



# Term Indexing for the LEO-II Prover

Frank Theiss and **Christoph Benz Müller**

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**:

- Implements **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

$$\cup = \lambda A_{\alpha \rightarrow o}, B_{\alpha \rightarrow o}, X_{\alpha}. (A X) \vee (B X) \quad \cap = \lambda A_{\alpha \rightarrow o}, B_{\alpha \rightarrow o}, X_{\alpha}. (A X) \wedge (B X)$$

leads to:

$$C_1 : [\lambda X_{\alpha}. (b X) \vee ((c X) \wedge (d X)) \neq? \lambda X_{\alpha}. ((b X) \vee (c X)) \wedge ((b X) \vee (d X))]$$

Goal directed functional and Boolean extensionality treatment:

$$C_2 : [(b x) \vee ((c x) \wedge (d x)) \Leftrightarrow ((b x) \vee (c x)) \wedge ((b x) \vee (d x))]^F$$

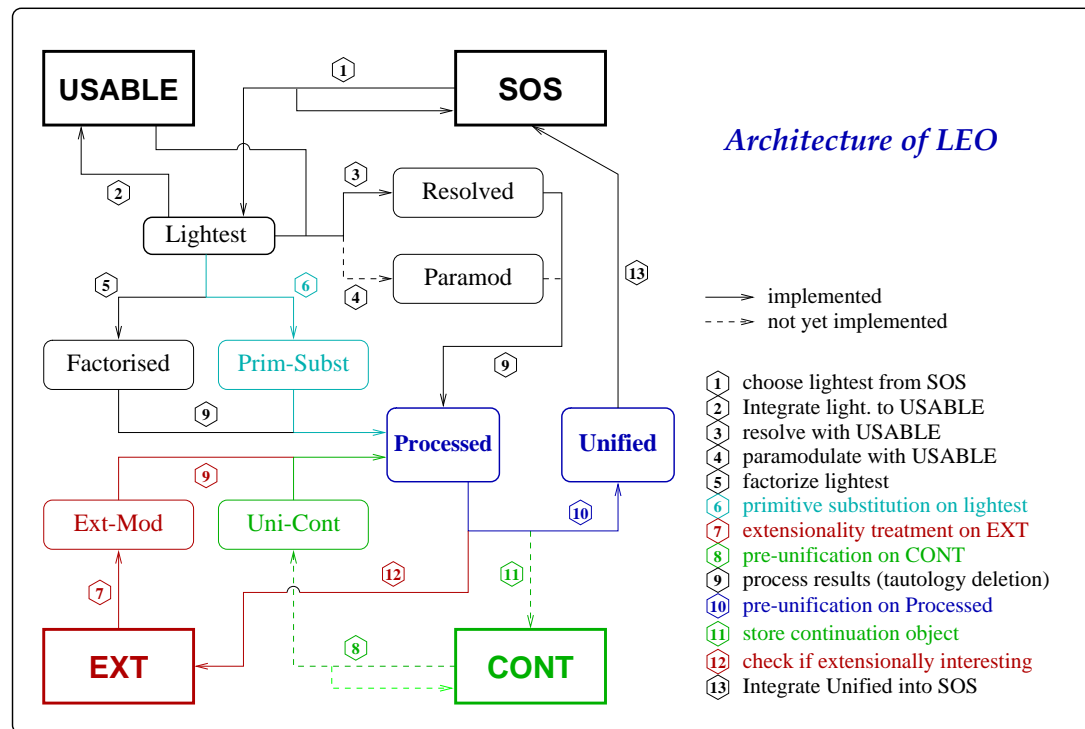
Clause normalization results then in a pure propositional, i.e. decidable, set of clauses. Only these clauses are still in the search space of  $\mathcal{LEO}$  (in total there are 33 clauses generated and  $\mathcal{LEO}$  finds the proof on a 2,5GHz PC in 820ms).

