Tabled higher-order logic programming

Thesis Proposal

Brigitte Pientka

Department of Computer Science
Carnegie Mellon University
Outline

- Introduction
- Illustrating example: subtyping
- Tabled higher-order logic programming
  1. Tabled logic programming interpreter
  2. Object- and meta-level theorem prover
- Thesis work
- Related work
- Conclusion
Introduction

- Higher-order logic programming
  Terms: (dependently) typed $\lambda$-calculus
 Clauses: implication, universal quantification
Introduction

• Higher-order logic programming
  Terms: (dependently) typed $\lambda$-calculus
  Clauses: implication, universal quantification

• Meta-language for specifying / implementing
  logical systems

  proofs about them
Introduction

- Higher-order logic programming
  Terms: (dependently) typed $\lambda$-calculus
  Clauses: implication, universal quantification

- Meta-language for specifying / implementing
  logical systems (type system, safety logic, congruence closure . . .)
  proofs about them
Introduction

- Higher-order logic programming
  Terms: (dependently) typed $\lambda$-calculus
  Clauses: implication, universal quantification

- Meta-language for specifying / implementing logical systems (type system, safety logic, congruence closure . . .)
  proofs about them (correctness, soundness etc.)
Introduction

• Higher-order logic programming
  Terms: (dependently) typed $\lambda$-calculus
  Clauses: implication, universal quantification

• Meta-language for specifying / implementing
  logical systems (type system, safety logic, congruence closure . . .)
  proofs about them (correctness, soundness etc.)

• Approaches: Elf, $\lambda$Prolog, Isabelle
Generic framework for . . .

- Implementing logical systems
- Executing them and generating certificate
- Checking certificate
- Reasoning with and about them
Generic framework for . . .

- Implementing logical systems
- higher-order logic program
- Executing them and generating certificate
- Checking certificate
- Reasoning with and about them
Generic framework for . . .

• Implementing logical systems
  higher-order logic program

• Executing them and generating certificate
  logic programming interpreter Elf

• Checking certificate

• Reasoning with and about them
Generic framework for . . .

- Implementing logical systems
  higher-order logic program
- Executing them and generating certificate
  logic programming interpreter Elf
- Checking certificate
type checker
- Reasoning with and about them
Generic framework for . . .

- Implementing logical systems
  higher-order logic program
- Executing them and generating certificate
  logic programming interpreter Elf
- Checking certificate
  type checker
- Reasoning with and about them
  object- and meta-level theorem prover Twelf
Generic framework for . . .

- Implementing logical systems
  higher-order logic program
- Executing them and generating certificate
  logic programming interpreter Elf
- Checking certificate
  type checker
- Reasoning with and about them
  object- and meta-level theorem prover Twelf

Reduces the effort required for each logical system
Generic framework for . . .

- Implementing logical systems
  higher-order logic program
- Executing them and generating certificate
  logic programming interpreter Elf
- Checking certificate
  type checker
- Reasoning with and about them
  object- and meta-level theorem prover Twelf

Reduces the effort required for each logical system
Proof search

- Search strategy
  - Depth-first: incomplete, infinite paths
  - Iterative deepening: complete, infinite paths
**Proof search**

- Search strategy
  - Depth-first: incomplete, infinite paths
  - Iterative deepening *with bound*: incomplete, infinite paths
Proof search

• Search strategy
  Depth-first: incomplete, infinite paths
  Iterative deepening with bound: incomplete, infinite paths

• Performance
  Redundant computation
Tabled logic programming

- Tabling, memoization, caching, loop detection, magic sets ...
- Eliminate infinite and redundant computation by memoization (Tamaki, Sato)
- Finds all possible answers to a query
- Terminates for programs in a finite domain
- Combines tabled and non-tabled execution
- Very successful: XSB system (Warren et al.)
Tabled higher-order logic programming allows us to

- efficiently execute logical systems and

- automate the reasoning with and about them.
Tabled higher-order logic programming allows us to

- efficiently execute logical systems and
  (interpreter using tabled search)

- automate the reasoning with and about them.
Tabled higher-order logic programming allows us to

