Memoization-based proof search in LF: an experimental evaluation of a propotype

Brigitte Pientka

Department of Computer Science

Carnegie Mellon University

Pittsburgh, PA, 15217, USA

email:bp@cs.cmu.edu

Outline

- LF as a logic programming framework
- Example: Type-system with subtyping
- Basics of tabled higher-order logic programming
- Experimental evaluation:
  1. Refinement type-checking:
     Depth-first vs tabled search
  2. Parsing into higher-order abstract syntax:
     Iterative deepening vs tabled search
- Related Work
- Conclusion and future work

LF as a logic programming framework

Logical framework LF [Harper,Honsell,Plotkin93]

- dependently typed λ-calculus

LF as a logic programming framework

Logical framework LF [Harper,Honsell,Plotkin93]
dependently typed λ-calculus

Framework for specifying and implementing

- logical systems
- proofs about them

LF as a logic programming framework

Logical framework LF [Harper,Honsell,Plotkin93]
dependently typed λ-calculus

Framework for specifying and implementing

- logical systems (safety logics, type system . . .)
- proofs about them

LF as a logic programming framework

Logical framework LF [Harper,Honsell,Plotkin93]
dependently typed λ-calculus

Framework for specifying and implementing

- logical systems (safety logics, type system . . .)
- proofs about them (correctness, soundness . . .)

LF as a logic programming framework

Logical framework LF [Harper,Honsell,Plotkin93]
dependently typed λ-calculus

Framework for specifying and implementing

- logical systems (safety logics, type system . . .)
- proofs about them (correctness, soundness . . .)

Proof search via higher-order logic programming [Pfenning91]

- Terms: (dependently) typed λ-calculus
- Clauses: implication, universal quantification

Proof search over declarative systems

Proof search problems:

- Infinite computation leads to non-termination.
  $\Rightarrow$ many specifications are not executable
- Redundant computation hampers performance.

"...it is very common for the proofs to have repeated sub-proofs that should be hoisted out and proved only once as lemmas." [Necula,Lee97]

Proof search over declarative systems

Proof search problems:

- Infinite computation leads to non-termination.
  $\Rightarrow$ many specifications are not executable
- Redundant computation hampers performance.

"...it is very common for the proofs to have repeated sub-proofs that should be hoisted out and proved only once as lemmas." [Necula,Lee97]

Solution: Memoization and re-use of sub-proofs

Memoization-based proof search

First-order tabelling [Tamaki,Sato86]

- Memoize atomic subgoals and re-use results
- Finds all possible answers to a query
- Terminates for programs in a finite domain
- Combine tabled and non-tabled execution
- Very successful: XSB system [Warren *et.al.*]

Memoization-based proof search

First-order tabelling [Tamaki,Sato86]

- Memoize atomic subgoals and re-use results
- Finds all possible answers to a query
- Terminates for programs in a finite domain
- Combine tabled and non-tabled execution
- Very successful: XSB system [Warren *et.al.*]

Higher-order tabelling (see also [Pientka, ICLP'02])

- Proof-theoretic characterization
- This talk: Experiments with higher-order tabling

Outline

- LF as a logic programming framework
- Example: Type-system with subtyping
- Basics of tabled higher-order logic programming
- Experimental evaluation:
  1. Refinement type-checking:
     Depth-first vs tabled search
  2. Parsing into higher-order abstract syntax:
     Iterative deepening vs tabled search
- Related Work
- Conclusion and future work

Declarative description of subtyping

types τ ::= zero | pos | nat | bit | $\tau_1 \Rightarrow \tau_2$ | $\ldots$

Example: $6 = 110$ and $110 \in$ nat

Declarative description of subtyping

types τ ::= zero $|$ pos $|$ nat $|$ bit $|$ $\tau_1 \Rightarrow \tau_2$ $|$ $\ldots$

Example: $6 = 110$ and $110 \in$ nat

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

Declarative description of subtyping

$$\text{types} \quad \tau \quad ::= \quad \mathsf{zero} \mid \mathsf{pos} \mid \mathsf{nat} \mid \mathsf{bit} \mid \tau_1 \Rightarrow \tau_2 \mid \ldots$$