# LEO



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

- 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**
- 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**

# LEO-II



- EPSRC project at University of Cambridge (with L. Paulson)

# LEO-II



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

# LEO-II



- **EPSRC project** at University of Cambridge (with L. Paulson)
- Reimplentation 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)
- Reimplentation 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)
- Reimplentation 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)
- Reimplentation 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)
- Reimplentation 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 ( $\Sigma$ )

$(\mathbf{F}_{\alpha \rightarrow \beta} \mathbf{B}_\alpha)_\beta$  Application

$(\lambda Y_\alpha \mathbf{A}_\beta)_{\alpha \rightarrow \beta}$   $\lambda$ -abstraction

# HOL-Syntax: Simply Typed $\lambda$ -Terms



## Typed Terms:

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

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

$(\mathbf{F}_{\alpha \rightarrow \beta} \mathbf{B}_\alpha)_\beta$  Application

$(\lambda Y_\alpha \mathbf{A}_\beta)_{\alpha \rightarrow \beta}$   $\lambda$ -abstraction

## Equality of Terms:

$\alpha$ -conversion Changing bound variables

$\beta$ -reduction  $((\lambda Y_\beta \mathbf{A}_\alpha) \mathbf{B}_\beta) \xrightarrow{\beta} [\mathbf{B}/Y]\mathbf{A}$

$\eta$ -reduction  $(\lambda Y_\alpha (\mathbf{F}_{\alpha \rightarrow \beta} Y)) \xrightarrow{\eta} \mathbf{F} \quad (Y_\beta \notin \text{Free}(\mathbf{F}))$

# HOL-Syntax: Simply Typed $\lambda$ -Terms



## Typed Terms:

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

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

$(\mathbf{F}_{\alpha \rightarrow \beta} \mathbf{B}_\alpha)_\beta$  Application

$(\lambda Y_\alpha \mathbf{A}_\beta)_{\alpha \rightarrow \beta}$   $\lambda$ -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:

$b$  translates to both  $x_0$  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:  $(\lambda.(\lambda.(x_1 x_0)))((\lambda.g x_0)c)$  normalises to  $(g c)$
- 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:  $(\lambda.(\lambda.(x_1 x_0)))((\lambda.g x_0)c)$  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:
  - ▶ def-element:
  - ▶ theorem1:
  - ▶ theorem1alt:

$$\emptyset := \lambda x_\iota. \perp$$

$$\in := \lambda y_\iota. \lambda s_{\iota \rightarrow o}. (s y)$$

$$\neg(A_\iota \in \emptyset)$$

$$(A_\iota \in \emptyset) \Rightarrow \perp$$



# Shared Representation of Terms

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

```
thf(emptyset,definition,  
    (emptyset :=  
      (^ [Z : $i] : $false))).
```