- efficiently execute logical systems and
  (interpreter using tabled search)
- automate the reasoning with and about them.
  (theorem prover using tabled search)
Illustrating example: subtyping

Types \( \tau \) ::= neg | zero | pos | nat | int
Illustrating example: subtyping

Types \( \tau \ ::= \ neg \mid zero \mid pos \mid nat \mid int \)

\[
\begin{align*}
\text{zero} & \triangleleft \text{nat} \\
\text{pos} & \triangleleft \text{nat}
\end{align*}
\]
Illustrating example: subtyping

Types $\tau ::= \text{neg} | \text{zero} | \text{pos} | \text{nat} | \text{int}$

$\text{zero} \leq \text{nat}$  $\text{pos} \leq \text{nat}$  $\text{nat} \leq \text{int}$  $\text{neg} \leq \text{int}$
Illustrating example: subtyping

Types \( \tau ::= \text{neg} \mid \text{zero} \mid \text{pos} \mid \text{nat} \mid \text{int} \)

\[
\begin{align*}
\text{zero} & \leq \text{nat} \\
\text{pos} & \leq \text{nat} \\
\text{nat} & \leq \text{int} \\
\text{neg} & \leq \text{int}
\end{align*}
\]

\[
\begin{align*}
\text{refl} & \quad T \leq T \\
\text{tr} & \quad T \leq R \\
& \quad R \leq S \\
& \quad T \leq S
\end{align*}
\]
Subtyping relation in Elf

refl : sub \( T T \).
tr : sub \( T S \)
    \( \leftarrow \) sub \( T R \)
    \( \leftarrow \) sub \( R S \).
zn : sub zero nat .
pn : sub pos nat .
nati : sub nat int .
negi : sub neg int .
Subtyping relation in Elf

refl : sub $T$ $T$.
tr : sub $T$ $S$
    ← sub $T$ $R$
    ← sub $R$ $S$.
zn : sub zero nat.
pn : sub pos nat.
nati : sub nat int.
negi : sub neg int.

Compute all supertypes of zero
: − ? sub zero $T$. 
Subtyping relation in Elf

refl : sub $T T$.
tr : sub $T S$
    ← sub $T R$
    ← sub $R S$.
zn : sub zero nat.
pn : sub pos nat.
nati : sub nat int.
negi : sub neg int.

Compute all supertypes of zero

: – ? sub zero $T$
refl: $T = \text{zero}$
Success
Subtyping relation in Elf

refl : sub $T \ T$.

tr : sub $T \ S$
    $\leftarrow$ sub $T \ R$
    $\leftarrow$ sub $R \ S$.

zn : sub zero nat .

pn : sub pos nat .

nati : sub nat int .

negi : sub neg int .

Compute all supertypes of zero

: –? sub zero $T$.

tr: sub zero $R \leftarrow$ sub $R \ T$. 

Tabled higher-order logic programming – p.9/30
Subtyping relation in Elf

refl : sub $T T$.

tr : sub $T S$

$\leftarrow$ sub $T R$

$\leftarrow$ sub $R S$.

zn : sub zero nat.

pn : sub pos nat.

nati : sub nat int.

negi : sub neg int.

Compute all supertypes of zero

: $\leftarrow$ ? sub zero $T$.

tr : sub zero $R$ $\leftarrow$ sub $R T$.

refl : sub zero $T$
Subtyping relation in Elf

refl : \( \text{sub } T \: T \).
tr : \( \text{sub } T \: S \)
    \( \leftarrow \text{sub } T \: R \)
    \( \leftarrow \text{sub } R \: S \).
zn : \( \text{sub zero } \text{nat} \).
pn : \( \text{sub pos } \text{nat} \).
nati : \( \text{sub nat } \text{int} \).
negi : \( \text{sub neg } \text{int} \).

