Term Indexing for the LEO-II Prover

Frank Theiss and Christoph Benzmüller

FR Informatik
Universität des Saarlandes
Saarbrücken, Germany

Computer Laboratory
The University of Cambridge
Cambridge, UK
Overview

- Motivation
Overview

- Motivation
- Some introductory conventions and remarks
Overview

- Motivation
- Some introductory conventions and remarks
- Closer look at some key features of the approach
Overview

- Motivation
- Some introductory conventions and remarks
- Closer look at some key features of the approach
- Preliminary Evaluation
Overview

- Motivation
- Some introductory conventions and remarks
- Closer look at some key features of the approach
- Preliminary Evaluation
- Conclusion
LEO

- Implements **extensional higher order resolution**: 
Implement extensional higher order resolution:

\[ \forall B_\alpha \rightarrow_o, C_\alpha \rightarrow_o, D_\alpha \rightarrow_o. B \cup (C \cap D) = (B \cup C) \cap (B \cup D) \]

Negation and definition expansion with

\[ \cup = \lambda A_\alpha \rightarrow_o, B_\alpha \rightarrow_o, X_\alpha.(A X) \lor (B X) \quad \cap = \lambda A_\alpha \rightarrow_o, B_\alpha \rightarrow_o, X_\alpha.(A X) \land (B X) \]

leads to:

\[ C_1 : [\lambda X_\alpha((b X) \lor ((c X) \land (d X)) \neq? \quad \lambda X_\alpha((b X) \lor (c X)) \land ((b X) \lor (d X)))] \]

Goal directed functional and Boolean extensionality treatment:

\[ C_2 : [(b x) \lor ((c x) \land (d x)) \iff (b x) \lor (c x)) \land ((b x) \lor (d x))] \]

Clause normalization results then in a pure propositional, i.e. decidable, set of clauses. Only these clauses are still in the search space of LEO (in total there are 33 clauses generated and LEO finds the proof on a 2,5GHz PC in 820ms).
LEO

- Implements extensional higher order resolution:
- Based on extended set of support architecture
LEO

- Implements extensional higher order resolution:
- Based on extended set of support architecture

Architecture of LEO:

1. choose lightest from SOS
2. Integrate light. to USABLE
3. resolve with USABLE
4. paramodulate with USABLE
5. factorize lightest
6. primitive substitution on lightest
7. extensionality treatment on EXT
8. pre-unification on CONT
9. process results (tautology deletion)
10. pre-unification on Processed
11. store continuation object
12. check if extensionally interesting
13. Integrate Unified into SOS
- Implements **extensional higher order resolution**: 
- Based on **extended set of support architecture** 
- Developed in **LISP** within the (old) **OMEGA** framework
LEO

- Implements **extensional higher order resolution**: 
- Based on **extended set of support architecture** 
- Developed in **LISP** within the (old) **OMEGA** framework 
- Uses **KEIM** generic datastructures (how inefficient?)
LEO

- Implements extensional higher order resolution:
- Based on extended set of support architecture
- Developed in LISP within the (old) OMEGA framework
- Uses KEIM generic datastructures (how inefficient?)
- Can handle only a few thousand clauses in search space
LEO

- Implements extensional higher order resolution:
- Based on extended set of support architecture
- Developed in LISP within the (old) OMEGA framework
- Uses KEIM generic datastructures (how inefficient?)
- Can handle only a few thousand clauses in search space
- Often generates lots of first order or propositional clauses
LEO

- Implements extensional higher order resolution:
- Based on extended set of support architecture
- Developed in LISP within the (old) OMEGA framework
- Uses KEIM generic datastructures (how inefficient?)
- Can handle only a few thousand clauses in search space
- Often generates lots of first order or propositional clauses
- Has been successfully combined with first order ATPs
- EPSRC project at University of Cambridge (with L. Paulson)
LEO-II

- EPSRC project at University of Cambridge (with L. Paulson)
- Reimplementation of LEO in OCAML
LEO-II

- EPSRC project at University of Cambridge (with L. Paulson)
- Reimplementation of LEO in OCAML
- Shall combine the lessons of recent research with new ideas
LEO-II

