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 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

- 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.*)

Thesis

Tabled higher-order logic programming allows us to

- efficiently execute logical systems and

- automate the reasoning with and about them.

Thesis

Tabled higher-order logic programming allows us to

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

Thesis

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 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

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

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

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

refl: sub zero T

tr: sub zero $R \leftarrow$ 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 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 $R \leftarrow$ 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 $R \leftarrow$ 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	$[\text{zero}\ /T]$, $[\text{nat}\ /T]$

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 $R \leftarrow$ 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 $R \leftarrow$ 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 $R \leftarrow$ 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 $R \leftarrow$ 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 $R \leftarrow$ 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 ?

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 $\quad$ e $\quad$ ::= $\quad$ 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 $([x]\ x))\ R \leftarrow$ sub $R\ T$.

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

Tabled computation (higher-order)

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

tp_sub: of (lam $([x]\ x))\ R \leftarrow$ sub $R\ T$.

Variant of previous goal

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

Tabled computation (higher-order)

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

tp˙sub: of (lam $([x]\ x))\ R \leftarrow$ sub $R\ T$.

Fail and suspend

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 $x\ T_2$

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

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 $x\ R \leftarrow$ 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 $x\ R \leftarrow$ 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 $x\ R \leftarrow$ sub $R\ T_2$