Compute all supertypes of zero
\( : \: ? \: \text{sub zero } T \).
tr: \( \text{sub zero } R \leftarrow \text{sub } R \: T \).
refl: \( \text{sub zero } T \)
refl: \( T = \text{zero} \)

Redundant answer
Subtyping relation in Elf

refl : sub $T T$.  
tr : sub $T S$
    <- sub $T R$
    <- sub $R S$.  
zn : sub zero nat .  
pn : sub pos nat .  
nati : sub nat int .  
negi : sub neg int .

Compute all supertypes of zero:

: - ? sub zero $T$.
tr: sub zero $R$ <- sub $R T$.
refl: sub zero $T$
tr: sub zero $R$ <- sub $R T$.  

Tabled higher-order logic programming – p.9/30
Subtyping relation in Elf

refl : sub $T \ S$.
tr : sub $T \ S$  
   ← sub $T \ R$
   ← sub $R \ S$.
zn : sub zero nat .
pn : sub pos nat .
nati : sub nat int .
negi : sub neg int .

Compute all supertypes of zero

: - ? sub zero $T$.
tr: sub zero $R$ ← sub $R \ T$.
reel: sub zero $T$
tr: sub zero $R$ ← sub $R \ T$.

Infinite path
Problem

- Redundant computation
- Infinite computation
- Non-termination instead of failure
- Sensitive to clause ordering
- Independent of the actual search strategy
Proof search

- Logic programming
  Depth-first

- Object-level theorem proving
  Iterative deepening with bound

- Meta-level theorem proving:
  Induction + case analysis + iterative deepening
Proof search

- Logic programming
  Depth-first
  program clauses

- Object-level theorem proving
  Iterative deepening with bound

- Meta-level theorem proving:
  Induction + case analysis + iterative deepening
Proof search

- Logic programming
  Depth-first
  program clauses

- Object-level theorem proving
  Iterative deepening with bound
  program clauses + lemmas

- Meta-level theorem proving:
  Induction + case analysis + iterative deepening
Proof search

- Logic programming
  - Depth-first
  - program clauses
- Object-level theorem proving
  - Iterative deepening with bound
  - program clauses + lemmas
- Meta-level theorem proving:
  - Induction + case analysis + iterative deepening
  - program clauses + lemmas + proof assumptions
Tabled logic programming

- Eliminate redundant and infinite paths from proof search using memoization
- Table:
  1. Record encountered sub-goals
  2. Store corresponding solutions
\%tabled sub .
refl : sub \(T T\).
tr : sub \(T S\)
    \(\leftarrow \) sub \(T R\)
    \(\leftarrow \) sub \(R S\).
zn : sub zero nat .
pn : sub pos nat .
nati : sub nat int .
negi : sub neg int .
Compute all supertypes of zero $	exttt{:–? sub zero } T$. 

Entry | Answer
---|---

**0** Entry

## Tabled computation

%\textit{tabled} sub.

\begin{align*}
\text{refl} : & \text{ sub } T T. \\
\text{tr} : & \text{ sub } T S \\
& \quad \leftarrow \text{ sub } T R \\
& \quad \leftarrow \text{ sub } R S. \\
\text{zn} : & \text{ sub zero nat.} \\
\text{pn} : & \text{ sub pos nat.} \\
\text{nati} : & \text{ sub nat int.} \\
\text{negi} : & \text{ sub neg int.}
\end{align*}
Compute all supertypes of zero

\[ \text{refl:} \quad T = \text{zero} \]

\[ \text{Success!} \]

%tabled sub.

refl : sub \( T \ T \).

tr : sub \( T \ S \)
    \[ \leftarrow \text{sub } T \ R \]
    \[ \leftarrow \text{sub } R \ S. \]

zn : sub zero nat.

pn : sub pos nat.

nati : sub nat int.

negi : sub neg int.

<table>
<thead>
<tr>
<th>Entry</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub zero ( T )</td>
<td></td>
</tr>
</tbody>
</table>
Tabled computation

Compute all supertypes of zero

%tabled sub.
refl : sub \( T \ T \).
tr : sub \( T \ S \)
    \( \leftarrow \) sub \( T \ R \)
    \( \leftarrow \) sub \( R \ S \).
zn : sub zero nat .
pn : sub pos nat .
nati : sub nat int .
negi : sub neg int .