Example: $6 = 110$ and $110 \in \mathsf{nat}$

$$\frac{}{\mathsf{zero} \preceq \mathsf{nat}}\,\mathsf{zn} \qquad \frac{}{\mathsf{pos} \preceq \mathsf{nat}}\,\mathsf{pn} \qquad \frac{}{\mathsf{nat} \preceq \mathsf{bit}}\,\mathsf{nb}$$

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

Typing rules for Mini-ML

expressions $e \quad ::= \quad \epsilon \mid e\ \mathsf{0} \mid e\ \mathsf{1} \mid \mathsf{lam}\ x.e \mid \mathsf{app}\ e_1\ e_2$

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

Implementation of subtyping

```
zn:  sub zero nat.
pn:  sub pos nat.
nb:  sub nat bit.
refl:  sub T T.
tr:  sub T S
     <- sub T R
     <- sub R S.
```

Implementation of subtyping

```
zn:  sub zero nat.
pn:  sub pos nat.
nb:  sub nat bit.
refl:  sub T T.
tr:  sub T S
     <- sub T R
     <- sub R S.
```

Not executable!

Implementation of typing rules

```
tp_sub:  of E T
      <- of E T'
      <- sub T' T.

tp_lam:  of (lam λx.E x) (T1 => T2)
      <-(Πx:exp.of x T1 -> of (E x) T2).
```

“forall `x:exp`, assume `of x T1`
and show `of (E x) T2`”

Implementation of typing rules

```
tp_sub:  of E T
     <- of E T'
     <- sub T' T.

tp_lam:  of (lam λx.E x) (T1 => T2)
     <-(Πx:exp.of x T1 -> of (E x) T2).
```

“forall `x:exp`, assume `of x T1`
and show `of (E x) T2`”

Redundancy: `tp_sub` is always applicable!

Outline

- LF as a logic programming framework
- Example: Type-system with subtyping
- Basics of tabled higher-order logic programming
- Experimental evaluation:
  1. Refinement type-checking:
     Depth-first vs tabled search
  2. Parsing into higher-order abstract syntax:
     Iterative deepening vs tabled search
- Related Work
- Conclusion and future work

Tabled higher-order logic programming

- Eliminate redundant and infinite paths from proof search using a memo-table

- Table entry: ($\Gamma \rightarrow a$, $\mathcal{A}$)
  - Γ : context of assumptions (i.e.x:exp, u:of x T1)
  - a : atomic goal (i.e. of (lam λx. x) T)
  - $\mathcal{A}$: list of answer substitutions for all free variables in Γ and a

- Depth-first multi-stage strategy adopted from [Tamaki,Sato89]

How higher-order tabling works...

Stage 1

$\cdot \longrightarrow$ of (lam λx.x) T

Entry	Answers

How higher-order tabling works...

Stage 1

$\cdot \longrightarrow$ of (lam λx.x) T

Entry	Answers
$\cdot \longrightarrow$ of (lam λx.x) T	

How higher-order tabling works...

Stage 1

$\cdot \longrightarrow$ of (lam λx.x) T

$\xrightarrow{\text{tp\_sub}}$ $\cdot \longrightarrow$ of (lam λx.x) R, sub R T

Entry	Answers
$\cdot \longrightarrow$ of (lam λx.x) T	

How higher-order tabling works...

Stage 1

$\cdot \longrightarrow$ of (lam λx.x) T

tp_sub $\rightarrow$ $\boxed{\cdot \longrightarrow \text{of (lam } \lambda\text{x.x) R, sub R T}}$ Suspend

Entry	Answers
$\cdot \longrightarrow$ of (lam λx.x) T	

How higher-order tabling works...

Stage 1

$\cdot \longrightarrow$ of (lam λx.x) T

tp_sub $\rightarrow$ $\boxed{\cdot \longrightarrow \text{of (lam } \lambda\text{x.x) R, sub R T}}$ Suspend

tp_lam $\rightarrow$ x:exp, u:of x T1 $\longrightarrow$ of x T2

Entry	Answers
$\cdot \longrightarrow$ of (lam λx.x) T	

How higher-order tabling works...

