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 λ-calculus

  Clauses: implication, universal quantification

Introduction

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

- Meta-language for specifying / implementing
  logical systems

  proofs about them

Introduction

- Higher-order logic programming

  Terms: (dependently) typed λ-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 λ-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 λ-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, λProlog, 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

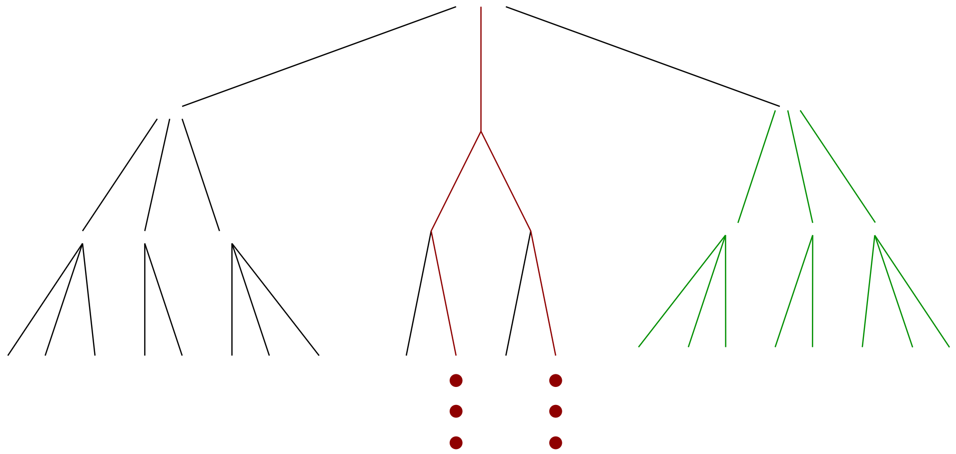
Generic framework for . . .