Add answer to table

<table>
<thead>
<tr>
<th>Entry</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub zero ( T )</td>
<td>[zero ( /T )]</td>
</tr>
</tbody>
</table>
%tabled sub.

refl : sub $T \hspace{0.5em} T$.

tr : sub $T \hspace{0.5em} S$
    \leftarrow sub $T \hspace{0.5em} R$
    \leftarrow sub $R \hspace{0.5em} S$.

zn : sub zero nat.

pn : sub pos nat.

nati : sub nat int.

negi : sub neg int.

Compute all supertypes of zero

):- ? sub zero $T$.

tr : sub zero $R$ \leftarrow sub $R \hspace{0.5em} T$.

Variant of previous goal

<table>
<thead>
<tr>
<th>Entry</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub zero $T$</td>
<td>[zero /$T$]</td>
</tr>
</tbody>
</table>
Tabled computation

%tabled sub.

refl : sub $T \, T$.

tr : sub $T \, S$
       ← sub $T \, R$
       ← sub $R \, S$.

zn : sub zero nat.

pn : sub pos nat.

nati : sub nat int.

negi : sub neg int.

Compute all supertypes of zero

: −? sub zero $T$.

tr : sub zero $R ← sub R \, T$.

Fail and suspend goal

<table>
<thead>
<tr>
<th>Entry</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub zero $T$</td>
<td>[zero / $T$]</td>
</tr>
</tbody>
</table>
\%tabled sub.

refl : sub \( T T \).

tr : sub \( T S \)
    \( \leftarrow \) sub \( T R \)
    \( \leftarrow \) sub \( R S \).

zn : sub zero nat.

pn : sub pos nat.

nati : sub nat int.

negi : sub neg int.

Compute all supertypes of zero
: - ? sub zero \( T \).

zn : \( T = \text{nat} \)

Success!

\begin{tabular}{c|c}
Entry & Answer \\
\hline
sub zero \( T \) & \( \left[ \text{zero} / T \right] \)
\end{tabular}
Compute all supertypes of zero

:─? sub zero T.

zn : T = nat

Add answer to table

<table>
<thead>
<tr>
<th>Entry</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub zero T</td>
<td>[zero /T], [nat /T]</td>
</tr>
</tbody>
</table>

Tabled higher-order logic programming – p.13/30
Compute all supertypes of `zero`:

```prolog
%tabled sub .
refl : sub T T.
tr : sub T S
    ← sub T R
    ← sub R S.
zn : sub zero nat .
pn : sub pos nat .
nati : sub nat int .
negi : sub neg int .
```

Add answer to table

<table>
<thead>
<tr>
<th>Entry</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub zero T</td>
<td>[zero /T], [nat /T]</td>
</tr>
</tbody>
</table>

First Stage completed!
Compute all supertypes of zero

%\textit{tabled} sub.

\texttt{refl : sub }T T. \\
\texttt{tr : sub }T S \leftarrow \texttt{sub }T R \\
\leftarrow \texttt{sub }R S.

\texttt{zn : sub zero nat.} \\
\texttt{pn : sub pos nat.} \\
\texttt{nati : sub nat int.} \\
\texttt{negi : sub neg int.}

\begin{tabular}{l|l}
\textbf{Entry} & \textbf{Answer} \\
\hline
\texttt{sub zero }T & \{\texttt{zero }/T\}, \{\texttt{nat }/T\} \\
\end{tabular}
Compute all supertypes of zero:

\[ T \] \leftarrow \text{sub} \ T \] \leftarrow \text{sub} \ T \] \leftarrow \text{sub} \ R \] \leftarrow \text{sub} \ R \ S.