Stage 1

$\cdot \longrightarrow$ of (lam λx.x) T

tp_sub $\rightarrow$ $\boxed{\cdot \longrightarrow \text{of (lam } \lambda\text{x.x) R, sub R T}}$ Suspend

tp_lam $\rightarrow$ x:exp, u:of x T1 $\longrightarrow$ of x T2

Entry	Answers
$\cdot \longrightarrow$ of (lam λx.x) T	
x:exp, u:of x T1 $\longrightarrow$ of x T2	

How higher-order tabling works...

Stage 1

$\cdot \longrightarrow$ of (lam λx.x) T

tp_sub $\longrightarrow$ $\boxed{\cdot \longrightarrow \text{of (lam } \lambda\text{x.x) R, sub R T}}$ Suspend

tp_lam $\longrightarrow$ x:exp, u:of x T1 $\longrightarrow$ of x T2 $\xrightarrow{u}$ T1 = S, T2 = S, T = (S $\Rightarrow$ S)

Entry	Answers
$\cdot \longrightarrow$ of (lam λx.x) T	
x:exp, u:of x T1 $\longrightarrow$ of x T2	

How higher-order tabling works...

Stage 1

$\cdot \longrightarrow$ of (lam λx.x) T

$\xrightarrow{\text{tp\_sub}}$ $\boxed{\cdot \longrightarrow \text{of (lam } \lambda\text{x.x) R, sub R T}}$ Suspend

$\xrightarrow{\text{tp\_lam}}$ x:exp, u:of x T1 $\longrightarrow$ of x T2 $\xrightarrow{\text{u}}$ T1 = S, T2 = S, T = (S $\Rightarrow$ S)

Entry	Answers
$\cdot \longrightarrow$ of (lam λx.x) T	T = (S $\Rightarrow$ S)
x:exp, u:of x T1 $\longrightarrow$ of x T2	T1 = S, T2 = S

How higher-order tabling works...

Stage 1

$\cdot \longrightarrow$ of (lam λx.x) T

$\xrightarrow{\text{tp\_sub}}$ $\boxed{\cdot \longrightarrow \text{of (lam } \lambda\text{x.x) R, sub R T}}$ Suspend

$\xrightarrow{\text{tp\_lam}}$ x:exp, u:of x T1 $\longrightarrow$ of x T2

$\xrightarrow{\text{u}}$ T1 = S, T2 = S, T = (S $\Rightarrow$ S)

$\xrightarrow{\text{tp\_sub}}$ x:exp, u:of x T1 $\longrightarrow$ of x R,
sub R T2

Entry	Answers
$\cdot \longrightarrow$ of (lam λx.x) T	T = (S $\Rightarrow$ S)
x:exp, u:of x T1 $\longrightarrow$ of x T2	T1 = S, T2 = S

How higher-order tabling works...

Stage 1

$\cdot \longrightarrow$ of (lam λx.x) T

$\xrightarrow{\text{tp\_sub}}$ $\boxed{\cdot \longrightarrow \text{of (lam } \lambda\text{x.x) R, sub R T}}$ Suspend

$\xrightarrow{\text{tp\_lam}}$ x:exp, u:of x T1 $\longrightarrow$ of x T2

$\xrightarrow{\text{u}}$ T1 = S, T2 = S, T = (S $\Rightarrow$ S)

$\xrightarrow{\text{tp\_sub}}$ $\boxed{\text{x:exp, u:of x T1} \longrightarrow \text{of x R, sub R T2}}$

Suspend

Entry	Answers
$\cdot \longrightarrow$ of (lam λx.x) T	T = (S $\Rightarrow$ S)
x:exp, u:of x T1 $\longrightarrow$ of x T2	T1 = S, T2 = S

Stage 1 finished

How higher-order tabling works...

