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
  - Tabled logic programming interpreter
  - Object- and meta-level theorem prover
- Thesis work
- Related work
- Conclusion
Outline

• Introduction
• Illustrating example: subtyping
• Tabled higher-order logic programming
  – Tabled logic programming interpreter
  – 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
- Executing them and generating certificate
  higher-order logic program
- 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 tree

- Search Strategy
  - Depth-first: incomplete, infinite paths
  - Iterative deepening: complete, infinite paths
- Performance: redundant computation
Proof search tree

- Search Strategy
  - Depth-first: incomplete, infinite paths
  - Iterative deepening: complete, infinite paths

- Performance: redundant computation
Proof search tree

• Search Strategy
  – Depth-first: incomplete, infinite paths
  – Iterative deepening: complete, infinite paths

• Performance: redundant computation
Tabled evaluation for Prolog

- 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.* )
This talk

1. Extend tabled logic programming to higher-order
2. Demonstrate the use of tabled search to
   • efficiently execute logical systems
   • automate reasoning with and about them.
This talk

1. Extend tabled logic programming to higher-order
2. Demonstrate the use of tabled search to
   • efficiently execute logical systems
     (interpreter using tabled search)
   • automate reasoning with and about them.
1. Extend tabled logic programming to higher-order
2. Demonstrate the use of tabled search to
   • efficiently execute logical systems
     (interpreter using tabled search)
   • automate reasoning with and about them.
     (theorem prover using tabled search)
Illustrating example: subtyping

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

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

\[ \underline{zn} \quad \underline{pn} \]

\( zero \preceq nat \quad pos \preceq nat \)
Illustrating example: subtyping

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

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

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

$\text{zn}$
zero $\leq$ nat

$\text{pn}$
pos $\leq$ nat

$\text{nati}$
nat $\leq$ int

$\text{negi}$
neg $\leq$ int

$\text{refl}$
$T \leq T$

$T \leq R$
$R \leq S$

$T \leq S$
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 .
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:
Compute all supertypes of zero

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.

refl : \( T = \text{zero} \)
Success
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
$\vdash \text{sub zero } T$.
tr: sub zero $R$ ; sub $R$ $T$. 
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 $R$ ; sub $R \ T$.
refl: sub zero $T$
Subtyping relation in Elf

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

: -? sub zero $T$.
tr: sub zero $R$; sub $R T$.
refl: 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.10/40
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:
$\text{: } \text{-- } ? \text{ sub zero } T.$

tr: sub zero $R$ ; sub $R T$.
refl: sub zero $T$
tr: sub zero $R$ ; sub $R T$.

Infinite path
Problem

- Redundant and 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. Store sub-goals
  2. Store solutions
  3. Retrieve solutions
- Depth-first multi-stage strategy
%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:

\[ \text{Entry} \quad \text{Answer} \]

\[ \text{sub zero } T \]
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 .

Entry | Answer
----- | ------
sub zero \( T \) | Success!
Tabled computation

\%
\textit{tabled} sub.

refl : sub \(T \, T\).

tr : sub \(T \, S\)

\quad \leftarrow \text{sub} \, \(T \, R\)

\quad \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

: –? sub zero \(T\).

refl: \(T = \text{zero}\)

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 higher-order logic programming – p.14/40
%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 $T$.
tr : sub zero $R$ ; sub $R \ T$.

Variant of previous goal

Entry | Answer
--- | ---
sub zero $T$ | [zero / $T$]
Tabled computation

%tabled sub.

refl : sub $T$ $T$.

tr : sub $T$ $S$
    left sub $T$ $R$
    left 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 computation

Compute all supertypes of zero

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 .