- 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

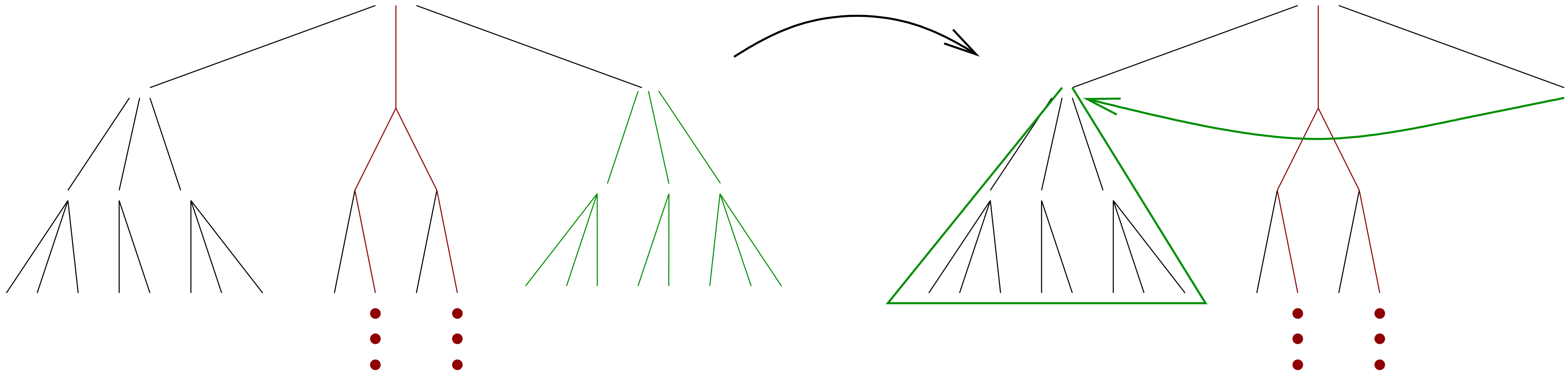
Proof search tree

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



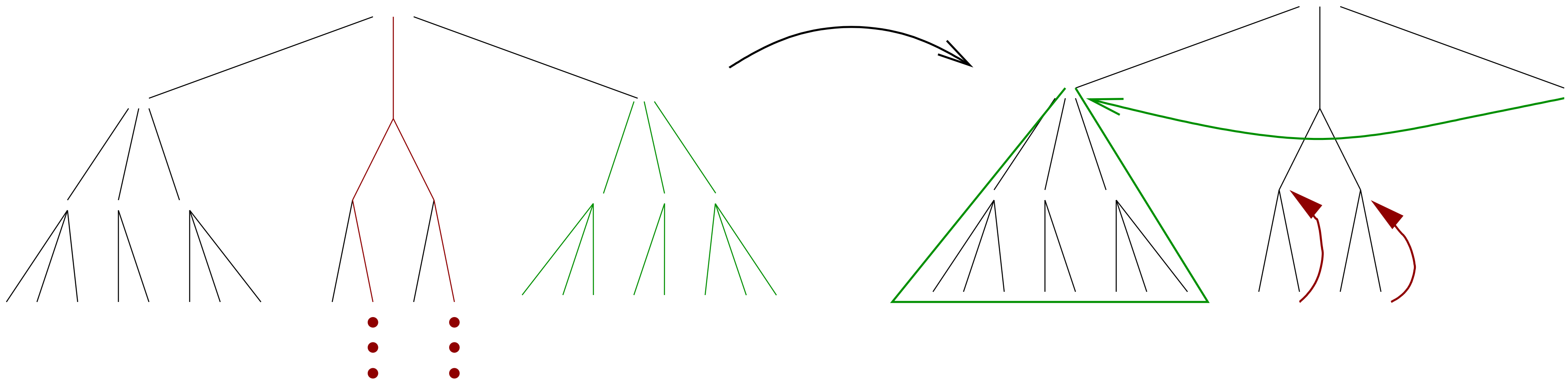
Proof search tree

- 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.

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.
     (theorem prover using tabled search)

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

Illustrating example: subtyping

Types τ ::= neg | zero | pos | nat | int

Illustrating example: subtyping

Types τ ::= neg | zero | pos | nat | int

$$\frac{}{\text{zero} \preceq \text{nat}}\ \text{zn} \qquad \frac{}{\text{pos} \preceq \text{nat}}\ \text{pn}$$

Illustrating example: subtyping

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

$$\frac{}{\text{zero} \preceq \text{nat}}\,\text{zn} \qquad \frac{}{\text{pos} \preceq \text{nat}}\,\text{pn} \qquad \frac{}{\text{nat} \preceq \text{int}}\,\text{nati} \qquad \frac{}{\text{neg} \preceq \text{int}}\,\text{negi}$$

Illustrating example: subtyping

Types τ ::= neg | zero | pos | nat | int

$$\frac{}{\text{zero} \preceq \text{nat}}\ \text{zn} \qquad \frac{}{\text{pos} \preceq \text{nat}}\ \text{pn} \qquad \frac{}{\text{nat} \preceq \text{int}}\ \text{nati} \qquad \frac{}{\text{neg} \preceq \text{int}}\ \text{negi}$$

$$\frac{}{T \preceq T}\ \text{refl} \qquad \frac{T \preceq R \qquad R \preceq S}{T \preceq S}\ \text{tr}$$

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
	$\leftarrow$ 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.

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.

refl: $T =$ 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 ; 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

$\leftarrow$ 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

refl: $T =$ zero

Redundant answer

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
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.