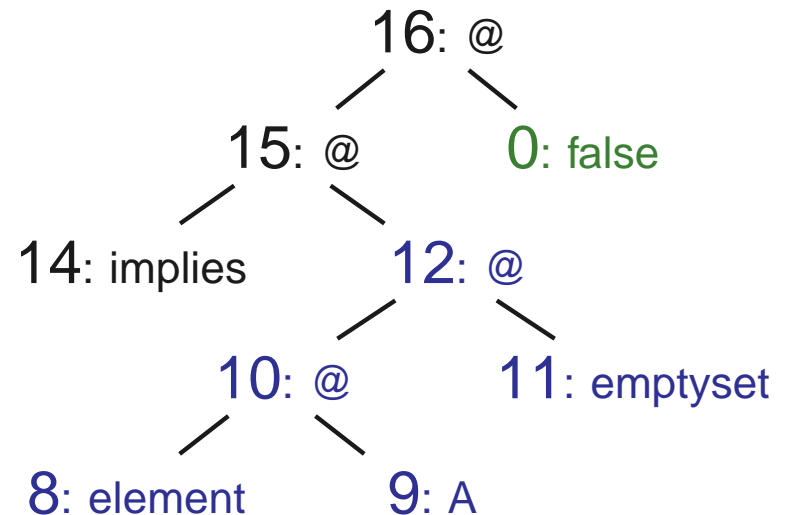
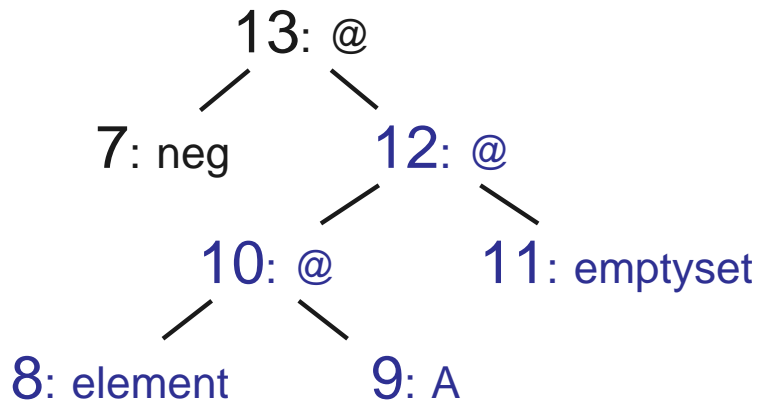
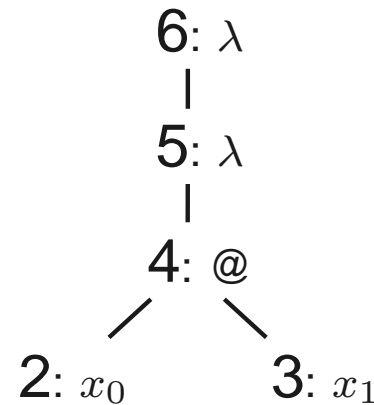
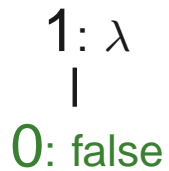
```
thf(element,definition,  
    (element :=  
      (^ [Y:$i, S:($i > $o)] : (S @ Y)))).
```

```
thf(theorem1,conjecture,  
    (~ ((element @ A) @ emptyset))).
```

```
thf(theorem1alt,conjecture,  
    (((element @ A) @ emptyset) => $false)).
```

# Shared Representation of Terms

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



# 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]: false
```

```
->index: 1
```

```
element: lambda [Y]: lambda [S]: S Y
```

```
->index: 6
```

```
theorem1: neg ((element A) emptyset)
```

```
->index: 13
```

```
theorem1alt: (implies ((element A) emptyset)) false
```

```
->index: 16
```

# Shared Representation of Terms



- Terms are represented as sets of term nodes
- Example: **Term set representation supported via hashtables**
  - ▶ ht `abstr_with_scope` :  $\mathbb{N} \rightarrow \mathbb{N}$ :  
lookup abstractions with a given scope  $i$
  - ▶ ht `appl_with_func` :  $\mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$ :  
lookup application with a given function  $i$  and argument  $j$
  - ▶ ht `appl_with_arg` :  $\mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$ :  
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

# Positions



- Consider term  $(\lambda.x_0)@(f@a)$ :

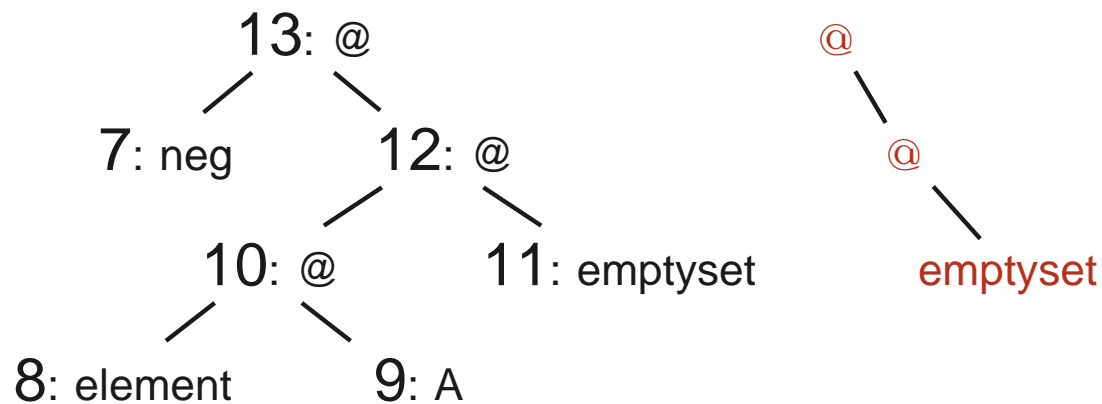
$(\lambda.x_0)@(f@a)$	:	[]
$\lambda.x_0$	:	[func]
$x_0$	:	[func; arg]
$f@a$	:	[arg]
$f$	:	[arg; func]
$a$	:	[arg; arg]

# Partial Syntax Trees and Replacement



## Example

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