<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 $T$ $S$
    \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 : $T = \text{nat}$

Add answer to table

<table>
<thead>
<tr>
<th>Entry</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub zero $T$</td>
<td>[\text{zero} / T], [\text{nat} / T]</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 $T$.
zn: $T = \text{nat}$

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>
</tbody>
</table>

First Stage completed!
Compute all supertypes of zero:

\[ \text{sub zero } T. \]

\[ \text{resume sub zero } R; \text{ sub } R T. \]

<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 zero

: – ? sub zero T.

resume sub zero R ; sub R T.

[ nat / R ] sub nat T.

%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 .

Entry | Answer
---|---
sub zero T | [ zero / T ], [ nat / T ]
Compute all supertypes of zero

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 .
egi : sub neg int .

Add goal to table

Compute all supertypes of zero

\[-?\] sub zero $T$.
resume sub zero $R$ ; sub $RT$.

\[\text{nat} / R\] sub nat $T$.

Entry | Answer
---|---
sub zero $T$ | [zero / $T$], [nat / $T$]
sub nat $T$ |
%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 $T$.

resume sub zero $R$ ; sub $R T$.

[nat / $R$] sub nat $T$

refl $T = \text{nat}$

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 computation

Compute all supertypes of zero

<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 T.
tr : sub T S
     ← sub T R
     ← sub R S.
zn : sub zero nat .
pn : sub pos nat .
nati : sub nat int .
egi : sub neg int .
Compute all supertypes of zero

%tabled sub.
refl : sub $T \times T$.
tr : sub $T \times S$
    \leftarrow sub $T \times R$
    \leftarrow sub $R \times S$.
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>{$zero/T$, $nat/T$, $int/T$}</td>
</tr>
<tr>
<td>sub nat $T$</td>
<td>{$nat/T$, $int/T$}</td>
</tr>
<tr>
<td>sub int $T$</td>
<td>{$int/T$}</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?
  - Variant
  - Subsumption
Strategy

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

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 find all answers to a query.
- 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
- Term indexing:
  1. Make table access efficient
  2. Make storage space small
First-order tabled logic programming

- Tabled logic programming
  - atomic subgoals
  - untyped first-order terms
- Procedural descriptions of tabling
  - SLD resolution with memoization (Tamaki, Sato)
  - SLG resolution (Warren, Chen)
- Term indexing (I.V.Ramakrishnan, Sekar, Voronkov)
  discrimination tries, substitution trees, path indexing
First-order tabled logic programming

• Tabled logic programming
  – atomic subgoals
  – untyped first-order terms
• Procedural descriptions of tabling
  – SLD resolution with memoization (Tamaki, Sato)
  – SLG resolution (Warren, Chen)
• Term indexing (I.V. Ramakrishnan, Sekar, Voronkov)
  discrimination tries, substitution trees, path indexing
Outline

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

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

• Extend tabling to higher-order
  1. Terms: dependently typed λ-calculus
  2. Clauses: implications, universal quantification
• Apply tabled search to
  1. higher-order logic programming
  2. object- and meta-level theorem proving
**Typing rules**

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

\[
\begin{align*}
\Gamma \vdash e : \tau' & \quad \tau' \leq \tau \\
\hline
\Gamma \vdash e : \tau & \quad \text{tp-sub}
\end{align*}
\]

\[
\begin{align*}
\Gamma, x : \tau_1 \vdash e : \tau_2 & \\
\hline
\Gamma \vdash \text{lam}\ x.e : \tau_1 \rightarrow \tau_2 & \quad \text{tp-lam}
\end{align*}
\]

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

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

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

:– ? of (lam ([x] x)) 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
tp-sub: of (lam ([x] x)) R ; 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

tp-sub: of (lam ([x] x)) R ; 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)

\[
: - \ ? \ \text{of} \ \text{lam} \ \left( [x] \ x \right) \ T \\
\text{tp-sub: of} \ \text{lam} \ \left( [x] \ x \right) \ R \ ; \ \text{sub} \ R \ T.
\]

Fail and suspend

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

\[ \text{tp-lam: } u : \text{of } x \ T_1 \vdash \text{of } x \ T_2 \]