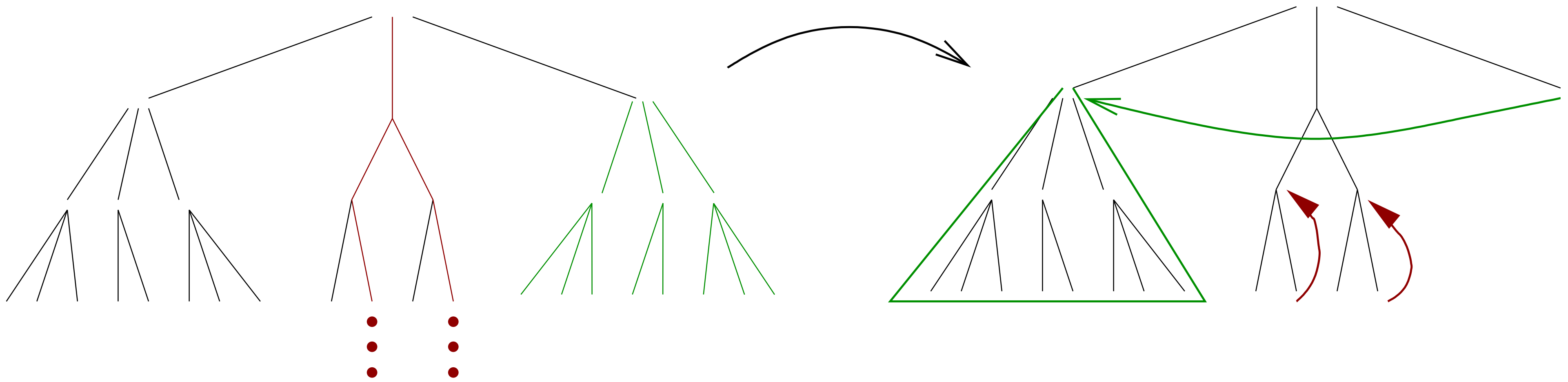
refl: sub zero T

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 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 .

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.

Entry	Answer
sub zero T	

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.

refl: $T =$ zero

Success!

Entry	Answer
sub zero T	

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.

refl: $T =$ zero

Add answer to table

Entry	Answer
sub zero T	[zero $/T$]

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.

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

$\leftarrow$ 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.

Fail and suspend goal

Entry	Answer
sub zero T	[zero $/T$]

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 : $T =$ nat

Success!

Entry	Answer
sub zero T	[zero $/T$]

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 : $T =$ nat

Add answer to table

Entry	Answer
sub zero T	[zero $/T$], [nat $/T$]

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: $T =$ nat

Add answer to table

Entry	Answer
sub zero T	$[$zero $/T]$, $[$nat $/T]$

First Stage completed!

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.

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

Entry	Answer
sub zero T	$[$zero $/T]$, $[$nat $/T]$

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.

resume sub zero R ; sub R T.

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

Entry	Answer
sub zero T	$[$zero $/T]$, $[$nat $/T]$

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.

resume sub zero R ; sub R T.

[nat $/R$] sub nat T.

Add goal to table

Entry	Answer
sub zero T	[zero $/T$], [nat $/T$]
sub nat T	

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.

resume sub zero R ; sub R T.

[nat $/R$] sub nat T

refl $T =$ nat

Success

Entry	Answer
sub zero T	[zero $/T$], [nat $/T$]
sub nat T	

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.

resume sub zero R ; sub R T.

[nat $/R$] sub nat T

refl $T =$ nat

Add answer to table

Entry	Answer
sub zero T	[zero $/T$], [nat $/T$]
sub nat T	[nat $/T$]

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.

Entry	Answer
sub zero T	[zero $/T$], [nat $/T$], [int $/T$]
sub nat T	[nat $/T$], [int $/T$]
sub int T	[int $/T$]

Strategy

- 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) | app e_1 e_2 |
lam $x.e$ | letn $u = e_1$ in e_2

$$\frac{\Gamma \vdash e : \tau' \qquad \tau' \preceq \tau}{\Gamma \vdash e : \tau}\ \text{tp-sub} \qquad \frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \text{lam}\ x.e : \tau_1 \to \tau_2}\ \text{tp-lam}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash [e_1/u]e_2 : \tau}{\Gamma \vdash \text{letn}\ u = e_1\ \text{in}\ e_2 : \tau}\ \text{tp-letn}$$

Type Checker in Elf

tp-sub :of E T
$\leftarrow$ of E T'
$\leftarrow$ sub T' T.

tp-lam :of $($lam $([x]\ E\ x))\ (T_1 \Rightarrow T_2)$
$\leftarrow (\{y\}$of $y\ T_1 \rightarrow$ of $(E\ y)\ T_2)$.

tp-letn :of $($letn $E_1\ ([u]\ E_2\ u))\ T$
$\leftarrow$ of E_1 T_1
$\leftarrow$ of $(E_2\ E_1)\ T$.