Stage 1

$\cdot \longrightarrow$ of (lam λx.x) T

$\xrightarrow{\text{tp\_sub}}$ $\boxed{\cdot \longrightarrow \text{of (lam } \lambda\text{x.x) R, sub R T}}$ Suspend - - - - - > Resume

$\xrightarrow{\text{tp\_lam}}$ x:exp, u:of x T1 $\longrightarrow$ of x T2

$\xrightarrow{\text{u}}$ T1 = S, T2 = S, T = (S $\Rightarrow$ S)

$\xrightarrow{\text{tp\_sub}}$ $\boxed{\text{x:exp, u:of x T1} \longrightarrow \text{of x R, sub R T2}}$

Suspend

Entry	Answers
$\cdot \longrightarrow$ of (lam λx.x) T	T = (S $\Rightarrow$ S)
x:exp, u:of x T1 $\longrightarrow$ of x T2	T1 = S, T2 = S

Stage 1 finished

How higher-order tabling works...

Stage 1

$\cdot \longrightarrow$ of (lam λx.x) T

$\xrightarrow{\text{tp\_sub}}$ $\boxed{\cdot \longrightarrow \text{of (lam } \lambda\text{x.x) R, sub R T}}$ Suspend $\dashrightarrow$ Resume

$\xrightarrow{\text{tp\_lam}}$ x:exp, u:of x T1 $\longrightarrow$ of x T2

$\xrightarrow{\text{u}}$ T1 = S, T2 = S, T = (S $\Rightarrow$ S)

$\xrightarrow{\text{tp\_sub}}$ $\boxed{\text{x:exp, u:of x T1} \longrightarrow \text{of x R, sub R T2}}$ $\dashrightarrow$ Resume

Suspend

Entry	Answers
$\cdot \longrightarrow$ of (lam λx.x) T	T = (S $\Rightarrow$ S)
x:exp, u:of x T1 $\longrightarrow$ of x T2	T1 = S, T2 = S

Stage 1 finished

Higher-order issues

- Dependencies among propositions

$$\text{x:exp, u:of x P} \longrightarrow \text{sub P R}$$

Higher-order issues

- Dependencies among propositions

$$\text{x:exp, u:of x P} \longrightarrow \text{sub P R,}$$

strengthen: $\longrightarrow$ sub P R

Higher-order issues

- Dependencies among propositions

  $$\text{x:exp, u:of x P} \longrightarrow \text{sub P R},$$

  strengthen: $$\longrightarrow \text{sub P R}$$

- Dependencies among terms

  $$\text{x:exp, u:of x T1} \longrightarrow \text{of x (R x u)},$$

Higher-order issues

- Dependencies among propositions

  x:exp, u:of x P $\longrightarrow$ sub P R,

  strengthen: $\longrightarrow$ sub P R

- Dependencies among terms

  x:exp, u:of x T1 $\longrightarrow$ of x (R x u),

  strengthen x:exp, u:of x T1 $\longrightarrow$ of x R

Higher-order issues

- Dependencies among propositions

$$\frac{\text{x:exp, u:of x P} \longrightarrow \text{sub P R,}}{\text{strengthen:} \qquad \longrightarrow \text{sub P R}}$$

- Dependencies among terms

$$\frac{\text{x:exp, u:of x T1} \longrightarrow \text{of x (R x u),}}{\text{strengthen} \quad \text{x:exp, u:of x T1} \longrightarrow \text{of x R}}$$

- Subordination analysis [Virga99]

Outline

- LF as a logic programming framework
- Example: Type-system with subtyping
- Basics of tabled higher-order logic programming
- Experimental evaluation:
  1. Refinement type-checking:
     Depth-first vs tabled search
  2. Parsing into higher-order abstract syntax:
     Iterative deepening vs tabled search
- Related Work
- Conclusion and future work

Outline

- LF as a logic programming framework
- Example: Type-system with subtyping
- Basics of tabled higher-order logic programming
- Experimental evaluation:
  1. Refinement type-checking:
     Depth-first vs tabled search
  2. Parsing into higher-order abstract syntax:
     Iterative deepening vs tabled search
- Related Work
- Conclusion and future work