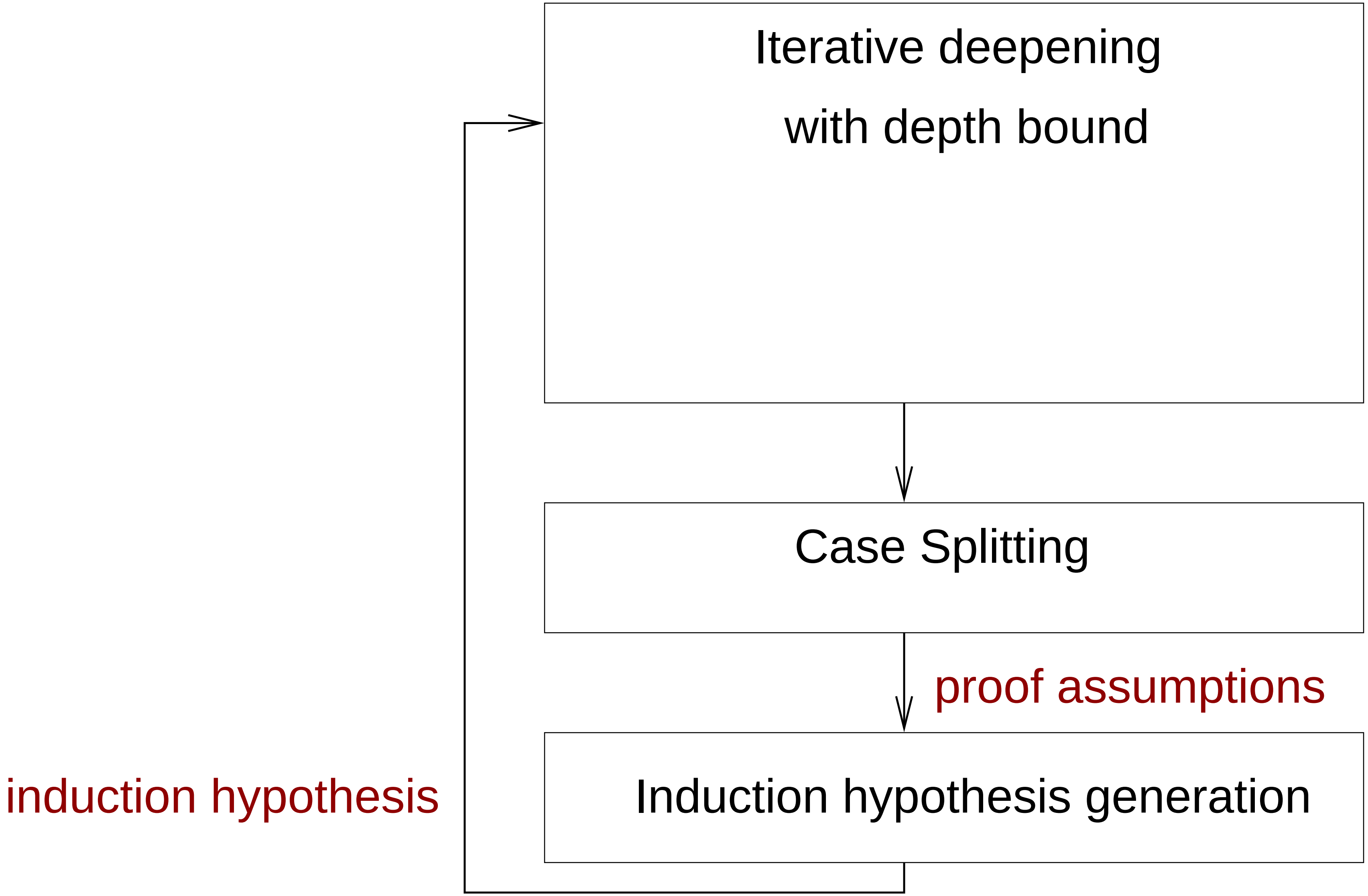
Suspend and fail

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

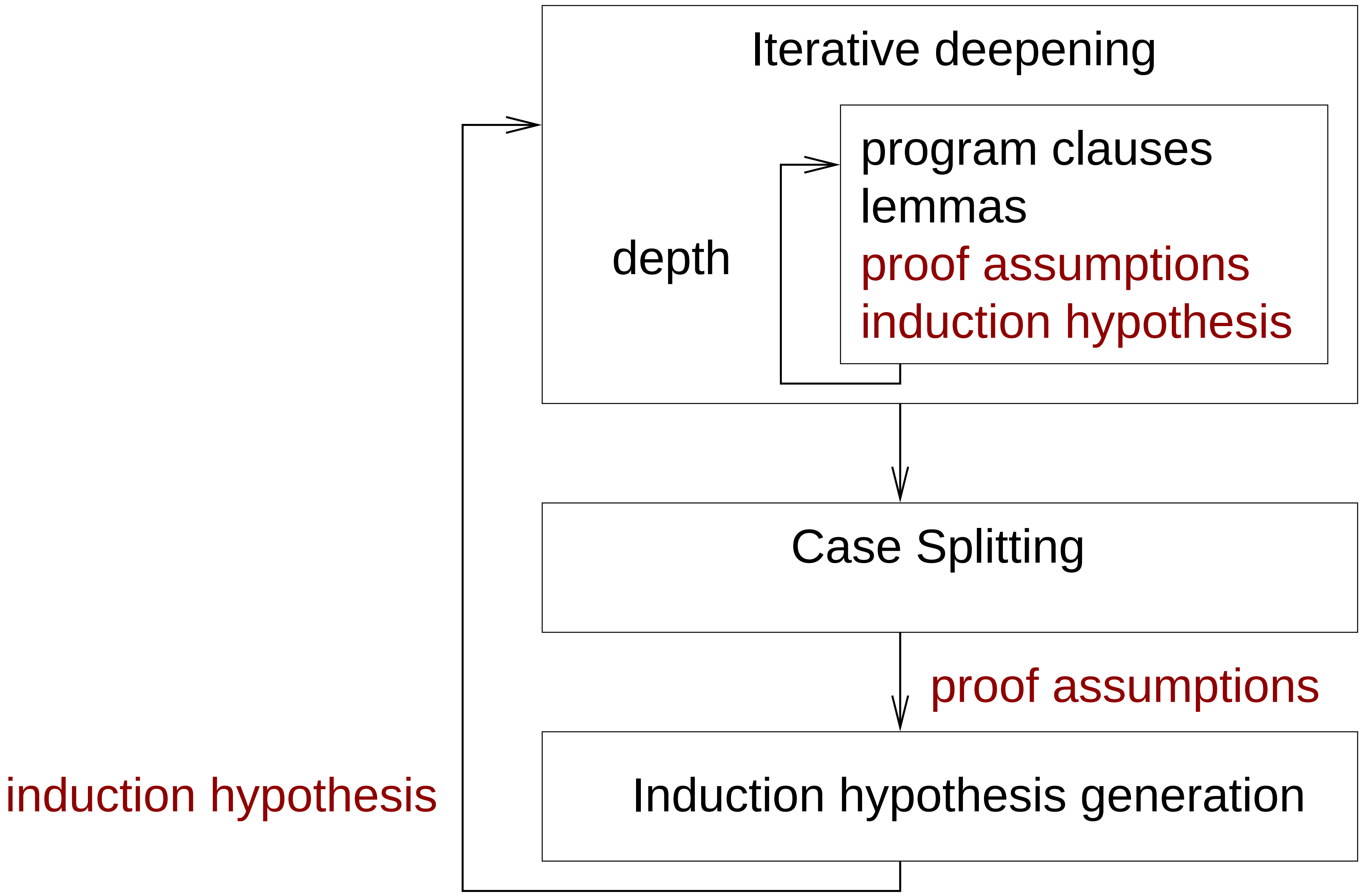
Challenges

- 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)
- Type dependencies
  (e.g. u : of x $T_1 \vdash$ of x $(R\ x\ u)$)
- Indexing for higher-order terms