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

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

tp-lam: u : of x T₁ ⊢ of x T₂

Add goal to table

<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 : of x T₁ ⊢ of x T₂</td>
<td></td>
</tr>
</tbody>
</table>
Tabled computation (higher-order)

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

tp-lam: u : of x T₁ ⊨ of x T₂

u:

\[ T_1 = P, \quad T_2 = P, \quad T = (P \Rightarrow P) \]

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 : of x T₁ ⊨ of x T₂</td>
<td></td>
</tr>
</tbody>
</table>
Tabled computation (higher-order)

: –? of (lam ([x] x)) T

tp-lam: u : of x T₁ ⊢ of x T₂

u:

\[ T₁ = P, \quad T₂ = P, \quad T = (P \Rightarrow P) \]

Add answers to table

<table>
<thead>
<tr>
<th>Entry</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>of (lam ([x] x)) T</td>
<td>([(P \Rightarrow 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)

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

\[
\text{tp-lam:} \ u : \ \text{of} \ x \ T_1 \vdash \ \text{of} \ x \ T_2
\]

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

<table>
<thead>
<tr>
<th>Entry</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{of} \ (\text{lam} \ ([x] \ x)) \ T</td>
<td>[(P \Rightarrow P)/T]]</td>
</tr>
<tr>
<td>\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)

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

\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 \ ; \ \text{sub } R \ T_2

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

\[ \text{First stage is completed} \]

<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>([(P \Rightarrow P)/T])</td>
</tr>
<tr>
<td>( \text{of } (\text{lam } ([x] \ x)) \ T )</td>
<td>([P/T_1, \ P/T_2])</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. \(u: \text{of} \ x \ T_1 \vdash \text{sub} \ R \ T_2,\)
  \(\vdash \text{sub} \ S \ T\)
- Type dependencies
  e.g. \(u: \text{of} \ x \ T_1 \vdash \text{of} \ x \ (R \ x \ u),\)
  \(u: \text{of} \ x \ T_1 \vdash \text{of} \ x \ R\)
- Indexing for higher-order terms
Introduction

Illustrating example: subtyping

Tabled higher-order logic programming
  – Tabled logic programming interpreter
  – Object- and meta-level theorem prover

Thesis work

Related work

Conclusion
Meta-level reasoning

- Prove theorems about a logical system (type preservation, soundness, correctness ...)
- Proofs by induction and case analysis
- Approaches:
  - λProlog(Felty, Miller), Isabelle(Paulson): based on tactics
  - Twelf(Schürmann, Pfenning): based on logic programming
Meta-level search

- Clauses: program, lemmas, proof assumptions
- Proof obligation (query): derive from clauses
- If we cannot derive the query from the clauses,
  1. Refine proof assumptions: case split (choice!)
  2. Generate induction hypothesis
  3. Try again
Meta-level search

- Clauses: program, lemmas, proof assumptions
- Proof obligation (query): derive from clauses
- If we cannot derive the query from the clauses,
  1. Refine proof assumptions: case split (choice!)
  2. Generate induction hypothesis
  3. Try again
- Without failure of logic programming search, no progress
Meta-level search

- Clauses: program, lemmas, proof assumptions
- Proof obligation (query): derive from clauses
- If we cannot derive the query from the clauses,
  1. Refine proof assumptions: case split (choice!)
  2. Generate induction hypothesis
  3. Try again
- Without failure of logic programming search, no progress fail quick and meaningful!
Redundant computation

Meta-level proof tree

- Object-level search
- Across branches
Redundant computation

Meta-Search

1. iteration

• Object-level search
• Across branches
• Across failed attempts

2. iteration

3. iteration

Object-level search
Across branches
Across failed attempts

Tabled higher-order logic programming – p.28/40
Redundant computation

Meta-Search

1. iteration

2. iteration

3. iteration

- Object-level search
- Across branches
- Across failed attempts
- Across parallel proof attempts
Benefits of tabled meta-level search