Tabled computation (higher-order)

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

Entry	Answer
of (lam $([x]\ x)$) T	

Tabled computation (higher-order)

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

tp-sub: of (lam $([x]\ x)$) R ; sub R T.

Entry	Answer
of (lam $([x]\ x)$) T	

Tabled computation (higher-order)

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

tp-sub: of (lam $([x]\ x))\ R$; sub $R\ T$.

Variant of previous goal

Entry	Answer
of (lam $([x]\ x))\ T$	

Tabled computation (higher-order)

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

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

Fail and suspend

Entry	Answer
of $(\text{lam}\ \ ([x]\ x))\ T$	

Tabled computation (higher-order)

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

tp-lam: u : of $x\ T_1 \vdash$ of $x\ T_2$

Entry	Answer
of (lam $([x]\ x))\ T$	

Tabled computation (higher-order)

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

tp-lam: u : of $x\ T_1 \vdash$ of $x\ T_2$

Add goal to table

Entry	Answer
of (lam $([x]\ x))\ T$	
u : of $x\ T_1 \vdash$ of $x\ T_2$	

Tabled computation (higher-order)

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

tp-lam: u : of $x\ T_1 \vdash$ of $x\ T_2$

u: $T_1 = P,\ T_2 = P,\ T = (P \Rightarrow P)$

Success

Entry	Answer
of (lam $([x]\ x))\ T$	
u : of $x\ T_1 \vdash$ of $x\ T_2$	

Tabled computation (higher-order)

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

tp-lam: u : of $x\ T_1 \vdash$ of $x\ T_2$

u: $\quad T_1 = P,\ T_2 = P,\ T = (P \Rightarrow P)$

Add answers to table

Entry	Answer
of (lam $([x]\ x))\ T$	$[(P \Rightarrow P)/T]$
u : of $x\ T_1 \vdash$ of $x\ T_2$	$[P/T_1,\ P/T_2]$

Tabled computation (higher-order)

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

tp-lam: u : of $x\ T_1 \vdash$ of $x\ T_2$

tp-sub: u : of $x\ T_1 \vdash$ of $x\ R$; sub $R\ T_2$

Entry	Answer
of (lam $([x]\ x))\ T$	$[(P \Rightarrow P)/T]$
u : of $x\ T_1 \vdash$ of $x\ T_2$	$[P/T_1,\ P/T_2]$

Tabled computation (higher-order)

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

tp-lam: u : of $x\ T_1 \vdash$ of $x\ T_2$

tp-sub: u : of $x\ T_1 \vdash$ of $x\ R$; sub $R\ T_2$

Variant of previous goal

Entry	Answer
of (lam $([x]\ x))\ T$	$[(P \Rightarrow P)/T]$
u : of $x\ T_1 \vdash$ of $x\ T_2$	$[P/T_1,\ P/T_2]$

Tabled computation (higher-order)

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

tp-lam: u : of $x\ T_1 \vdash$ of $x\ T_2$

tp-sub: u : of $x\ T_1 \vdash$ of $x\ R$; sub $R\ T_2$

Suspend and fail

Entry	Answer
of (lam $([x]\ x))\ T$	$[(P \Rightarrow P)/T]$
u : of $x\ T_1 \vdash$ of $x\ T_2$	$[P/T_1,\ P/T_2]$

Tabled computation (higher-order)

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

First stage is completed

Entry	Answer
of (lam $([x]\ x))\ T$	$[(P \Rightarrow P)/T]$
u : of $x\ T_1 \vdash$ of $x\ T_2$	$[P/T_1,\ P/T_2]$