- EPSRC project at University of Cambridge (with L. Paulson)
- Reimplementation of LEO in OCAML
- Shall combine the lessons of recent research with new ideas
- Shall be easy to integrate with proof assistants (ISABELLE/HOL, HOL, OMEGA, . . . )
LEO-II

- EPSRC project at University of Cambridge (with L. Paulson)
- Reimplementation of LEO in OCAML
- Shall combine the lessons of recent research with new ideas
- Shall be easy to integrate with proof assistants (ISABELLE/HOL, HOL, OMEGA, . . .)
- HOTPTP as input syntax
LEO-II

- EPSRC project at University of Cambridge (with L. Paulson)
- Reimplementation of LEO in OCAML
- Shall combine the lessons of recent research with new ideas
- Shall be easy to integrate with proof assistants (ISABELLE/HOL, HOL, OMEGA, . . . )
- HOTPTP as input syntax
- Shall cooperate with first order automated theorem provers
LEO-II

- EPSRC project at University of Cambridge (with L. Paulson)
- Reimplementation of LEO in OCAML
- Shall combine the lessons of recent research with new ideas
- Shall be easy to integrate with proof assistants (ISABELLE/HOL, HOL, OMEGA, ...)
- HOTPTP as input syntax
- Shall cooperate with first order automated theorem provers
- Shall be way more efficient than LEO (which was developed rather as an academic demonstrator than a prototype)
Motivation

- Term indexing is standard in first order theorem proving
Motivation

- Term indexing is standard in first order theorem proving
- Efficiency is now an important issue in LEO-II redevelopment
Motivation

- Term indexing is standard in first order theorem proving
- Efficiency is now an important issue in LEO-II redevelopment
- Comparably few techniques studied for higher order logic
Motivation

- Term indexing is standard in first order theorem proving
- Efficiency is now an important issue in LEO-II redevelopment
- Comparably few techniques studied for higher order logic
- One example: Brigitte Pientka’s work [ICLP-03]
  - based on higher order substitution tree indexing
  - relies on unification of linear higher order patterns
Motivation

- Term indexing is standard in first order theorem proving
- Efficiency is now an important issue in LEO-II redevelopment
- Comparably few techniques studied for higher order logic
- One example: Brigitte Pientka’s work [ICLP-03]
  - based on higher order substitution tree indexing
  - relies on unification of linear higher order patterns
- Our approach
  - based on Stickel’s coordinate and path indexing [SRI-Report-89]
  - low level operations (e.g., operations on hashtables)
HOL-Syntax: Simple Types

Simple Types $\mathcal{T}$:

- $o$ (truth values)
- $\iota$ (individuals)
- $(\alpha \rightarrow \beta)$ (functions from $\alpha$ to $\beta$)
HOL-Syntax: Simply Typed $\lambda$-Terms

Typed Terms:

$X_\alpha$ Variables ($\mathcal{V}$)

$c_\alpha$ Constants & Parameters ($\Sigma$)

$(F_{\alpha \rightarrow \beta} B_\alpha)_\beta$ Application

$(\lambda Y_\alpha A_\beta)_{\alpha \rightarrow \beta}$ $\lambda$-abstraction
HOL-Syntax: Simply Typed $\lambda$-Terms

Typed Terms:

- $X_\alpha$ Variables ($\mathcal{V}$)
- $c_\alpha$ Constants & Parameters ($\Sigma$)
- $(F_{\alpha \rightarrow \beta} B_\alpha)_\beta$ Application
- $(\lambda Y_\alpha A_\beta)_{\alpha \rightarrow \beta}$ $\lambda$-abstraction

Equality of Terms:

- $\alpha$-conversion Changing bound variables
  - $\beta$-reduction $((\lambda Y_\beta A_\alpha) B_\beta) \xrightarrow{\beta} [B/Y]A$
  - $\eta$-reduction $(\lambda Y_\alpha (F_{\alpha \rightarrow \beta} Y)) \xrightarrow{\eta} F$ $(Y_\beta \notin \text{Free}(F))$
HOL-Syntax: Simply Typed $\lambda$-Terms

Typed Terms:

- $X_\alpha$: Variables ($\mathcal{V}$)
- $c_\alpha$: Constants & Parameters ($\Sigma$)
- $(F_{\alpha \to \beta} B_\alpha)_\beta$: Application
- $(\lambda Y_\alpha A_\beta)_{\alpha \to \beta}$: $\lambda$-abstraction

