
- Translation ⇒ (NL → NL) ambiguity, loss of info
- Generation ⇒ (FL → NL) very hard

Text in standards such as RFCs, ISO, ANSI, patents
an exception is not thrown and the AID is equal to the subsequence from offset plus 1 to offset plus length of bArray...

- What is the suitable division between literal and non literal(semantic) translation?
- What should be translated and what should not (formulas, …)?
- Complexity grows with the size of statement
Some terminology

NL = Natural Language
FL = Formal Language
NLP = Natural language processing

Anaphora = reference back to the text. e.g. the man smashes the fly. He was annoyed.

- Generation ⇒ (FL → NL) trivial generation -Bad Text
- Checking ⇒ (NL → FL → Boolean) less-ambiguous
- Parsing ⇒ (NL → FL) ambiguity
- Translation ⇒ (NL → NL) ambiguity, loss of info
- Generation ⇒ (FL → NL) very hard

Text in standards such as RFCs, ISO, ANSI, patents

an exception is not thrown and the AID is equal to the subsequence from offset plus 1 to offset plus length of bArray...

- What is the suitable division between literal and non literal(semantic) translation?
- What should be translated and what should not (formulas, ...)?
- Complexity grows with the size of statement
Some terminology

NL = Natural Language
FL = Formal Language
NLP = Natural language processing

Anaphora = reference back to the text. e.g. the man smashes the fly. He was annoyed.

- Generation ⇒ (FL → NL) trivial generation - Bad Text
- Checking ⇒ (NL → FL → Boolean) less-ambiguous
- Parsing ⇒ (NL → FL) ambiguity
- Translation ⇒ (NL → NL) ambiguity, loss of info
- Generation ⇒ (FL → NL) very hard

Text in standards such as RFCs, ISO, ANSI, patents
an exception is not thrown and the AID is equal to the subsequence from offset plus 1 to offset plus length of bArray...

- What is the suitable division between literal and non literal (semantic) translation?
- What should be translated and what should not (formulas, …)?
- Complexity grows with the size of statement
Some terminology

NL = Natural Language
FL = Formal Language
NLP = Natural language processing

Anaphora = reference back to the text. e.g. the man smashes the fly. He was annoyed.

• Generation ⇒ (FL → NL) trivial generation -Bad Text
• Checking ⇒ (NL → FL → Boolean) less-ambiguous
• Parsing ⇒ (NL → FL) ambiguity
• Translation ⇒ (NL → NL) ambiguity, loss of info

Generation ⇒ (FL → NL) very hard

Text in standards such as RFCs, ISO, ANSI, patents

an exception is not thrown and the AID is equal to the subsequence from offset plus 1 to offset plus length of bArray...

• What is the suitable division between literal and non literal (semantic) translation?
• What should be translated and what should not (formulas, …)?
• Complexity grows with the size of statement
Some terminology

NL = Natural Language
FL = Formal Language
NLP = Natural language processing
Anaphora = reference back to the text. e.g. the man smashes the fly. He was annoyed.

- **Generation** ⇒ (FL → NL) trivial generation -Bad Text
- **Checking** ⇒ (NL → FL → Boolean) less-ambiguous
- **Parsing** ⇒ (NL → FL) ambiguity
- **Translation** ⇒ (NL → NL) ambiguity, loss of info

- Generation ⇒ (FL → NL) very hard

Text in standards such as RFCs, ISO, ANSI, patents

an exception is not thrown and the AID is equal to the subsequence from offset plus 1 to offset plus length of bArray...

- What is the suitable division between literal and non literal(semantic) translation?
- What should be translated and what should not (formulas, …)?
- Complexity grows with the size of statement
Some terminology

NL = Natural Language
FL = Formal Language
NLP = Natural language processing

Anaphora = reference back to the text. e.g. the man smashes the fly. He was annoyed.

- Generation ⇒ (FL → NL) trivial generation -Bad Text
- Checking ⇒ (NL → FL → Boolean) less-ambiguous
- Parsing ⇒ (NL → FL) ambiguity
- Translation ⇒ (NL → NL) ambiguity, loss of info
- Generation ⇒ (FL → NL) very hard

Text in standards such as RFCs, ISO, ANSI, patents

an exception is not thrown and the AID is equal to the subsequence from offset plus 1 to offset plus length of bArray...