<table>
<thead>
<tr>
<th>Entry</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub zero ( T )</td>
<td>[zero /( T )], [nat /( T )]</td>
</tr>
</tbody>
</table>
Compute all supertypes of \( \text{zero} \):

\[
\text{refl} : \quad \text{sub } T \ T.
\]

\[
\text{tr} : \quad \text{sub } T \ S \quad \leftarrow \quad \text{sub } T \ R
\]

\[
\quad \leftarrow \quad \text{sub } R \ S.
\]

\[
\text{zn} : \quad \text{sub } \text{zero } \text{nat}.
\]

\[
\text{pn} : \quad \text{sub } \text{pos } \text{nat}.
\]

\[
\text{nati} : \quad \text{sub } \text{nat } \text{int}.
\]

\[
\text{negi} : \quad \text{sub } \text{neg } \text{int}.
\]

Add goal to table

<table>
<thead>
<tr>
<th>Entry</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub zero ( T )</td>
<td>[zero /( T )], [nat /( T )]</td>
</tr>
<tr>
<td>sub nat ( T )</td>
<td></td>
</tr>
</tbody>
</table>
%tabled sub.

refl : sub T T.
tr : sub T S 
    <- sub T R 
    <- sub R S.
zn : sub zero nat .

pn : sub pos nat .
nati : sub nat int .
negi : sub neg int .

Compute all supertypes of zero

: - ? sub zero T.

resume sub zero R <- sub R T.
[nat / R] sub nat T
refl T = nat

Success

<table>
<thead>
<tr>
<th>Entry</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub zero T</td>
<td>[zero / T], [nat / T]</td>
</tr>
<tr>
<td>sub nat T</td>
<td></td>
</tr>
</tbody>
</table>
%tabled sub.

refl : sub $T \ T$.

tr : sub $T \ S$

\[ \leftarrow \text{sub} \ T \ R \]

\[ \leftarrow \text{sub} \ R \ S. \]

zn : sub zero nat.

pn : sub pos nat.

nati : sub nat int.

negi : sub neg int.

Compute all supertypes of zero


resume sub zero $R \leftarrow \text{sub} \ R \ T$.

\[ [\text{nat} / R] \quad \text{sub nat} \ T \]

refl $T = \text{nat}$

Add answer to table

<table>
<thead>
<tr>
<th>Entry</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub zero $T$</td>
<td>[zero /$T$], [nat /$T$]</td>
</tr>
<tr>
<td>sub nat $T$</td>
<td>[nat /$T$]</td>
</tr>
</tbody>
</table>
%tabled sub .

refl : sub $T \rightarrow T$.

tr : sub $T \rightarrow S$.

\[ \leftarrow \text{sub } T \rightarrow R \]
\[ \leftarrow \text{sub } R \rightarrow S. \]

zn : sub $\text{zero nat}$.

pn : sub $\text{pos nat}$.

nati : sub $\text{nat int}$.

negi : sub $\text{neg int}$.

Compute all supertypes of $\text{zero}$:

\[ :-? \text{ sub zero } T. \]

<table>
<thead>
<tr>
<th>Entry</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub $\text{zero } T$</td>
<td>$[\text{zero }/T], [\text{nat }/T], [\text{int }/T]$</td>
</tr>
<tr>
<td>sub $\text{nat } T$</td>
<td>$[\text{nat }/T], [\text{int }/T]$</td>
</tr>
<tr>
<td>sub $\text{int } T$</td>
<td>$[\text{int }/T]$</td>
</tr>
</tbody>
</table>
• When to suspend goals?
Strategy

- When to suspend goals?
- When to retrieve answers?
Strategy

• When to suspend goals?
• When to retrieve answers?
• How to retrieve answers (order)?
Strategy

- When to suspend goals?
- When to retrieve answers?
- How to retrieve answers (order)?
- What is the retrieval condition?
Strategy

- When to suspend goals?
- When to retrieve answers?
- How to retrieve answers (order)?
- What is the retrieval condition?