Equality of Terms:

Every term has a unique $\beta\eta$-normal form (up to $\alpha$-conversion).
de Bruijn Notation

- Instead of named bound variables we use de Bruijn indices
de Bruijn Notation

- Instead of named bound variables we use de Bruijn indices

- Example:

\[ \lambda a. \lambda b. (b = ((\lambda c. (c b)) a)) \]

translates to

\[ \lambda \lambda. (x_0 = ((\lambda. (x_0 x_1)) x_1)) \]
de Bruijn Notation

- Instead of named bound variables we use de Bruijn indices

- Example:

\[ \lambda a. \lambda b. (b = ((\lambda c. (c \ b)) \ a)) \]

translates to

\[ \lambda \lambda. (x_0 = ((\lambda. (x_0 \ x_1)) \ x_1)) \]

- Different occurrences of the same bound variable may have different de Bruijn indices:

\[ b \text{ translates to both } x_0 \text{ and } x_1 \]
de Bruijn Notation

- Instead of named bound variables we use de Bruijn indices

- Example:

\[
\lambda a . \lambda b . (b = ((\lambda c . (c b)) a))
\]

translates to

\[
\lambda \lambda . (x_0 = ((\lambda . (x_0 \ x_1)) \ x_1))
\]

- Different occurrences of the same bound variable may have different de Bruijn indices:

  \( b \) translates to both \( x_0 \) and \( x_1 \)

- Different occurrences of the same de Bruijn index may refer to different \( \lambda \)-binders:

  \( x_0 \) relates to bound variable \( b \) and \( c \)
Some Important Remarks

- Only terms in $\beta\eta$ normal form ($\eta$ short and $\beta$ normal) are indexed
Some Important Remarks

- Only terms in $\beta\eta$ normal form ($\eta$ short and $\beta$ normal) are indexed.
- Example: $(\lambda.(\lambda.(x_1 x_0)))((\lambda.g x_0)c)$ normalises to $(g c)$.
Some Important Remarks

- Only terms in $\beta\eta$ normal form ($\eta$ short and $\beta$ normal) are indexed

- Example: $(\lambda.(\lambda.(x_1x_0))))((\lambda.gx_0)c)$ normalises to $(gc)$

- Invariant for LEO-II: normalisation after each calculus step
Some Important Remarks

- Only terms in $\beta\eta$ normal form ($\eta$ short and $\beta$ normal) are indexed
- Example: $(\lambda.(\lambda.(x_1 \, x_0))))((\lambda.g \, x_0)c)$ normalises to $(g \, c)$
- Invariant for LEO-II: normalisation after each calculus step
- Types
  - provide an additional criterion for distinction of terms (e.g., different occurrences of the same de Bruijn index may have different types)
  - no further impact on the indexing mechanism
Some Important Remarks

- Only terms in $\beta\eta$ normal form ($\eta$ short and $\beta$ normal) are indexed
- Example: $((\lambda.(x_1 \ x_0)))((\lambda.g \ x_0)c)$ normalises to $(g \ c)$
- Invariant for LEO-II: normalisation after each calculus step
- Types
  - provide an additional criterion for distinction of terms (e.g., different occurrences of the same de Bruijn index may have different types)
  - no further impact on the indexing mechanism
- Due to Currying all our applications have just one argument
Key Features of Our Approach

1. Shared representation of terms
Key Features of Our Approach

1. Shared representation of terms
2. Use of partial syntax trees to speedup logical computations
Key Features of Our Approach

1. Shared representation of terms
2. Use of partial syntax trees to speedup logical computations
3. Indexing of subterm occurrences
Key Features of Our Approach

1. Shared representation of terms
2. Use of partial syntax trees to speedup logical computations
3. Indexing of subterm occurrences
4. Indexing of bound variable occurrences
   (support for explicit substitutions)
Shared Representation of Terms

- Terms are represented as sets of term nodes
Shared Representation of Terms

Terms are represented as sets of term nodes

- Symbol \( s \in \Sigma \) → a term node \( n : \text{symbol}(s) \) is created

- Bound variable \( x_d \) (\( d \) is de Bruijn index) → a term node \( m : \text{bound}(\text{type}, d) \) is created