Challenges

- Store goals together with context : $\Gamma \vdash a$
- Redesign table operations : goal $(\Gamma \vdash a)\ \in$ Table
- Context dependencies
  e.g. u : of x $T_1 \vdash$ sub R T_2,
  $\vdash$ sub S T
- Type dependencies
  e.g. u : of x $T_1 \vdash$ of x $(R\ x\ u)$,
  u : of x $T_1 \vdash$ of x R
- Indexing for higher-order terms

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

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 tacitics
  - 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

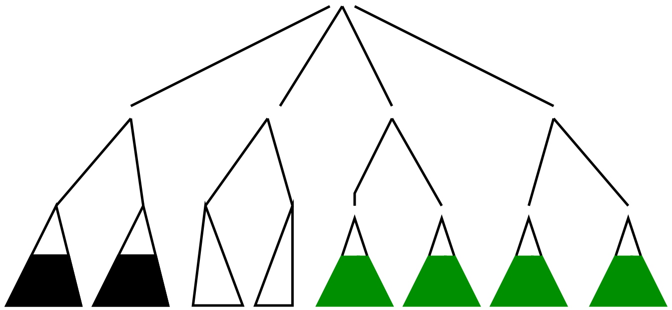
- Clauses: program, lemmas, proof assumptions
- Proof obligation (query): derive from clauses
- If we **cannot** derive the query from the clauses,
  1. Refine proof assumptions: case split (choice!)
  2. Generate induction hypothesis
  3. Try again
- Without failure of logic programming search, no progress

