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**

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

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

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

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

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

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

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 $\longrightarrow$ 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 $\Rightarrow$ (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 $\Rightarrow$ (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 $\Rightarrow$ (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 $\Rightarrow$ (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 $\Rightarrow$ (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

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

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

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

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

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

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

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

Outline

1. **Introduction**
2. **Mathematical Proofs and specifications**
   - Software Specifications
   - The Mathematical Proofs
3. **Implementation Details**
4. **Future Work**

Software Specifications

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**

Mathematical Proofs and 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

Mathematical Proofs and 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

Mathematical Proofs and 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

Mathematical Proofs and 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.

The Mathematical Proofs

- 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

The Mathematical Proofs

- 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

The Mathematical Proofs

- 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

The Mathematical Proofs

- 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

The Mathematical Proofs

- 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

The Mathematical Proofs

- 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

The kind of text we want to parse

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.

The kind of text we want to parse

Specifications:

- In[2] every intermediate or end system, the following relationship must hold for these parameters for all network interfaces. The symbol ">=" 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.

  SYSTEM-LEVEL-MAX >= PORT-LEVEL-MAX >= PORT-LEVEL-MIN >= SYSTEM-LEVEL-MIN

- When[3] 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 begining at the octet indicated by the pointer, and increments the pointer by four.

[2] RFC 1108, U.S. DOD Security Option, November 1991, page 9

[3] RFC 791, Internet Protocol Specification, September 1981, page 21

The data structure for Proofs in BNF notation

$$
\begin{array}{lclcl}
\langle \mathit{Justification} \rangle & ::= & \langle \mathit{String} \rangle & & \\
\langle \mathit{Formula} \rangle & ::= & \langle \mathit{String} \rangle & & \\
\langle \mathit{Local\text{-}Justification} \rangle & ::= & \epsilon & \mid & \langle \mathit{String} \rangle \\
\langle \mathit{Var} \rangle & ::= & \langle \mathit{String} \rangle & & \\
\langle \mathit{ListRule} \rangle & ::= & \epsilon & \mid & \langle \mathit{Rule} \rangle \; \langle \mathit{ListRule} \rangle
\end{array}
$$

$$
\begin{array}{lcl}
\langle \mathit{Proof} \rangle & ::= & \texttt{trivial} \; \langle \mathit{Justification} \rangle \\
 & \mid & \langle \mathit{Rule} \rangle \; \langle \mathit{Justification} \rangle
\end{array}
$$

$$
\begin{array}{lcl}
\langle \mathit{Rule} \rangle & ::= & \texttt{assume} \; \langle \mathit{Formula} \rangle \; \langle \mathit{Local\text{-}Justification} \rangle \; \langle \mathit{Rule} \rangle \\
 & \mid & \texttt{let} \; \langle \mathit{Var} \rangle \; \langle \mathit{Rule} \rangle \\
 & \mid & \texttt{split} \; \langle \mathit{ListRule} \rangle \\
 & \mid & \texttt{show} \; \langle \mathit{Formula} \rangle \; \langle \mathit{Proof} \rangle \\
 & \mid & \texttt{continue} \; \langle \mathit{Proof} \rangle
\end{array}
$$

The data structure for Proofs in BNF notation

$$
\begin{array}{lcll}
\langle \textit{Justification} \rangle & ::= & \langle \textit{String} \rangle & \\
\langle \textit{Formula} \rangle & ::= & \langle \textit{String} \rangle & \\
\langle \textit{Local-Justification} \rangle & ::= & \epsilon & \mid \ \langle \textit{String} \rangle \\
\langle \textit{Var} \rangle & ::= & \langle \textit{String} \rangle & \\
\langle \textit{ListRule} \rangle & ::= & \epsilon & \mid \ \langle \textit{Rule} \rangle \ \langle \textit{ListRule} \rangle
\end{array}
$$

$$
\begin{array}{lcl}
\langle \textit{Proof} \rangle & ::= & \texttt{trivial} \ \langle \textit{Justification} \rangle \\
 & \mid & \langle \textit{Rule} \rangle \ \langle \textit{Justification} \rangle
\end{array}
$$

$$
\begin{array}{lcl}
\langle \textit{Rule} \rangle & ::= & \texttt{assume} \ \langle \textit{Formula} \rangle \ \langle \textit{Local-Justification} \rangle \ \langle \textit{Rule} \rangle \\
 & \mid & \texttt{let} \ \langle \textit{Var} \rangle \ \langle \textit{Rule} \rangle \\
 & \mid & \texttt{split} \ \langle \textit{ListRule} \rangle \\
 & \mid & \texttt{show} \ \langle \textit{Formula} \rangle \ \langle \textit{Proof} \rangle \\
 & \mid & \texttt{continue} \ \langle \textit{Proof} \rangle
\end{array}
$$