- Application \((s \ t)\) (\( s, t \) already represented by nodes \( i, j \)) → a term node \( l : \text{application}(i, j) \) is created

- Abstraction \( \lambda t \) (\( t \) is already represented by \( i \)) → a term node \( k : \text{abstraction}(\text{type}, i) \) is created
Shared Representation of Terms

- Terms are represented as sets of term nodes
- Example:
Shared Representation of Terms

- Terms are represented as sets of term nodes
- Example: Original problem terms
  - def-emptyset: \( \emptyset := \lambda x. \bot \) 
  - def-element: \( \in := \lambda y. \lambda s. \rightarrow o. (s \ y) \) 
  - theorem1: \( \neg (A_l \in \emptyset) \) 
  - theorem1alt: \( (A_l \in \emptyset) \Rightarrow \bot \)
Shared Representation of Terms

- Terms are represented as sets of term nodes
- Example: HOTPPTP encoding

\[
\text{thf}(\text{emptyset}, \text{definition},
\text{emptyset} := \langle^
\text{Z : i} : \text{false} \rangle).
\]

\[
\text{thf}(\text{element}, \text{definition},
\text{element} := \langle^
\text{Y:i, S:(i > o)} : (S @ Y) \rangle).
\]

\[
\text{thf}(\text{theoreml}, \text{conjecture},
\langle \neg ((\text{element} @ A) @ \text{emptyset}) \rangle).
\]

\[
\text{thf}(\text{theoremlalt}, \text{conjecture},
((\text{element} @ A) @ \text{emptyset}) \Rightarrow \text{false})).
\]
Shared Representation of Terms

- Terms are represented as sets of term nodes
- Example: Graph representation of terms

```
1: λ
   0: false

6: λ
   5: λ
   4: @
   2: x₀
   3: x₁

13: @
   7: neg
   10: @
   8: element
   9: A

12: @

16: @
   15: @
   14: implies
   12: @
   10: @
   11: emptyset
   8: element
   9: A
```

© Theiss and Benzmüller, 2006

– p.11
Shared Representation of Terms

- Terms are represented as sets of term nodes
- Example: Representation as term sets

0: symbol false  
1: abstr(i,0)  
2: bound(i -> o,0)  
3: bound(i,1)  
4: appl(2,3)  
5: abstr(i -> o,4)  
6: abstr(i,5)  
7: symbol neg  
8: symbol element  
9: symbol A  
10: appl(8,9)  
11: symbol emptyset  
12: appl(10,11)  
13: appl(7,12)  
14: symbol implies  
15: appl(14,12)  
16: appl(15,0)
Shared Representation of Terms

- Terms are represented as sets of term nodes
- Example: Parsing returns pointers to this term set / index

emptyset: $\lambda [Z]: \text{false}$
->index: 1

element: $\lambda [Y]: \lambda [S]: S \; Y$
->index: 6

theoreml: $\text{neg} \; ((\text{element} \; A) \; \text{emptyset})$
->index: 13

theoremlalt: $(\text{implies} \; ((\text{element} \; A) \; \text{emptyset})) \; \text{false}$
->index: 16
Shared Representation of Terms

- Terms are represented as sets of term nodes
- Example: Term set representation supported via hashtables

- \[ \text{ht abstr}\text{\_with\_scope} : \mathbb{IN} \rightarrow \mathbb{IN} : \]
  lookup abstractions with a given scope \( i \)

- \[ \text{ht appl}\text{\_with\_func} : \mathbb{IN} \rightarrow \mathbb{IN} \rightarrow \mathbb{IN} : \]
  lookup application with a given function \( i \) and argument \( j \)

- \[ \text{ht appl}\text{\_with\_arg} : \mathbb{IN} \rightarrow \mathbb{IN} \rightarrow \mathbb{IN} : \]
  lookup application with a given argument \( j \) and function \( i \)
Partial Syntax Trees (PST)

- Representation of the paths to particular symbol or subterm occurrences within a term
Partial Syntax Trees (PST)

- Representation of the paths to particular symbol or subterm occurrences within a term
- Called *partial* because they only represent *relevant parts* of a term
Partial Syntax Trees (PST)

- Representation of the paths to particular symbol or subterm occurrences within a term
- Called partial because they only represent relevant parts of a term
- Allow for early detection of branches in a term’s syntax tree with no occurrences of a specific symbol/subterm, since these branches are not represented
Partial Syntax Trees (PST)