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

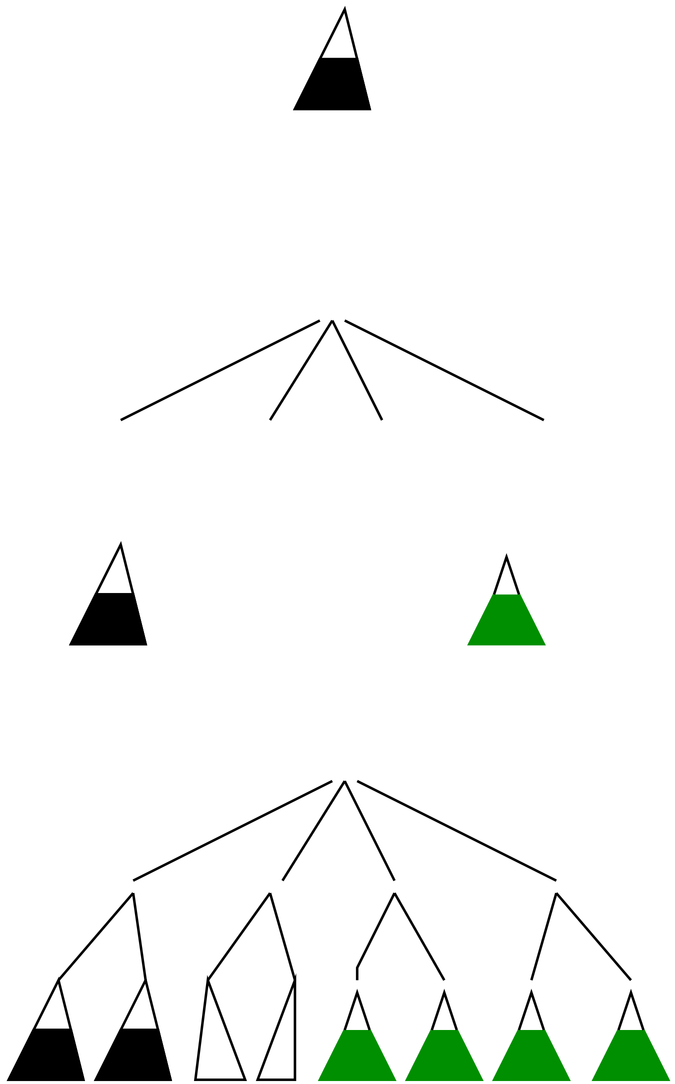
Redundant computation

Meta-Search

1. iteration

2. iteration

3. iteration



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

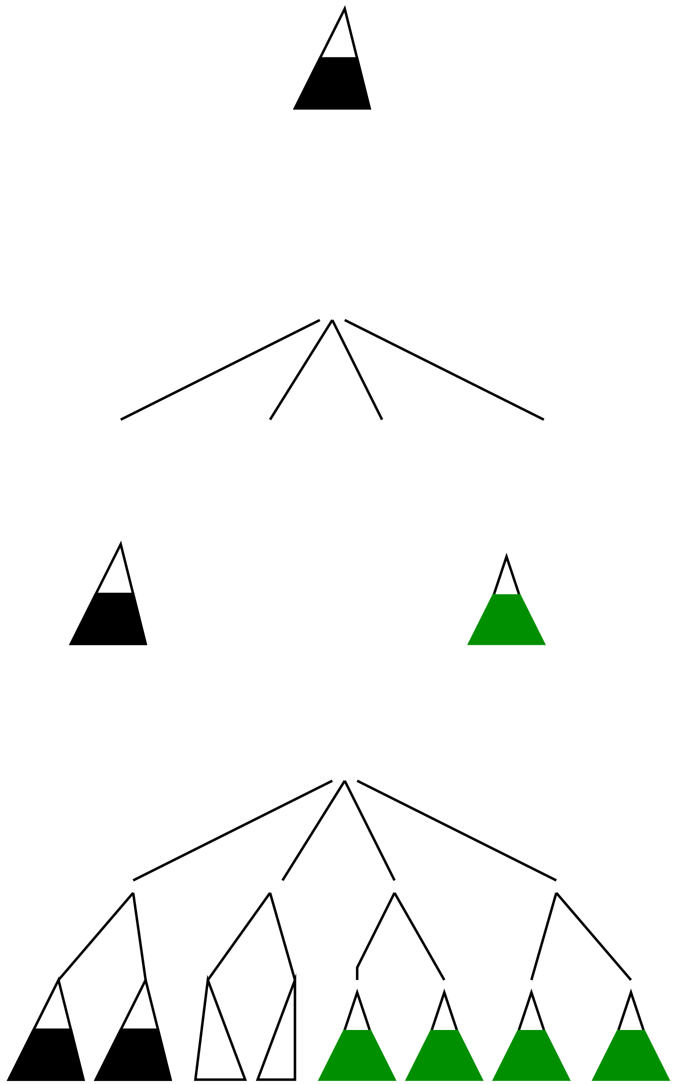
Redundant computation

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

Thesis

Tabled higher-order logic programming allows us to

- efficiently execute logical systems
- automate reasoning with and about them.

Thesis

Tabled higher-order logic programming allows us to

- efficiently execute logical systems
  (interpreter using tabled search)
- automate reasoning with and about them.

Thesis

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

	Elf	variant	subsumption
subtyping1			
zsuper	∞	$\surd$	$\surd$
casez1	∞	$\surd$	$\surd$
disprove			
zerop	∞	$\surd$	$\surd$
casez2	∞	$\surd$	$\surd$
subtyping			
tid	∞	$\surd$	$\surd$
sarrow	∞	$\surd$	$\surd$

Examples: interpreter - 2

Warning: table everything; no indexing

	Elf	variant	subsumption
refinement types:			
shiftl	$\surd$	na	$-$
inc	$\surd$	na	$-$
plus	$\surd$	na	$\equiv$
plus'	$\surd$	na	$+$
term rewriting λcalculus:			
rsym5	no	$\surd$	na
comb	no	$\surd$	na

Object-level reasoning - 3

Warning: table everything; no indexing

	Spass	Twelf	variant	subsumption
conversions λcalculus:				
rsym5		no	$\surd$	na
comb		no	$\surd$	na
Cartesian closed categories:				
l1	no	no	?	?
l2	no	no	?	?
l3	no	no	?	?

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 betwen CCC and λcalculus

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 λ-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 λ-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 λ-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)