Multi-stage strategy:
only re-use answers from previous stages
Advantages

- Translating inference rules to logic program is straightforward.
- Programs have better complexities.
- Order of clauses is less important.
- Computation will terminate for finite domain.
- We can dis-prove more conjectures.
- Table contains useful debugging information.
Trade-off

Price to pay:

- More complicated semantics
- Overhead caused by memoization
Trade-off

Price to pay:

- More complicated semantics
- Overhead caused by memoization

Solution:

- Combine tabled and non-tabled proof search
- Make table access efficient: term indexing
Typing rules

Mini ML  \[ e ::= n(e) \mid z \mid s(e) \mid \text{app } e_1 e_2 \mid \]
\[ \text{lam } x.e \mid \text{letn } u = e_1 \text{ in } e_2 \]

\[ \Gamma \vdash e : \tau' \quad \tau' \preceq \tau \]
\[ \frac{}{\Gamma \vdash e : \tau} \quad \text{tp-sub} \]

\[ \Gamma \vdash \text{letn } u = e_1 \text{ in } e_2 : \tau \]

\[ \Gamma, x : \tau_1 \vdash e : \tau_2 \]
\[ \frac{}{\Gamma \vdash \text{lam } x.e : \tau_1 \rightarrow \tau_2} \quad \text{tp-lam} \]

\[ \Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash [e_1/u]e_2 : \tau \]
\[ \frac{}{\Gamma \vdash \text{letn } u = e_1 \text{ in } e_2 : \tau} \quad \text{tp-letn} \]
**Type Checker in Elf**

\[\text{tp}\cdot\text{sub} : \text{of } E T \quad \text{tp}\cdot\text{lam} : \text{of } (\text{lam } ([x] E x)) (T_1 \Rightarrow T_2)\]
\[\quad \leftarrow \text{of } E T' \quad \leftarrow (\{y\}\text{of } y T_1 \rightarrow \text{of } (E y) T_2).\]
\[\quad \leftarrow \text{sub } T' T.\]

\[\text{tp}\cdot\text{letn} : \text{of } (\text{letn } E_1 ([u] E_2 u)) T\]
\[\quad \leftarrow \text{of } E_1 T_1\]
\[\quad \leftarrow \text{of } (E_2 E_1) T.\]
### Tabled computation (higher-order)

\[
: - ? \text{ of (lam \ (} \left[ x \right] \ x) \) } T
\]

<table>
<thead>
<tr>
<th>Entry</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>of (lam \ (} \left[ x \right] \ x) ) } T</td>
<td></td>
</tr>
</tbody>
</table>
Tabled computation (higher-order)

\[ : - ? \text{ of } (\text{lam } ([x] x)) T \]

\[ \text{tp} \cdot \text{sub}: \text{ of } (\text{lam } ([x] x)) \quad R \leftarrow \text{sub } R T. \]

<table>
<thead>
<tr>
<th>Entry</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>of (lam ([x] x)) T</td>
<td></td>
</tr>
</tbody>
</table>
Tabled computation (higher-order)

: – ? of (lam ([x] x)) T

\[ \text{tp\_sub} \text{ of (lam ([x] x)) R \leftarrow \text{sub R T}}. \]

Variant of previous goal

<table>
<thead>
<tr>
<th>Entry</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>of (lam ([x] x)) T</td>
<td></td>
</tr>
</tbody>
</table>
Tabled computation (higher-order)

\[ \text{tp} \cdot \text{sub} : \text{of} \ (\text{lam} \ ([x] \ x)) \ T \]

Fail and suspend

<table>
<thead>
<tr>
<th>Entry</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>of ( \text{lam} \ ([x] \ x) ) ( T )</td>
<td></td>
</tr>
</tbody>
</table>
Tabled computation (higher-order)

: - ? of (lam (\([x] x\))) \(T\)

\(\text{tp\_lam: } u : \text{of } x \ T_1 \vdash \text{of } x \ T_2\)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>of (lam (([x] x))) (T)</td>
<td></td>
</tr>
</tbody>
</table>
Tabled computation (higher-order)

: – ? of (lam ([x] x)) $T$

tp\_lam: $u: \text{of } x \ T_1 \vdash \text{of } x \ T_2$

Add goal to table

<table>
<thead>
<tr>
<th>Entry</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u: \text{of } x \ T_1 \vdash \text{of } x \ T_2$</td>
<td></td>
</tr>
</tbody>
</table>
Tabled computation (higher-order)

\[ : - ? \quad \text{of} \quad \text{lam} \quad ([x] \ x) \quad T \]

\[
\text{tp} \cdot \text{lam}: \quad u : \quad \text{of} \quad x \quad T_1 \vdash \text{of} \quad x \quad T_2 \\
\]
\[ u : \quad T_1 = P, \quad T_2 = P, \quad T = (P \Rightarrow P) \]