- What is the suitable division between literal and non literal(semantic) translation?
- What should be translated and what should not (formulas, …)?
- Complexity grows with the size of statement
Some terminology

NL = Natural Language
FL = Formal Language
NLP = Natural language processing

Anaphora = reference back to the text. e.g. the man smashes the fly. He was annoyed.

- Generation $\Rightarrow$ (FL $\rightarrow$ NL) trivial generation -Bad Text
- Checking $\Rightarrow$ (NL $\rightarrow$ FL $\rightarrow$ Boolean) less-ambiguous
- Parsing $\Rightarrow$ (NL $\rightarrow$ FL) ambiguity
- Translation $\Rightarrow$ (NL $\rightarrow$ NL) ambiguity, loss of info
- Generation $\Rightarrow$ (FL $\rightarrow$ NL) very hard

Text in standards such as RFCs, ISO, ANSI, patents

an exception is not thrown and the AID is equal to the subsequence from offset plus 1 to offset plus length of bArray...

- What is the suitable division between literal and non literal (semantic) translation?
- What should be translated and what should not (formulas, . . . )?
- Complexity grows with the size of statement
1 Introduction

2 Mathematical Proofs and specifications
   - Software Specifications
   - The Mathematical Proofs

3 Implementation Details

4 Future Work
Why NL tools for Software Specifications?

- Specifications & Standards written in natural language (RFCs, ISO, ANSI, patents etc)
- The difficulties of software designers & engineers with Mathematical formalism
- Usefulness:
  - Formal methods
  - Human computer interaction
  - Natural language technology
Outline

1 Introduction

2 Mathematical Proofs and specifications
   - Software Specifications
   - The Mathematical Proofs

3 Implementation Details

4 Future Work
A large portion of mathematical text consists of proofs

Mathematical Proofs:
combination of natural language and mathematical formulas (domains/theories)

Therefore, a clear distinction between:
- Mathematical Proofs
- Other domain/Theories

Domains in our consideration
- Logic & Arithmetic (also needed for specifications)
- Formal Specifications
A large portion of mathematical text consists of proofs

Mathematical Proofs:
combination of natural language and mathematical formulas (domains/theories)

Therefore, a clear distinction between:
- Mathematical Proofs
- Other domain/Theories

Domains in our consideration
- Logic & Arithmetic (also needed for specifications)
- Formal Specifications
A large portion of mathematical text consists of proofs

Mathematical Proofs: combination of natural language and mathematical formulas (domains/theories)

Therefore, a clear distinction between:
- Mathematical Proofs
- Other domain/Theories

Domains in our consideration
- Logic & Arithmetic (also needed for specifications)
- Formal Specifications
A large portion of mathematical text consists of proofs

Mathematical Proofs:
combination of natural language and mathematical formulas (domains/theories)

Therefore, a clear distinction between:
- Mathematical Proofs
- Other domain/Theories

Domains in our consideration
- Logic & Arithmetic (also needed for specifications)
- Formal Specifications
Why considering formal Specifications as a domain/theory of Mathematics?

- NLP for both mathematical proofs and formal software specifications is quite similar and pose very similar issues.
- The language of proof is often used to write statements.
- Resources for mathematical proofs: A prerequisite and reusable for specification.
Why considering formal Specifications as a domain/theory of Mathematics?

- NLP for both mathematical proofs and formal software specifications is quite similar and pose very similar issues.
- The language of proof is often used to write statements.
- Resources for mathematical proofs: A prerequisite and reusable for specification.
Why considering formal Specifications as a domain/theory of Mathematics?

- NLP for both mathematical proofs and formal software specifications is quite similar and pose very similar issues.
- The language of proof is often used to write statements.
- Resources for mathematical proofs: A prerequisite and reusable for specification.
• **Starting point**: *Natural language parser* independent of any domain/theory
  
  Parse proof line by line, push it into a data structure/Abstract syntax
  
  Formalism from other domains considered as strings at this stage
  
  Very similar to the *Natural deduction* style of proving
  
  Rules: inside the proof
  
  First prototype: only dealing with English
Starting point: *Natural language parser* independent of any domain/theory

Parse proof line by line, push it into a *data structure/Abstract syntax*

Formalism from other domains considered as strings at this stage

Very similar to the *Natural deduction* style of proving

Rules: inside the proof

First prototype: only dealing with English
- **Starting point**: *Natural language parser* independent of any domain/theory
- Parse proof line by line, push it into a data structure/Abstract syntax
- Formalism from *other domains* considered as strings at this stage
- Very similar to the *Natural deduction* style of proving
- Rules: inside the proof
- First prototype: only dealing with English
• Starting point: *Natural language parser* independent of any domain/theory