```
positiontable:
[arg; arg] : emptyset
end.
```

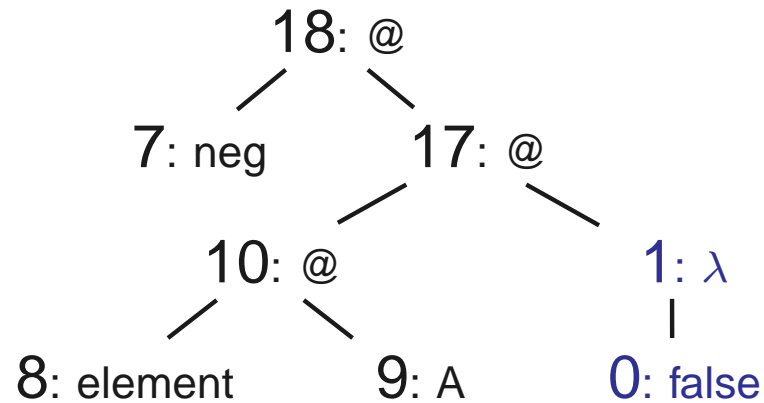
```
 $p_{\text{emptyset-1}} = pst(\_, \_, p_{\text{emptyset-2}})$ 
 $p_{\text{emptyset-2}} = pst(\_, \_, p_{\text{emptyset-3}})$ 
 $p_{\text{emptyset-3}} = pst(\_, \_, \_) \quad - \text{emptyset}$ 
```

# Partial Syntax Trees and Replacement



## Example

- Modified term after replacement

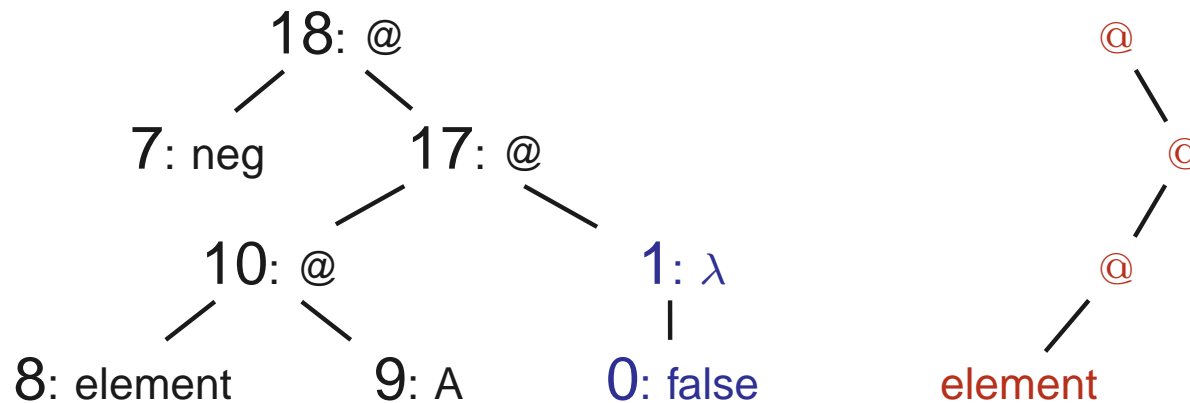


# Partial Syntax Trees and Replacement



## Example

- **PST** for occurrences of symbol element in term