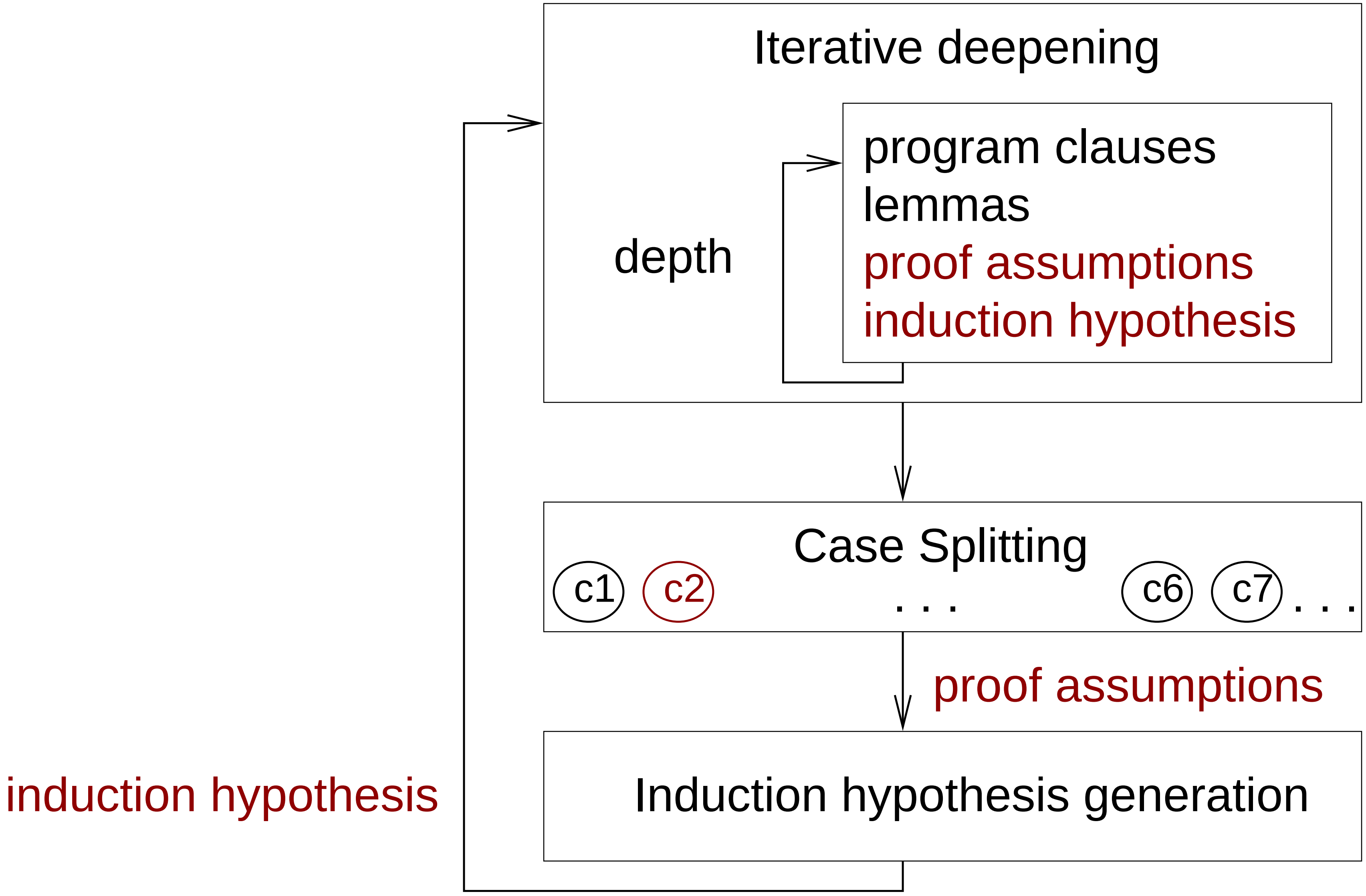
Meta-level reasoning



Meta-level reasoning



Meta-level reasoning