Success

<table>
<thead>
<tr>
<th>Entry</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>of (lam ([x] x)) T</td>
<td></td>
</tr>
<tr>
<td>( u : \quad \text{of} \quad x \quad T_1 \vdash \text{of} \quad x \quad T_2 )</td>
<td></td>
</tr>
</tbody>
</table>
### Tabled computation (higher-order)

\[ : \neg \ ? \ of \ (\text{lam} \ ([x] \ x)) \ T \]

\text{tp.lam:} \ u : \ of \ x \ T_1 \vdash \ of \ x \ T_2 \\
\text{u:} \quad T_1 = P, \ T_2 = P, \ T = (P \Rightarrow P) \\
\text{Add answers to table}

<table>
<thead>
<tr>
<th>Entry</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>of \ (\text{lam} \ ([x] \ x)) \ T</td>
<td>[(P \Rightarrow P)/T]]</td>
</tr>
<tr>
<td>\text{u:} \ of \ x \ T_1 \vdash \ of \ x \ T_2</td>
<td>[P/T_1, \ P/T_2]</td>
</tr>
</tbody>
</table>
Tabled computation (higher-order)

\[\text{tp´lam: } u : \text{of } x \; T_1 \vdash \text{of } x \; T_2\]
\[\text{tp´sub: } u : \text{of } x \; T_1 \vdash \text{of } x \; R \leftarrow \text{sub } R \; T_2\]

<table>
<thead>
<tr>
<th>Entry</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>of (lam (([x] ; x)) ; T)</td>
<td>([\text{(P } \Rightarrow ; \text{P})/T])</td>
</tr>
<tr>
<td>(u : \text{of } x ; T_1 \vdash \text{of } x ; T_2)</td>
<td>([P/T_1, ; P/T_2])</td>
</tr>
</tbody>
</table>
Tabled computation (higher-order)

: - ? of (lam ([x] x)) T

tp.lam: u : of x T₁ ⊨ of x T₂

tp.sub: u : of x T₁ ⊨ of x R ← sub R T₂

Variant of previous goal

<table>
<thead>
<tr>
<th>Entry</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>of (lam ([x] x)) T</td>
<td>([P ⇒ P)/T]</td>
</tr>
<tr>
<td>u : of x T₁ ⊨ of x T₂</td>
<td>[P/T₁, P/T₂]</td>
</tr>
</tbody>
</table>
Tabled computation (higher-order)

: – ? of (lam ([x] x)) T

tp.lam: u : of x T₁ ⊸ of x T₂

tp.sub: u : of x T₁ ⊸ of x R ← sub R T₂

Suspend and fail

<table>
<thead>
<tr>
<th>Entry</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>of (lam ([x] x)) T</td>
<td>[(P ⇒ P)/T]</td>
</tr>
<tr>
<td>u : of x T₁ ⊸ of x T₂</td>
<td>[P/T₁, P/T₂]</td>
</tr>
</tbody>
</table>
Challenges

• Store goals together with context: \( \Gamma \vdash a \)

• Redesign table operations: goal \((\Gamma \vdash a) \in \text{Table}\)

• Context dependencies
  (e.g. \( u : \text{of} x T_1 \vdash \text{sub} R T_2 \))

• Type dependencies
  (e.g. \( u : \text{of} x T_1 \vdash \text{of} x (R x u) \))

• Indexing for higher-order terms
Meta-level reasoning

Iterative deepening with depth bound

Case Splitting

proof assumptions

Induction hypothesis generation

induction hypothesis
Meta-level reasoning

Iterative deepening

program clauses
lemmas
proof assumptions
induction hypothesis

depth

Case Splitting

proof assumptions

induction hypothesis

Induction hypothesis generation
Meta-level reasoning

- Iterative deepening
  - program clauses
  - lemmas
  - proof assumptions
  - induction hypothesis