• Parse proof line by line, push it into a *data structure/Abstract syntax*

• Formalism from *other domains* considered as strings at this stage

• Very similar to the *Natural deduction* style of proving

• Rules: inside the proof

• First prototype: only dealing with English
Starting point: *Natural language parser* independent of any domain/theory

Parse proof line by line, push it into a *data structure/Abstract syntax*

Formalism from *other domains* considered as strings at this stage

Very similar to the *Natural deduction* style of proving

Rules: inside the proof

First prototype: only dealing with English
Starting point: *Natural language parser* independent of any domain/theory

Parse proof line by line, push it into a *data structure/Abstract syntax*

Formalism from *other domains* considered as strings at this stage

Very similar to the *Natural deduction* style of proving

Rules: inside the proof

First prototype: only dealing with English
Mathematics:
Prove that the sum of two even integers is always even.

Proof:

- Consider two even integers $x$ and $y$. We have to prove that their sum is even.
- They can be written as $x = 2a$ and $y = 2b$ respectively for smaller integers $a$ and $b$.
- Then the sum $x + y = 2a + 2b = 2(a + b)$.
- From this, it is clear that 2 is a factor of $x + y$, so the sum of two even integers is always even.
Specifications:

- In every intermediate or end system, the following relationship must hold for these parameters for all network interfaces. The symbol "\( \geq \)" is interpreted relative to the linear ordering defined for security levels specified in Section 2.3 for the "LEVEL" parameters, and as set inclusion for the "AUTHORITY" parameters.
  \[
  \text{SYSTEM-LEVEL-MAX} \geq \text{PORT-LEVEL-MAX} \geq \text{PORT-LEVEL-MIN} \geq \text{SYSTEM-LEVEL-MIN}
  \]

- When an internet module routes a datagram it checks to see if the record route option is present. If it is, it inserts its own internet address as known in the environment into which this datagram is being forwarded into the recorded route beginning at the octet indicated by the pointer, and increments the pointer by four.

---

The data structure for Proofs in BNF notation

\[ \langle \text{Justification} \rangle ::= \langle \text{String} \rangle \]
\[ \langle \text{Formula} \rangle ::= \langle \text{String} \rangle \]
\[ \langle \text{Local-Justification} \rangle ::= \epsilon \mid \langle \text{String} \rangle \]
\[ \langle \text{Var} \rangle ::= \langle \text{String} \rangle \]
\[ \langle \text{ListRule} \rangle ::= \epsilon \mid \langle \text{Rule} \rangle \langle \text{ListRule} \rangle \]

\[ \langle \text{Proof} \rangle ::= \text{trivial} \langle \text{Justification} \rangle \]
\[ \mid \langle \text{Rule} \rangle \langle \text{Justification} \rangle \]

\[ \langle \text{Rule} \rangle ::= \text{assume} \langle \text{Formula} \rangle \langle \text{Local-Justification} \rangle \langle \text{Rule} \rangle \]
\[ \mid \text{let} \langle \text{Var} \rangle \langle \text{Rule} \rangle \]
\[ \mid \text{split} \langle \text{ListRule} \rangle \]
\[ \mid \text{show} \langle \text{Formula} \rangle \langle \text{Proof} \rangle \]
\[ \mid \text{continue} \langle \text{Proof} \rangle \]
The data structure for Proofs in BNF notation

\[
\begin{align*}
\langle \text{Justification} \rangle & ::= \langle \text{String} \rangle \\
\langle \text{Formula} \rangle & ::= \langle \text{String} \rangle \\
\langle \text{Local-Justification} \rangle & ::= \epsilon \mid \langle \text{String} \rangle \\
\langle \text{Var} \rangle & ::= \langle \text{String} \rangle \\
\langle \text{ListRule} \rangle & ::= \epsilon \mid \langle \text{Rule} \rangle \langle \text{ListRule} \rangle \\
\langle \text{Proof} \rangle & ::= \text{trivial} \langle \text{Justification} \rangle \\
& \mid \langle \text{Rule} \rangle \langle \text{Justification} \rangle \\
\langle \text{Rule} \rangle & ::= \text{assume} \langle \text{Formula} \rangle \langle \text{Local-Justification} \rangle \langle \text{Rule} \rangle \\
& \mid \text{let} \langle \text{Var} \rangle \langle \text{Rule} \rangle \\
& \mid \text{split} \langle \text{ListRule} \rangle \\
& \mid \text{show} \langle \text{Formula} \rangle \langle \text{Proof} \rangle \\
& \mid \text{continue} \langle \text{Proof} \rangle
\end{align*}
\]
The data structure for Proofs in BNF notation