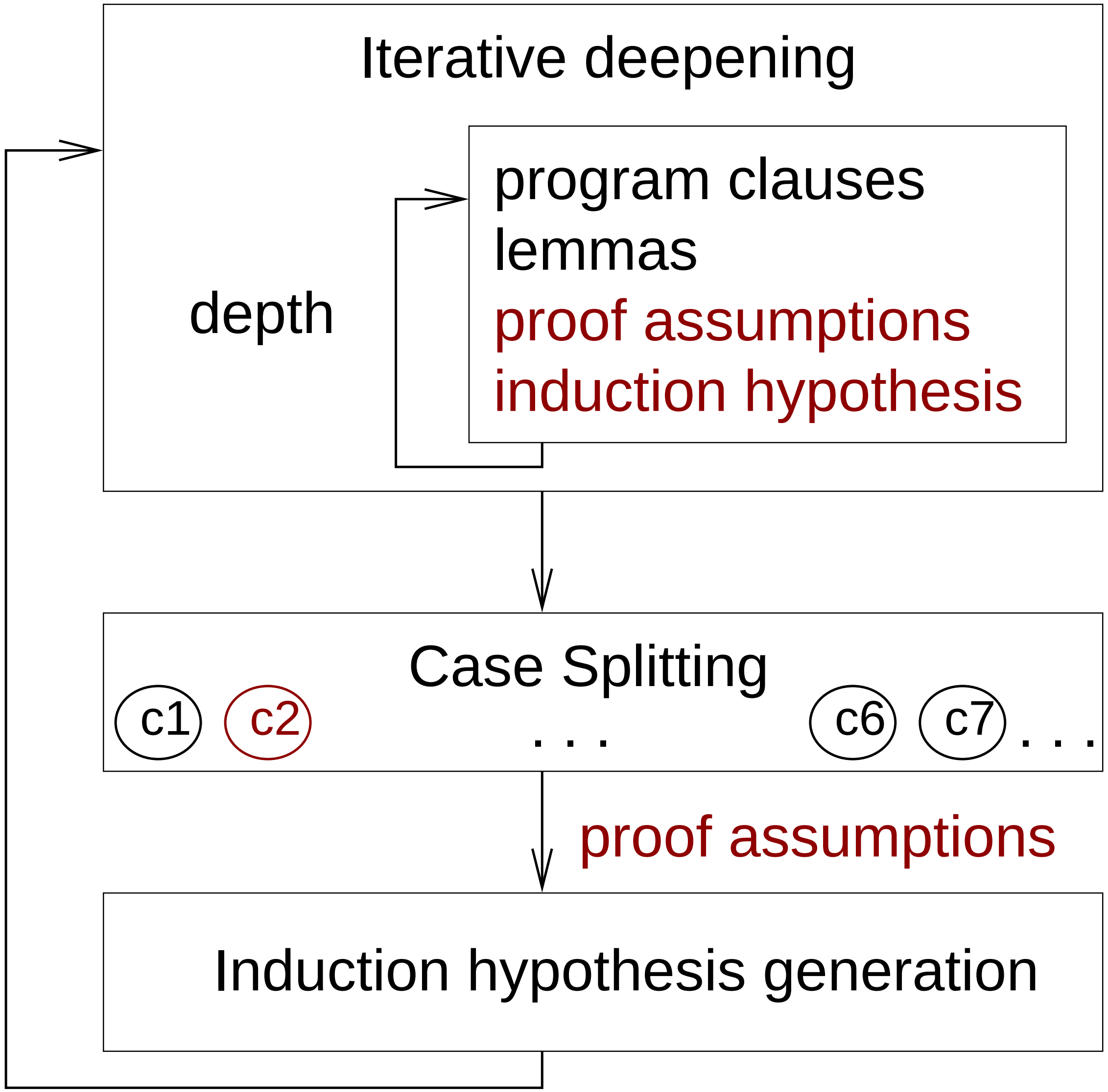


Meta-level reasoning

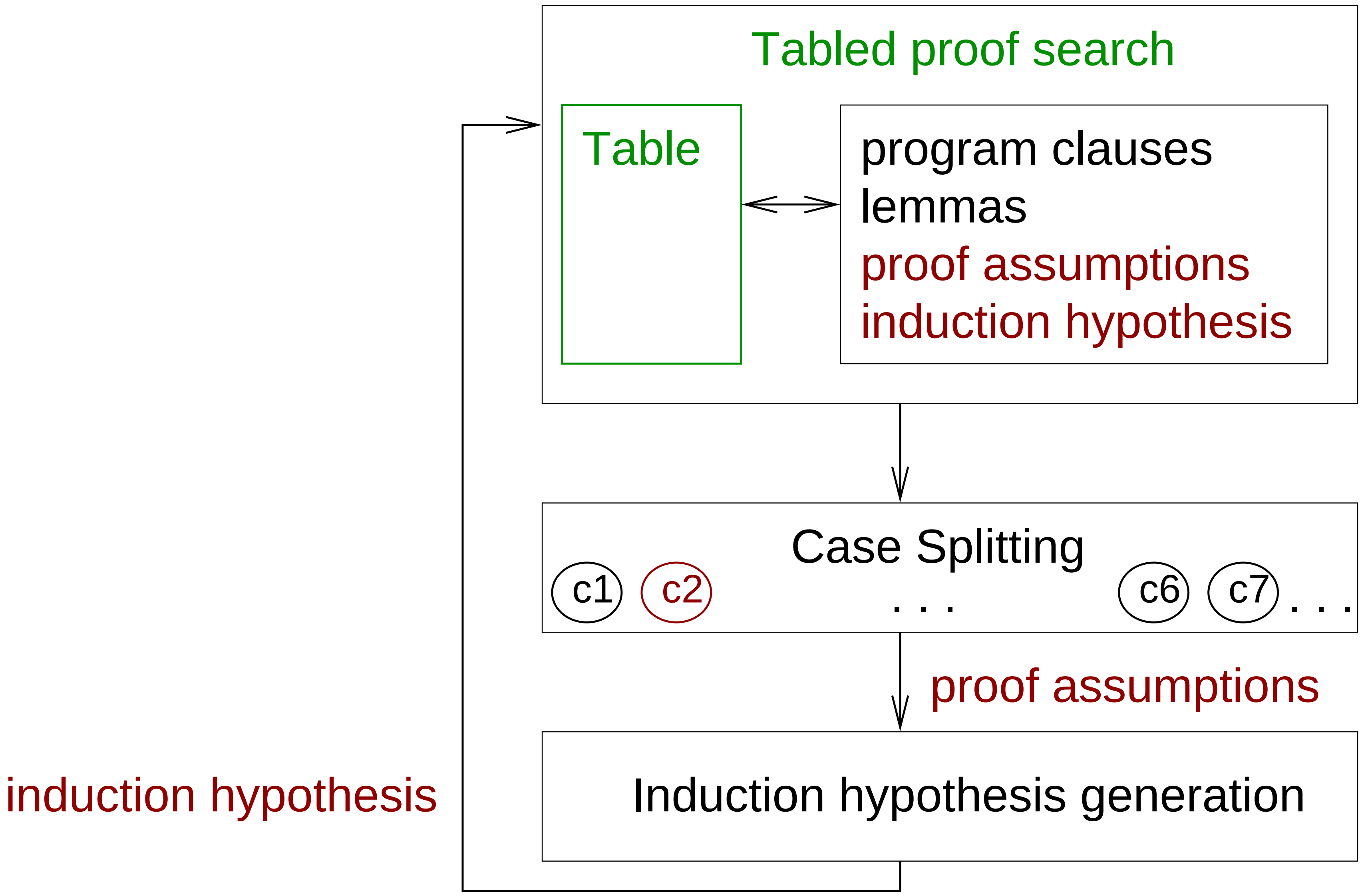
Drawbacks:

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

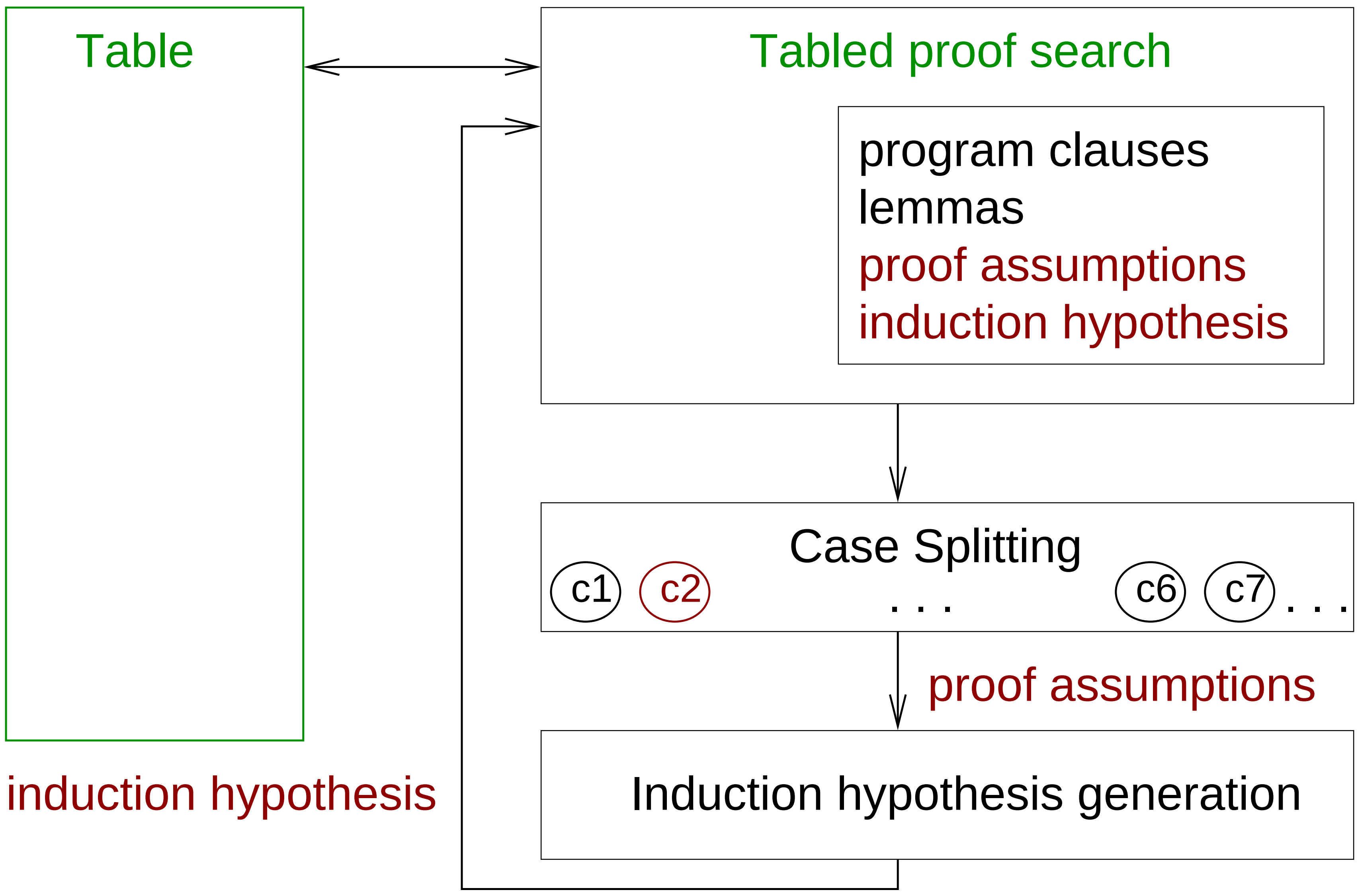
induction hypothesis



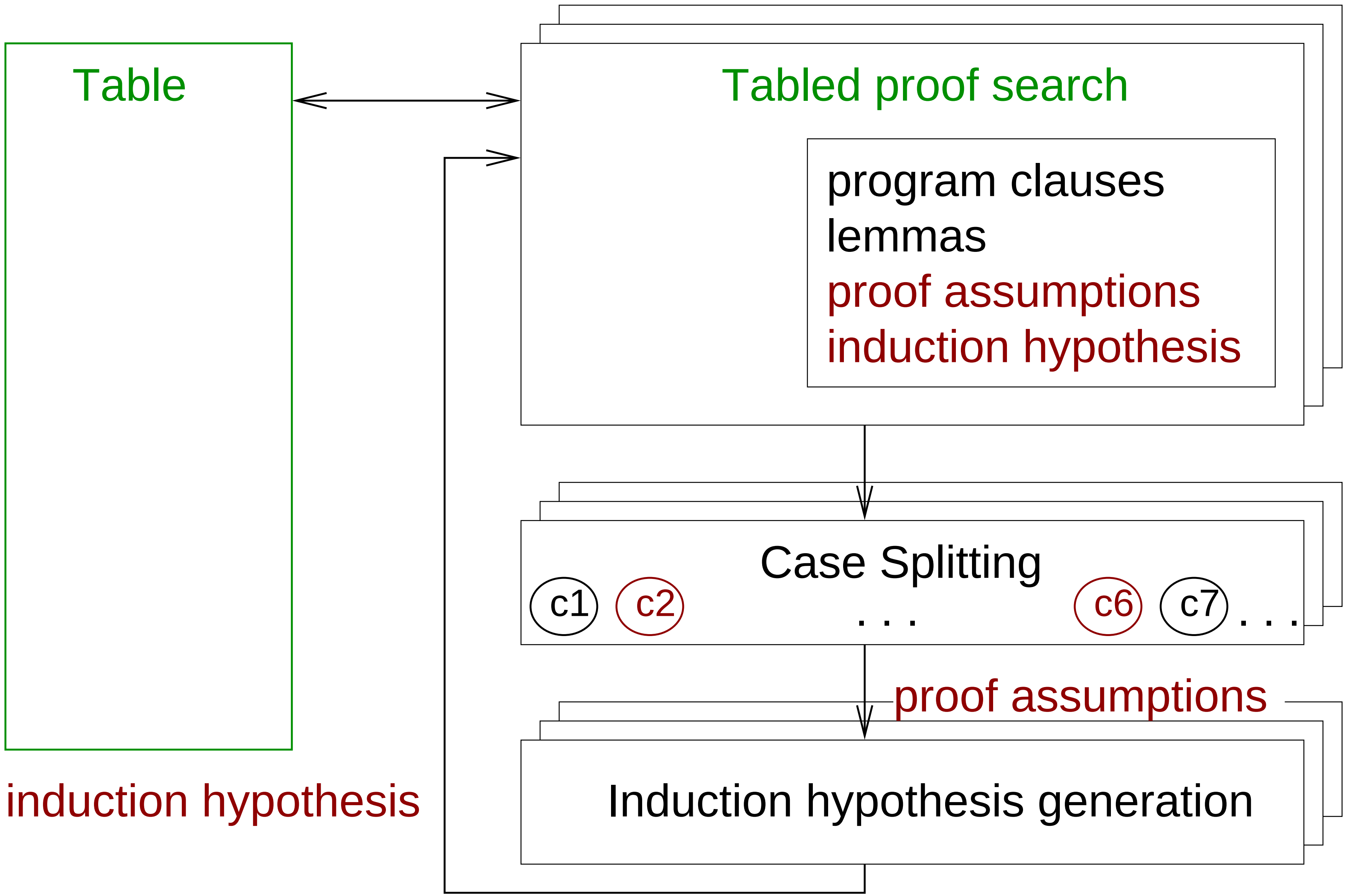
Meta-level reasoning with tabling



Meta-level reasoning with tabling



Meta-level reasoning with tabling



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 λ-calculus
  - Conversions in the λ-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 betwen CCC and λcalculus

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), λProlog(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)

Related Work

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