- Case Splitting
  - \( c_1 \), \( c_2 \), \( \ldots \), \( c_6 \), \( c_7 \), \( \ldots \)

- Induction hypothesis
  - induction hypothesis generation
Meta-level reasoning

Drawbacks:

- No sharing across iterations
- Focus on one split
- No sharing across cases
- No usefull failure

Induction hypothesis generation

Iterative deepening

- Program clauses
- Lemmas
- Proof assumptions
- Induction hypothesis

Case Splitting

\[ c_1 \quad c_2 \quad \ldots \quad c_6 \quad c_7 \quad \ldots \]

Proof assumptions

Tabled higher-order logic programming – p.21/30
Meta-level reasoning with tabling

Tabled proof search

Table

program clauses
lemmas
proof assumptions
induction hypothesis

Case Splitting

\[ c_1, c_2, \ldots, c_6, c_7, \ldots \]

proof assumptions

Induction hypothesis generation

induction hypothesis
Meta-level reasoning with tabling

Tabled proof search
- program clauses
- lemmas
- proof assumptions
- induction hypothesis

Case Splitting
- c1, c2
- . . .
- c6, c7
- . . .

Induction hypothesis generation

induction hypothesis

Table

Tabled higher-order logic programming – p.21/30
Meta-level reasoning with tabling

Tabled proof search
- program clauses
- lemmas
- proof assumptions
- induction hypothesis

Case Splitting
- \( c_1 \)
- \( c_2 \)
- \( \ldots \)
- \( c_6 \)
- \( c_7 \)
- \( \ldots \)

proof assumptions

Induction hypothesis generation

Table

induction hypothesis
Meta-level search based on tabling

- Redundancy elimination during object-level search
- Detection of unprovable branches
- Preservation of partial results across case splitting and induction hypothesis generation
- Proving different case split in parallel
- Detection of redundant case splits
Overview of Thesis

• Proof-theoretical characterization: Soundness of interpreter
• Design of efficient implementation techniques
  1. Higher-order terms indexing
  2. Context handling
• Implementation and Validation
  1. Logic programming
  2. Object and meta-level theorem proving
Preliminary Experiments

- Specification (formerly not executable)
  - Type systems: subtyping, intersections
  - Rewriting based on $\lambda$-calculus
  - Conversions in the $\lambda$-calculus
  - Graph transition systems

- Implementations: better performance
  - Refinement types
  - Polymorphisms
Other examples

Logical systems:

- Cartesian closed categories (CCC)
- Natural deduction calculi (NK, NJ)
- Decision procedures (e.g. congruence closure algorithms)
- Parsing grammars

Examples for meta-reasoning:

- Soundness of Kolmogoroff translation between NK and NJ
- Translation between CCC and λ-calculus
Related Work

Tabled first-order logic programming:
- SLD resolution with memoization (Tamaki, Sato)
- Extensions to WAM (Warren, Chen)

Object and meta-level reasoning:
- Based on tactics: Isabelle (Paulson), \lambda Prolog (Felty, Miller)
- Based on higher-order logic programming: Twelf (Schürmann, Pfenning)
Related Work

Proof-theoretical characterization

- Uniform proofs (Miller, Nadathur, Pfenning, Scedrov)
- Proof Irrelevance (Pfenning)

Implementation techniques (mainly first-order)

- Term indexing (I.V.Ramakrishnan, Sekar, Voronkov)
- Substitution trees (Graf), higher-order (Klein)
Certificates:

- Justifiers: XSB (Roychoudhury, I.V.Ramakrishnan)
- Bit-strings: variant of PCC (Necula, Rahul)
- Proof terms: *Elf*, *Twelf* (Schürmann, Pfenning)
Conclusion

- Tabled higher-order logic programming
- Tabled proof search impacts
  1. Logic programming interpreter
  2. Object- and meta-level theorem prover
- Proof-theoretic characterization
- Implementation of prototype
- Preliminary experiments