\[
\begin{align*}
\langle \text{Justification} \rangle & ::= \langle \text{String} \rangle \\
\langle \text{Formula} \rangle & ::= \langle \text{String} \rangle \\
\langle \text{Local-Justification} \rangle & ::= \epsilon \quad | \quad \langle \text{String} \rangle \\
\langle \text{Var} \rangle & ::= \langle \text{String} \rangle \\
\langle \text{ListRule} \rangle & ::= \epsilon \quad | \quad \langle \text{Rule} \rangle \langle \text{ListRule} \rangle \\
\langle \text{Proof} \rangle & ::= \text{trivial} \langle \text{Justification} \rangle \\
& \quad | \quad \langle \text{Rule} \rangle \langle \text{Justification} \rangle \\
\langle \text{Rule} \rangle & ::= \text{assume} \langle \text{Formula} \rangle \langle \text{Local-Justification} \rangle \langle \text{Rule} \rangle \\
& \quad | \quad \text{let} \langle \text{Var} \rangle \langle \text{Rule} \rangle \\
& \quad | \quad \text{split} \langle \text{ListRule} \rangle \\
& \quad | \quad \text{show} \langle \text{Formula} \rangle \langle \text{Proof} \rangle \\
& \quad | \quad \text{continue} \langle \text{Proof} \rangle
\end{align*}
\]
An Example Proof

Prove that the sum of two even integers is always even.

Proof:

Consider two even integers $x$ and $y$. We have to prove that their sum is even.

They can be written as $x = 2a$ and $y = 2b$ respectively for smaller integers $a$ and $b$.

Then the sum $x + y = 2a + 2b = 2(a + b)$.

From this, it is clear that 2 is a factor of $x + y$, so the sum of two even integers is always even.

Mathematical Vernacular/Concrete Syntax

- let "x" assume "x belongs Z"
- let "y" assume "y belongs Z"
- assume "even(x)" assume "even(y)"
- show "even(x+y)" ?
  "None"

... 

- let "a,b" assume "a,b belongs Z"
- assume "x=2*a, y=2*b"
- continue ?
  "byEvenness"
- "None"
Prove that the sum of two even integers is always even.

... Then the sum $x + y = 2a + 2b = 2(a + b)$.
From this, it is clear that 2 is a factor of $x + y$, so the sum of two even integers is always even.

... assume "$x+y=2*a+2*b$"
assume "$x+y=2*(a+b)$"
continue?
"None"
"byEvenness"
"None"

... assume "$2|(x+y)$"
continue trivial
"MultiplicationRule"
"None"
"byEvenness"
"None"
Prove that the sum of two even integers is always even.

Proof:

- Consider two even integers \( x \) and \( y \). We have to prove that their sum is even.
- They can be written as \( x = 2a \) and \( y = 2b \) respectively for smaller integers \( a \) and \( b \).
- Then the sum \( x + y = 2a + 2b = 2(a + b) \).
- From this, it is clear that \( 2 \) is a factor of \( x + y \), so the sum of two even integers is always even.

Natural deduction style representation

\[
\begin{align*}
\ldots & (2 | (x + y)) \vdash \text{trivial} \\
\ldots x + y = 2 \ast a + 2 \ast b; x + y = 2 \ast (a + b); \ldots & \vdash \text{continue} \\
a, b, x, y \in \mathbb{Z}, x = 2 \ast a; y = 2 \ast b; a < x; b < y; \ldots & \vdash \text{continue} \\
x, y \in \mathbb{Z}, \text{even}(x), \text{even}(y) & \vdash \text{even}(x + y) \\
x, y \in \mathbb{Z} & \vdash \text{even}(x), \text{even}(y) \to \text{even}(x + y) \\
\vdash \forall x, y \in \mathbb{Z}, \text{even}(x), \text{even}(y) \to \text{even}(x + y)
\end{align*}
\]
• we assume that "\textit{sqrt of 2 is rational}" and obtain a contradiction.
• we assume that "\textit{sqrt of 2 is rational}" and get a contradiction.
• we assume that "\textit{x is an integer}". we show that "\textit{it is true}".
• we assume that "\textit{x is an integer}" and we show that "\textit{it is true}".
• let "\textit{x is an integer}". we have to show "\textit{some formula}" and it is trivial.
• let/suppose "\textit{x is an integer}". we have to show "\textit{some formula}" and it is trivial by "\textit{Some Hint}".
• We show that "\textit{foo}". we suppose that "\ldots" by "\textit{x}". We assume that "\ldots" by "\textit{NN}" and we suppose that "\textit{NN}" by "\textit{NN}".
• Show "\textit{foo}" it is trivial and suppose "\textit{NN}" by "\textit{x}".
Outline