- Representation of the paths to particular symbol or subterm occurrences within a term
- Called partial because they only represent relevant parts of a term
- Allow for early detection of branches in a term’s syntax tree with no occurrences of a specific symbol/subterm, since these branches are not represented
- Need to define position before we can give an example PST
Consider term $(\lambda x_0)(f@a)$:

\[
\begin{align*}
(\lambda x_0)(f@a) & : [] \\
\lambda x_0 & : [\text{func}] \\
x_0 & : [\text{func}; \text{arg}] \\
f@a & : [\text{arg}] \\
f & : [\text{arg}; \text{func}] \\
a & : [\text{arg}; \text{arg}] 
\end{align*}
\]
Partial Syntax Trees and Replacement

Example

- **PST** for occurrences of symbol emptyset in theorem1

```
13: @
  
  7: neg
  
  12: @

10: @
  
11: emptyset

8: element

9: A
```

positiontable:
[arg; arg] : emptyset
end.

\[
p_{\text{emptyset-1}} = \text{pst}(\_, \_, p_{\text{emptyset-2}})
\]

\[
p_{\text{emptyset-2}} = \text{pst}(\_, \_, p_{\text{emptyset-3}})
\]

\[
p_{\text{emptyset-3}} = \text{pst}(\_, \_, \_ - \text{emptyset})
\]
Partial Syntax Trees and Replacement

Example

- Modified term after replacement

```
  18: @
    /   \
  7: neg 17: @
    /    /
 10: @ 1: λ
     / \
8: element 9: A
   /
0: false
```
Partial Syntax Trees and Replacement

Example

- PST for occurrences of symbol element in term

```
positiontable:
[arg; func; func] : element
end.
```

\[
p_{\text{emptyset-1}} = pst(_, _, p_{\text{emptyset-2}})
p_{\text{emptyset-2}} = pst(_, p_{\text{emptyset-3}}, _)
p_{\text{emptyset-3}} = pst(_, p_{\text{emptyset-4}}, _)
p_{\text{emptyset-4}} = pst(_, _, _) - element
\]
Partial Syntax Trees and Replacement

Example

- Modified term after replacement

```
21: @
  7: neg
  20: @
    19: @
      1: λ
   6: λ
   9: A
   0: false
  5: λ
  4: @

2: x₀
3: x₁
```
Partial Syntax Trees and Replacement

Example

- Use available information for normalisation

```
21: @
  7: neg
  20: @
    19: @
      6: λ
      9: A
        1: λ
        0: false
          4: @
            2: x₀
            3: x₁
```

© Theiss and Benzmüller, 2006
Partial Syntax Trees and Replacement

Example

- We also index occurrences of bound variables

```
  6: λ
  |  
  5: λ
  |  
  4: @
  |  
  3: x₁
```
Partial Syntax Trees and Replacement

Example

- Normalisation (required before term goes to index again)

\[
\begin{array}{c}
21: @ \\
7: \text{neg} \\
20: @ \\
19: @ \\
6: \lambda \\
5: \lambda \\
4: @ \\
2: x_0 \\
3: x_1
\end{array}
\quad \rightarrow 
\begin{array}{c}
24: @ \\
7: \text{neg} \\
0: \text{false}
\end{array}
\]

(LEO-II will later immediately say: Proof found!)
Partial Syntax Trees and Replacement

Example

- Normalisation (required before term goes to index again)

```
21: @
  7: neg   20: @
    19: @
      6: λ   9: A
        5: λ
          4: @
            2: x₀  3: x₁
```

```
24: @
  7: neg
    0: false
```

- Hence, we guide replacement & normalization with PSTs
Partial Syntax Trees and Replacement

Example

- Normalisation (required before term goes to index again)

```
21: @
  7: neg
  20: @
    19: @
      6: λ
      9: A
      0: false
      1: λ
```

- Hence, we guide replacement & normalization with PSTs
- Provide also PSTs for bound variable occurrences
Building and Using the Index

- Index records whether and at which positions a subterm occurs in a term (symbols and nonprimitive subterms can be handled – tradeoff?)
Building and Using the Index

- Index records whether and at which positions a subterm occurs in a term (symbols and nonprimitive subterms can be handled – tradeoff?)