```

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

```

```

pemptyset-1 = pst(−, −, pemptyset-2)
pemptyset-2 = pst(−, pemptyset-3, −)
pemptyset-3 = pst(−, pemptyset-4, −)
pemptyset-4 = pst(−, −, −)      − element

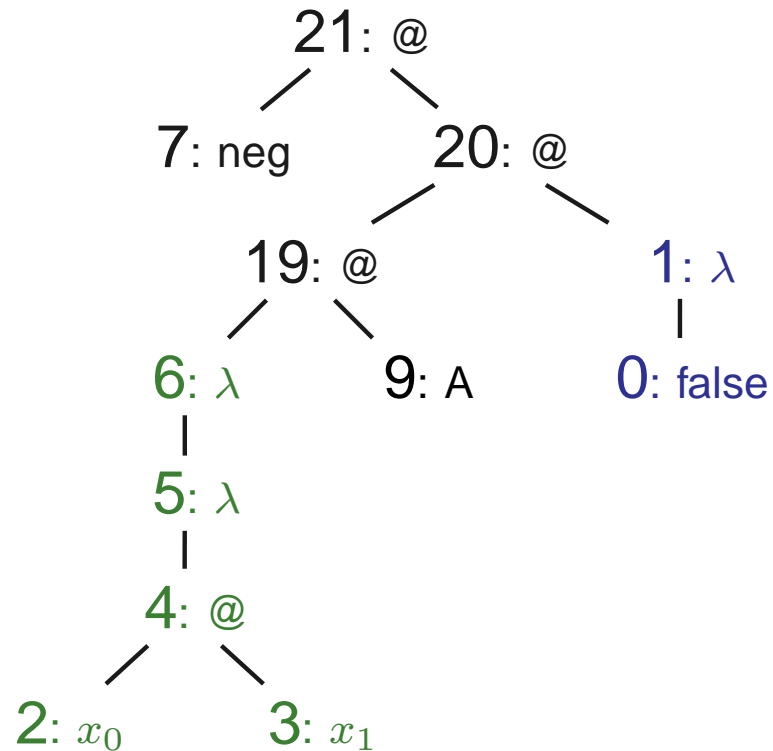
```

# Partial Syntax Trees and Replacement



## Example

- Modified term after replacement

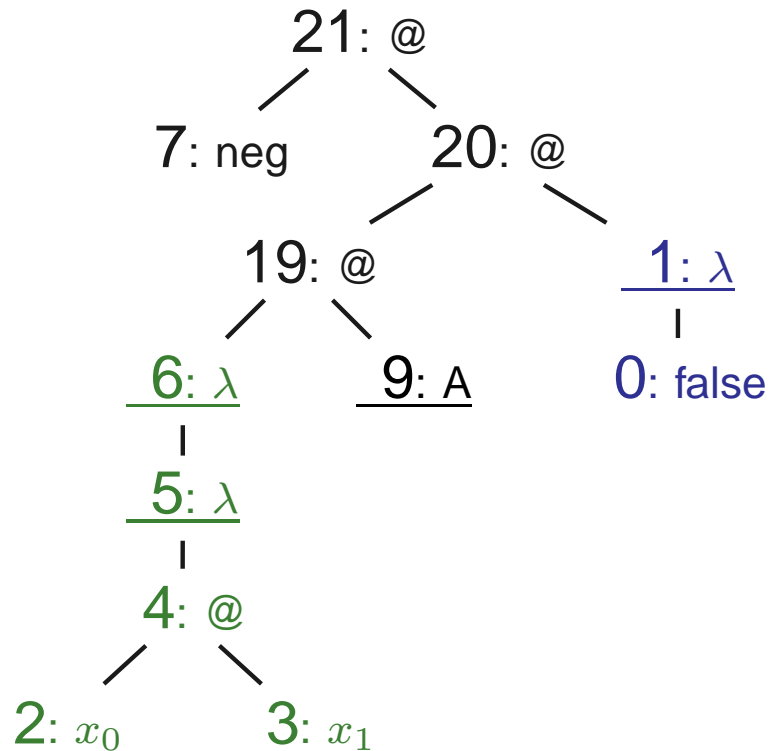


# Partial Syntax Trees and Replacement



## Example

- Use available information for normalisation



# Partial Syntax Trees and Replacement



## Example

- We also index occurrences of bound variables

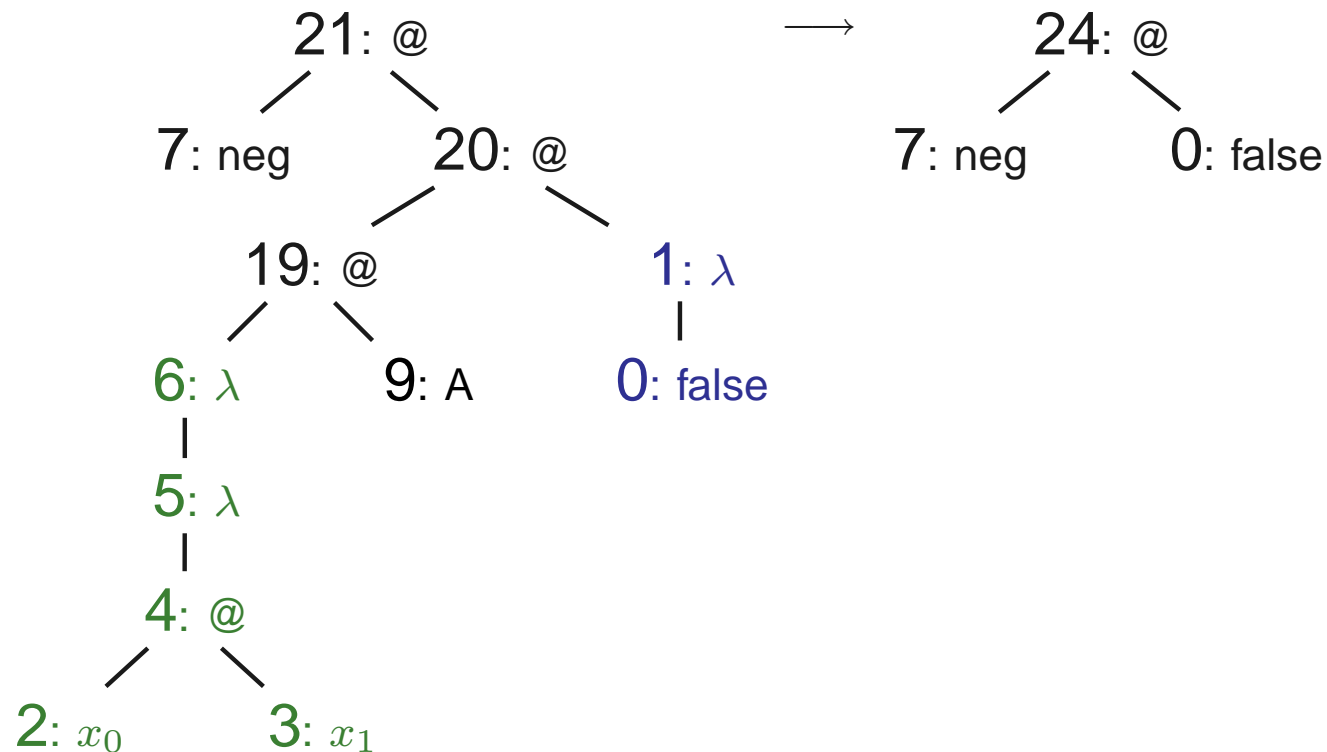