1. Introduction

2. Mathematical Proofs and specifications
   - Software Specifications
   - The Mathematical Proofs

3. Implementation Details

4. Future Work
Grammatical Framework (GF) to a large extent

- Mostly developed at Chalmers University, Sweden by Aarne Ranta
- Homepage: www.cs.chalmers.se/ aarne/GF

Grammar formalism based on type theory

- Special purpose programming language

Grammar = The Abstract syntax + Concrete syntax

- Abstract syntax= semantic conditions (correct syntactic structures / trees)

- Concrete syntax= abstract syntax into strings along-with the grammatical features (and back, by reversibility)

- A framework to defining interlinguas & translations
Grammatical Framework (GF) to a large extent
- Mostly developed at Chalmers University, Sweden by Aarne Ranta
- Homepage: www.cs.chalmers.se/ aarne/GF

Grammar formalism based on type theory
- Special purpose programming language
- Grammar = The Abstract syntax + Concrete syntax
- Abstract syntax = semantic conditions (correct syntactic structures / trees)
- Concrete syntax = abstract syntax into strings along-with the grammatical features (and back, by reversibility)
- A framework to defining interlinguas & translations
Grammatical Framework (GF) to a large extent
- Mostly developed at Chalmers University, Sweden by Aarne Ranta
- Homepage: www.cs.chalmers.se/aarne/GF

Grammar formalism based on type theory
- Special purpose programming language
- Grammar = The Abstract syntax + Concrete syntax
- Abstract syntax = semantic conditions (correct syntactic structures / trees)
- Concrete syntax = abstract syntax into strings along-with the grammatical features (and back, by reversibility)
- A framework to defining interlinguas & translations
Grammatical Framework (GF) to a large extent

- Mostly developed at Chalmers University, Sweden by Aarne Ranta
- Homepage: www.cs.chalmers.se/ aarne/GF

Grammar formalism based on **type theory**

- Special purpose programming language
  - Grammar = The Abstract syntax + Concrete syntax
  - Abstract syntax = semantic conditions (correct syntactic structures / trees)
  - Concrete syntax = abstract syntax into strings along-with the grammatical features (and back, by reversibility)
  - A framework to defining interlinguas & translations
Grammatical Framework (GF) to a large extent
- Mostly developed at Chalmers University, Sweden by Aarne Ranta
- Homepage: www.cs.chalmers.se/ aarne/GF

Grammar formalism based on type theory
Special purpose programming language
Grammar = The Abstract syntax + Concrete syntax
Abstract syntax= semantic conditions (correct syntactic structures / trees)
Concrete syntax= abstract syntax into strings along-with the grammatical features (and back, by reversibility)

A framework to defining interlinguas & translations
Grammatical Framework (GF) to a large extent
- Mostly developed at Chalmers University, Sweden by Aarne Ranta
- Homepage: www.cs.chalmers.se/ aarne/GF

Grammar formalism based on type theory
Special purpose programming language
Grammar = The Abstract syntax + Concrete syntax
Abstract syntax = semantic conditions (correct syntactic structures / trees)
Concrete syntax = abstract syntax into strings along-with the grammatical features (and back, by reversibility)
A framework to defining interlinguas & translations
Implementation Details

What Framework?