- For each new (normalised) term to be indexed the term set is computed and termset hashtables are updated
Building and Using the Index

- Index records whether and at which positions a subterm occurs in a term (symbols and nonprimitive subterms can be handled – tradeoff?)
- For each new (normalised) term to be indexed the term set is computed and termset hashtables are updated
- At the same time term is analysed for contained subterms
Building and Using the Index

- Index records whether and at which positions a subterm occurs in a term (symbols and nonprimitive subterms can be handled – tradeoff?)
- For each new (normalised) term to be indexed the term set is computed and termset hashtables are updated
- At the same time term is analysed for contained subterms
- Result is a PST for each subterm of a given term to be indexed
Building and Using the Index

- Index records whether and at which positions a subterm occurs in a term (symbols and nonprimitive subterms can be handled – tradeoff?)
- For each new (normalised) term to be indexed the term set is computed and termset hashtables are updated
- At the same time term is analysed for contained subterms
- Result is a PST for each subterm of a given term to be indexed
- PSTs are added to hashtable \( \text{occurrences} : IN \rightarrow IN \rightarrow PST \)
Building and Using the Index

- Index records whether and at which positions a subterm occurs in a term (symbols and nonprimitive subterms can be handled – tradeoff?)
- For each new (normalised) term to be indexed the term set is computed and termset hashtables are updated
- At the same time term is analysed for contained subterms
- Result is a PST for each subterm of a given term to be indexed
- PSTs are added to hashtable occurrences: $IN \rightarrow IN \rightarrow PST$
- Second hashtable occurs_in: $IN \rightarrow IN^*$
Building and Using the Index

- Index records whether and at which positions a subterm occurs in a term (symbols and nonprimitive subterms can be handled – tradeoff?)
- For each new (normalised) term to be indexed the term set is computed and termset hashtables are updated
- At the same time term is analysed for contained subterms
- Result is a PST for each subterm of a given term to be indexed
- PSTs are added to hashtable $\text{occurrences} : \mathbb{IN} \to \mathbb{IN} \to \text{PST}$
- Second hashtable $\text{occurs\_in} : \mathbb{IN} \to \mathbb{IN}^*$
- Third hashtable $\text{occurs\_at} : \text{pos} \to \mathbb{IN} \to \mathbb{IN}^*$
Preliminary Evaluation

- 977 random terms from a HOTPTP version of Jutting’s Automath encoding of Landau’s book Grundlagen der Analysis
Preliminary Evaluation

- **977 random terms** from a HOTPTP version of Jutting’s Automath encoding of Landau’s book *Grundlagen der Analysis*

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of indexed terms</td>
<td>977</td>
</tr>
<tr>
<td>Number of created term nodes</td>
<td>11618</td>
</tr>
<tr>
<td>Average term size</td>
<td>54</td>
</tr>
<tr>
<td>Number of nodes with no parent nodes</td>
<td>904</td>
</tr>
<tr>
<td>Number of nodes with one parent node</td>
<td>9633</td>
</tr>
<tr>
<td>Number of nodes with two more parent nodes</td>
<td>1083</td>
</tr>
<tr>
<td>Maximum number of parent nodes</td>
<td>2778 (symbol ∀)</td>
</tr>
<tr>
<td>Average number of parent nodes</td>
<td>1.68</td>
</tr>
<tr>
<td>Average number of terms a node occurs in</td>
<td>33.5</td>
</tr>
<tr>
<td>-”-(for symbols)</td>
<td>493.9</td>
</tr>
<tr>
<td>-”-(for nonprimitive term nodes)</td>
<td>24</td>
</tr>
<tr>
<td>Average PST/term size for symbol occurrences</td>
<td>0.21</td>
</tr>
<tr>
<td>Average PST/term size for bound variable occurrences</td>
<td>0.33</td>
</tr>
<tr>
<td>Average PST/term size for all term nodes</td>
<td>0.12</td>
</tr>
</tbody>
</table>
Preliminary Evaluation

- **977 random terms** from a HOTPTP version of Jutting’s Automath encoding of Landau’s book Grundlagen der Analysis
- Average of 1.68 parent nodes seems low / relativised by average of **33.5 terms a node occurs in**
Preliminary Evaluation