The data structure for Proofs in BNF notation

$$
\begin{array}{lclcl}
\langle \textit{Justification} \rangle & ::= & \langle \textit{String} \rangle & & \\
\langle \textit{Formula} \rangle & ::= & \langle \textit{String} \rangle & & \\
\langle \textit{Local-Justification} \rangle & ::= & \epsilon & | & \langle \textit{String} \rangle \\
\langle \textit{Var} \rangle & ::= & \langle \textit{String} \rangle & & \\
\langle \textit{ListRule} \rangle & ::= & \epsilon & | & \langle \textit{Rule} \rangle \; \langle \textit{ListRule} \rangle
\end{array}
$$

$$
\begin{array}{lcl}
\langle \textit{Proof} \rangle & ::= & \texttt{trivial} \; \langle \textit{Justification} \rangle \\
 & | & \langle \textit{Rule} \rangle \; \langle \textit{Justification} \rangle
\end{array}
$$

$$
\begin{array}{lcl}
\langle \textit{Rule} \rangle & ::= & \texttt{assume} \; \langle \textit{Formula} \rangle \; \langle \textit{Local-Justification} \rangle \; \langle \textit{Rule} \rangle \\
 & | & \texttt{let} \; \langle \textit{Var} \rangle \; \langle \textit{Rule} \rangle \\
 & | & \texttt{split} \; \langle \textit{ListRule} \rangle \\
 & | & \texttt{show} \; \langle \textit{Formula} \rangle \; \langle \textit{Proof} \rangle \\
 & | & \texttt{continue} \; \langle \textit{Proof} \rangle
\end{array}
$$

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"

Mathematical Vernacular/Concrete Syntax

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"
  ```

Natural deduction style representation

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

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\ldots(2|(x+y)) \vdash \mathit{trivial}}\ \mathit{MultiplicationRule}}{\ldots x+y=2*a+2*b; x+y=2*(a+b); \cdots \vdash \mathit{continue}}}{a, b, x, y \in \mathbf{Z}, x=2*a; y=2*b; a<x; b<y; \cdots \vdash \mathit{continue}}\ \mathit{ByEvenness}}{x, y \in \mathbf{Z}, \mathit{even}(x), \mathit{even}(y) \vdash \mathit{even}(x+y)}}{x, y \in \mathbf{Z} \vdash \mathit{even}(x), \mathit{even}(y) \rightarrow \mathit{even}(x+y)}}{\vdash \mathit{forall} x, y \in \mathbf{Z}, \mathit{even}(x), \mathit{even}(y) \rightarrow \mathit{even}(x+y)}
$$

What we can parse now?

- we assume that "`sqrt of 2 is rational`" and obtain a contradiction.
- we assume that "`sqrt of 2 is rational`" and get a contradiction.
- we assume that "`x is an integer`". we show that "`it is true`".
- we assume that "`x is an integer`" and we show that "`it is true`".
- let "`x is an integer`". we have to show "some formula" and it is trivial.
- let/suppose "`x is an integer`". we have to show "`some formula`" and it is trivial by "`Some Hint`".
- We show that "`foo`". we suppose that "`...`" by "`x`". We assume that "`...`" by "`NN`" and we suppose that "`NN`" by "`NN`".
- Show "`foo`" it is trivial and suppose "`NN`" by "`x`".

Outline

1. **Introduction**
2. **Mathematical Proofs and specifications**
   - Software Specifications
   - The Mathematical Proofs
3. **Implementation Details**
4. **Future Work**

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

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

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

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

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

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

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

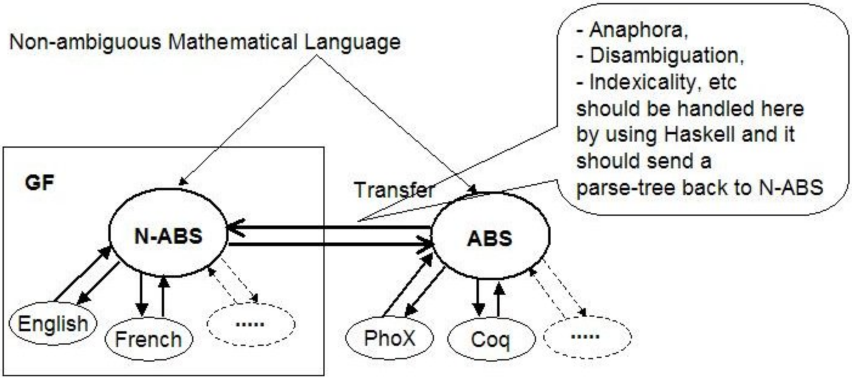
Grammar writing

- 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
```

Big picture



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

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