- Grammatical Framework (GF) to a large extent
  - Mostly developed at Chalmers University, Sweden by Aarne Ranta
  - Homepage: www.cs.chalmers.se/ aarne/GF

- Grammar formalism based on type theory

- Special purpose programming language

- Grammar = The Abstract syntax + Concrete syntax

- Abstract syntax = semantic conditions (correct syntactic structures / trees)

- Concrete syntax = abstract syntax into strings along-with the grammatical features (and back, by reversibility)

- A framework to defining interlinguas & translations

```
Abstract
  /
 / \
Concrete1 Concrete2
```
Grammar writing

- Writing Application grammar
  - **linguistic expertise** - good knowledge of a particular language
  - **domain expertise** - good knowledge of an application domain
  - Very time consuming to write grammars from scratch
  - Expensive

- Solution:
  - Grammar reuse
  - GF Resource library: basic grammar of ten languages
  - Application grammar = Domain grammar + Resource grammar
  - Better implementation with less efforts and time
Grammar writing

- Writing Application grammar
  - linguistic expertise - good knowledge of a particular language
  - domain expertise - good knowledge of an application domain
  - Very time consuming to write grammars from scratch
  - Expensive

- Solution:
  - Grammar reuse
  - GF Resource library: basic grammar of ten languages
  - Application grammar = Domain grammar + Resource grammar
  - Better implementation with less efforts and time
Grammar writing

- Writing Application grammar
  - linguistic expertise - good knowledge of a particular language
  - domain expertise - good knowledge of an application domain
  - Very time consuming to write grammars from scratch
  - Expensive

- Solution:
  - Grammar reuse
    - GF Resource library: basic grammar of ten languages
    - Application grammar = Domain grammar + Resource grammar
    - Better implementation with less efforts and time
Grammar writing

- **Writing Application grammar**
  - **linguistic expertise** - good knowledge of a particular language
  - **domain expertise** - good knowledge of an application domain
  - Very time consuming to write grammars from scratch
  - Expensive

- **Solution:**
  - Grammar reuse
  - GF Resource library: basic grammar of ten languages
  - Application grammar = Domain grammar + Resource grammar
  - Better implementation with less efforts and time
Grammar writing

- Writing Application grammar
  - **linguistic expertise** - good knowledge of a particular language
  - **domain expertise** - good knowledge of an application domain
  - Very time consuming to write grammars from scratch
  - Expensive

- Solution:
  - Grammar reuse
  - **GF Resource library**: basic grammar of ten languages
  - Application grammar = Domain grammar + Resource grammar
  - Better implementation with less efforts and time
Grammar Engineering

Mathematics   - top level
/        \
Arithm   Specs ... - Intermediate level
\       / /
Logic Module - Base level
Abstract syntax is independent of any specific theorem prover
Outline

1. Introduction

2. Mathematical Proofs and specifications
   - Software Specifications
   - The Mathematical Proofs

3. Implementation Details

4. Future Work
Ongoing & Future Work

- Building grammar and lexicon (in progress)
  - Grammar and lexicon for parsing proofs
  - Grammar and lexicon for parsing Logic & Arithmetic
- Solving anaphoric resolution
- Introducing structural text for proofs & handling cases
- Software Specifications
  - Grammar and lexicon for NL
  - Grammar and lexicon for FL
  - Comparing abstract syntaxes to select/reject a sentence

The main goal
- To find a subset of NL good & large enough to write proofs and specifications interactively
Ongoing & Future Work

- Building grammar and lexicon (in progress)
  - Grammar and lexicon for parsing proofs
  - Grammar and lexicon for parsing Logic & Arithmetic
- Solving anaphoric resolution
  - Introducing structural text for proofs & handling cases
- Software Specifications
  - Grammar and lexicon for NL
  - Grammar and lexicon for FL
  - Comparing abstract syntaxes to select/reject a sentence

The main goal
- To find a subset of NL good & large enough to write proofs and specifications interactively
Ongoing & Future Work

- Building grammar and lexicon (in progress)
  - Grammar and lexicon for parsing proofs
  - Grammar and lexicon for parsing Logic & Arithmetic
- Solving **anaphoric resolution**
- Introducing **structural text** for proofs & handling cases

- Software Specifications
  - Grammar and lexicon for NL
  - Grammar and lexicon for FL
  - comparing abstract syntaxes to select/reject a sentence

The main goal
- To find a subset of NL **good & large enough** to write proofs and specifications interactively
Future Work

Ongoing & Future Work

- Building grammar and lexicon (in progress)
  - Grammar and lexicon for parsing proofs
  - Grammar and lexicon for parsing Logic & Arithmetic
- Solving anaphoric resolution
- Introducing structural text for proofs & handling cases

Software Specifications

- Grammar and lexicon for NL
- Grammar and lexicon for FL
- comparing abstract syntaxes to select/reject a sentence

The main goal

- To find a subset of NL good & large enough to write proofs and specifications interactively
Questions / Suggestions / Feedback?
Thanks