- 977 random terms from a HOTPTP version of Jutting’s Automath encoding of Landau’s book Grundlagen der Analysis
- Average of 1.68 parent nodes seems low / relativised by average of 33.5 terms a node occurs in
- One indicator for term retrieval performance: 99.7% of candidate nodes can be excluded (average of occurrences 33.5 versus 11618 terms)
Preliminary Evaluation

- 977 random terms from a HOTPTP version of Jutting’s Automath encoding of Landau’s book Grundlagen der Analysis

- Average of 1.68 parent nodes seems low / relativised by average of 33.5 terms a node occurs in

- One indicator for term retrieval performance: 99.7% of candidate nodes can be excluded (average of occurrences 33.5 versus 11618 terms)

- Occurs check for symbols, bound variables, and non primitive subterms in constant time is expected to have strong impact on crucial operations (e.g., replacement, substitution, global unfolding of definitions)
Preliminary Evaluation

■ 977 random terms from a HOTPTP version of Jutting’s Automath encoding of Landau’s book Grundlagen der Analysis

■ Average of 1.68 parent nodes seems low / relativised by average of 33.5 terms a node occurs in

■ One indicator for term retrieval performance: 99.7% of candidate nodes can be excluded (average of occurrences 33.5 versus 11618 terms)

■ Occurs check for symbols, bound variables, and non primitive subterms in constant time is expected to have strong impact on crucial operations (e.g., replacement, substitution, global unfolding of definitions)

■ One indicator for improvement for replacement operations is PST/term size rate: 0.21 (= speedup factor 5) for symbols and 0.33 (speedup factor 3) for bound variables
Conclusion

- Presented a term indexing approach that
Conclusion

- Presented a term indexing approach that
  - combines ideas from first order coordinate and path indexing [Stickel-89]
Conclusion

- Presented a term indexing approach that
  - combines ideas from first order coordinate and path indexing [Stickel-89]
  - with novel ideas, especially partial syntax trees,
Conclusion

- Presented a term indexing approach that
  - combines ideas from first order coordinate and path indexing [Stickel-89]
  - with novel ideas, especially partial syntax trees,
  - in a higher order context
Conclusion

- Presented a term indexing approach that
  - combines ideas from first order coordinate and path indexing \[\text{[Stickel-89]}\]
  - with novel ideas, especially partial syntax trees,
  - in a higher order context

- Orthogonal/complementary to Pientka’s approach — combination ?
Conclusion

- Presented a term indexing approach that
  - combines ideas from first order coordinate and path indexing [Stickel-89]
  - with novel ideas, especially partial syntax trees,
  - in a higher order context
- Orthogonal/complementary to Pientka’s approach — combination?
- Future work:
Conclusion

- Presented a term indexing approach that
  - combines ideas from first order coordinate and path indexing [Stickel-89]
  - with novel ideas, especially partial syntax trees,
  - in a higher order context

- Orthogonal/complementary to Pientka’s approach — combination?

- Future work:
  - Study alternative term representation techniques (suspension calculus, spine representation, explicit substitutions) in our framework
Conclusion

- Presented a term indexing approach that
  - combines ideas from first order coordinate and path indexing [Stickel-89]
  - with novel ideas, especially partial syntax trees,
  - in a higher order context

- Orthogonal/complementary to Pientka’s approach — combination?

- Future work:
  - Study alternative term representation techniques (suspension calculus, spine representation, explicit substitutions) in our framework
  - Proper evaluation in LEO-II
Conclusion

- Presented a term indexing approach that
  - combines ideas from first order coordinate and path indexing [Stickel-89]
  - with novel ideas, especially partial syntax trees,
  - in a higher order context

- Orthogonal/complementary to Pientka’s approach — combination?

- Future work:
  - Study alternative term representation techniques
    (suspension calculus, spine representation, explicit substitutions) in our framework
  - Proper evaluation in LEO-II
Conclusion

- Presented a term indexing approach that
  - combines ideas from first order coordinate and path indexing [Stickel-89]
  - with novel ideas, especially partial syntax trees,
  - in a higher order context

- Orthogonal/complementary to Pientka’s approach — combination?

- Future work:
  - Study alternative term representation techniques (suspension calculus, spine representation, explicit substitutions) in our framework
  - Proper evaluation in LEO-II