# Partial Syntax Trees and Replacement



## Example

- Normalisation (required before term goes to index again)



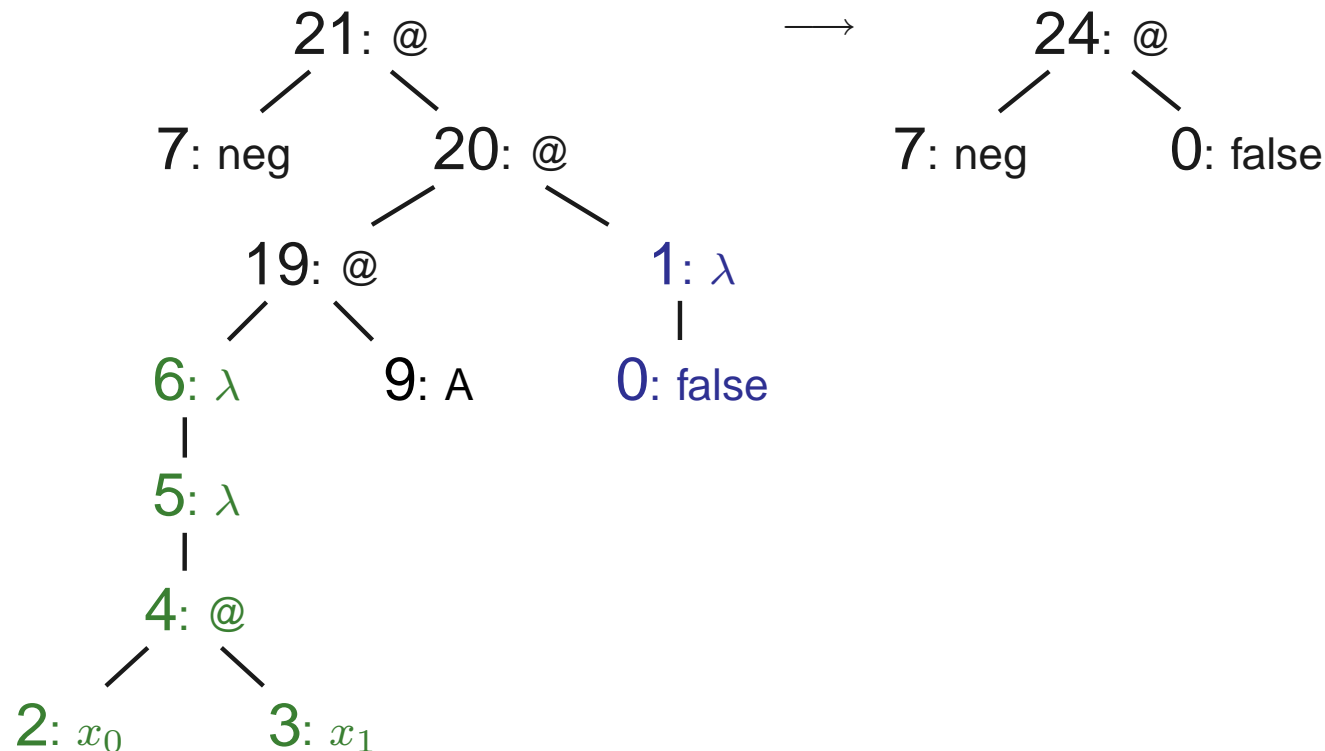
(LEO-II will later immediately say: Proof found!)

# Partial Syntax Trees and Replacement



## Example

- Normalisation (required before term goes to index again)



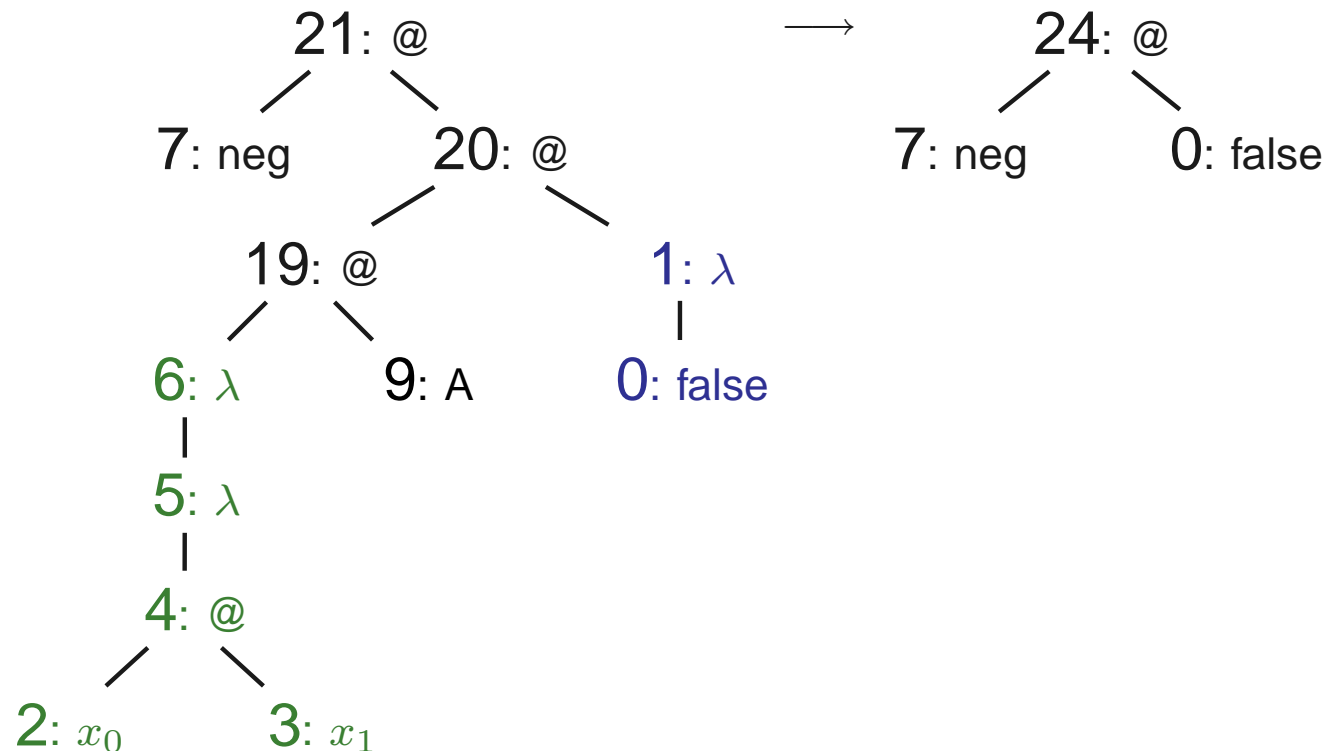
- Hence, we guide replacement & normalization with PSTs

# Partial Syntax Trees and Replacement



## Example

- Normalisation (required before term goes to index again)



- 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  $occurrences : IN \rightarrow IN \rightarrow PST$
- Second hashtable  $occurs\_in : IN \rightarrow IN^*$
- Third hashtable  $occurs\_at : pos \rightarrow IN \rightarrow IN^*$

# 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

Number of indexed terms	977
Number of created term nodes	11618
Average term size	54
Number of nodes with no parent nodes	904
Number of nodes with one parent node	9633
Number of nodes with two more more parent nodes	1083
Maximum number of parent nodes	2778 (symbol $\forall$ )
Average number of parent nodes	1.68
Average number of terms a node occurs in	33.5
"-"(for symbols)	493.9
"-"(for nonprimitive term nodes)	24
Average PST/term size for symbol occurrences	0.21
Average PST/term size for bound variable occurrences	0.33
Average PST/term size for all term nodes	0.12

# 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 [StickeI-89]

# Conclusion



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

# Conclusion



- Presented a term indexing approach that
  - ▶ combines ideas from first order coordinate and path indexing [StickeI-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 [StickeI-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 [StickeI-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 [StickeI-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 [StickeI-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 [StickeI-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 [StickeI-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