- Redundancy elimination during object-level search
- Preservation of partial results across cases and iterations
- Detection of unprovable branches
- Faster failure
- Proving different case split in parallel
- Detection of redundant case splits (e.g. split a and then split b, split b and then split a)
Outline

• Introduction
• Illustrating example: subtyping
• Tabled higher-order logic programming
  – Tabled logic programming interpreter
  – Object- and meta-level theorem prover
• Thesis work
• Conclusion
Tabled higher-order logic programming allows us to

- efficiently execute logical systems

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

- efficiently execute logical systems (interpreter using tabled search)
- automate reasoning with and about them.
Tabled higher-order logic programming allows us to

- efficiently execute logical systems
  (interpreter using tabled search)
- automate reasoning with and about them.
  (theorem prover using tabled search)
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
**Examples: interpreter - 1**

Warning: table everything; no indexing

<table>
<thead>
<tr>
<th></th>
<th>Elf</th>
<th>variant</th>
<th>subsumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>subtyping1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>zsuper</td>
<td>∞</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>casez1</td>
<td>∞</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>disprove</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>zerop</td>
<td>∞</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>casez2</td>
<td>∞</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>subtyping</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tid</td>
<td>∞</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>sarrow</td>
<td>∞</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

Tabled higher-order logic programming – p.33/40
### Examples: interpreter - 2

**Warning:** table everything; no indexing

<table>
<thead>
<tr>
<th>Elf variant subsumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>refinement types:</td>
</tr>
<tr>
<td>shiftl ✓ na −</td>
</tr>
<tr>
<td>inc ✓ na −</td>
</tr>
<tr>
<td>plus ✓ na ≡</td>
</tr>
<tr>
<td>plus’ ✓ na +</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>term rewriting λcalculus:</th>
</tr>
</thead>
<tbody>
<tr>
<td>rsym5 no ✓ na</td>
</tr>
<tr>
<td>comb no ✓ na</td>
</tr>
</tbody>
</table>
### Warning: table everything; no indexing

<table>
<thead>
<tr>
<th>conversions</th>
<th>λcalculus:</th>
<th>Spass</th>
<th>Twelf</th>
<th>variant</th>
<th>subsumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>rsym5</td>
<td>no</td>
<td>no</td>
<td>✓</td>
<td>na</td>
<td></td>
</tr>
<tr>
<td>comb</td>
<td>no</td>
<td>no</td>
<td>✓</td>
<td>na</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cartesian closed categories:</th>
</tr>
</thead>
<tbody>
<tr>
<td>l1</td>
</tr>
<tr>
<td>l2</td>
</tr>
<tr>
<td>l3</td>
</tr>
</tbody>
</table>
Other examples

Logical systems:

- 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 \( \lambda \) calculus
Outline

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

• Extension of tabling to higher-order setting
  1. Terms: dependently typed $\lambda$-calculus
  2. Table: store goals with a context
• Application of tabled search to
  1. higher-order logic programming
  2. object- and meta-level theorem proving
• Proof-theoretical characterization of tabled search
• Implementation of a prototype
Contributions

• Extension of tabling to higher-order setting
  1. Terms: dependently typed \( \lambda \)-calculus
  2. Table: store goals with a context
• Application of tabled search to
  1. higher-order logic programming
  2. object- and meta-level theorem proving
• Proof-theoretical characterization of tabled search
• Implementation of a prototype
Contributions

- Extension of tabling to higher-order setting
  1. Terms: dependently typed $\lambda$-calculus
  2. Table: store goals with a context
- Application of tabled search to
  1. higher-order logic programming
  2. object- and meta-level theorem proving
- Proof-theoretical characterization of tabled search
- Implementation of a prototype
Near Future

- Soundness of the interpreter
- Indexing for higher-order terms
- Redesign of the meta-theorem prover
Related Work

Proof-theoretical characterization

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

Certificates:

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