Refinement type checking

- Type-inference with subtyping and intersections
- Bi-directional type-checking algorithm [Davies, Pfenning00]
- Distinguish between expressions for which
  1. a type can be synthesized
  2. can be checked against a given type

Depth-first vs Memoization(all solutions)

Program	Depth-First	Memoization
plus’4	483.070 sec	2.330 sec
plus4	696.730 sec	3.150 sec
plus4(np)	22.770 sec	1.95 sec
sub’1a	0.070 sec	0.240 sec
sub1b	3.88 sec	7.560 sec
sub3b	10.440 sec	11.200 sec
mult1(np)	1133.490 sec	4.690 sec
mult1a	807.730 sec	4.730 sec
mult4	∞	17.900 sec
mult4(np)	∞	13.140 sec

Depth-first vs Memoization(first solution)

Program	Depth-First	Memoization
plus'4	0.08 sec	0.180 sec
plus4	0.1 sec	0.430 sec
plus4(np)	22.770 sec	1.95 sec
sub'1a	0.050 sec	0.240 sec
sub1b	0.250 sec	5.020 sec
sub3b	0.350 sec	8.160 sec
mult1(np)	1133.490 sec	4.690 sec
mult1a	0.160 sec	2.900 sec
mult4	0.250 sec	7.150 sec
mult4(np)	∞	13.020 sec

Evaluation

- Simple memoization improves performance
- #Entries in table < 300
- #SuspendedGoals < 200
- Quick failure is important for program development
- Overhead of memoization may hurt performance
- Multi-stage strategy delays the reuse of answers SCC(strongly connected components)

Type-checker with explicit memoization?

- Investigate special memoization techniques [Davies,Pfenning00]
- Implementation is non-trivial.
- Proofs are larger.
- Sending and checking proofs takes longer.
- Harder to reason about this implementation

Outline

- LF as a logic programming framework
- Example: Type-system with subtyping
- Basics of tabled higher-order logic programming
- Experimental evaluation:
  1. Refinement type-checking:
     Depth-first vs tabled search
  2. Parsing into higher-order abstract syntax:
     Iterative deepening vs tabled search
- Related Work
- Conclusion and future work

Outline

- LF as a logic programming framework
- Example: Type-system with subtyping
- Basics of tabled higher-order logic programming
- Experimental evaluation:
  1. Refinement type-checking:
     Depth-first vs tabled search
  2. Parsing into higher-order abstract syntax:
     Iterative deepening vs tabled search
- Related Work
- Conclusion and future work

Parsing into higher-order abstract syntax

Tokens T :

'forall' | 'exist' | 'and' | 'or' | 'imp' | 'not' | '(' | ')' | 'true' | 'false'

Propositions A :

atom P | $\neg A$ | A & A | A v A | $A \Rightarrow A$ | true | false |

forall $x.A$ | exists $x.A$ | (A)

Precedence $\neg >$ & $> \vee > \Rightarrow$

Associativity

& , $\vee$: left associative

$\Rightarrow$: right associative

Implementation (idea by D.S.Warren)

```
% implication -- right associative
fimp: fi Ctx S S' (P1 => P2)
      <- fo Ctx S ('imp' ; S1) P1
      <- fi Ctx S1 S' P2.
ci:   fi Ctx S S' P
      <- fo Ctx S S' P.
% disjunction -- left associative
for: fo Ctx S S' (P1 v P2)
     <- fo Ctx S ('or' ; S1) P1
     <- fa Ctx S1 S' P2.
```

Iterative Deepening vs Memoization

Length of input	Iter. deepening	Memoization
5	0.020 sec	0.010 sec
20	1.610 sec	0.260 sec
32	208.010 sec	2.020 sec
56	∞	7.980 sec
107	∞	86.320 sec

Evaluation

- Memoization outperforms iter. deepening
- Iterative deepening requires depth-bound
  - Failure meaningless
  - No decision procedure
- #Entries in table < 1000
- #SuspendedGoals < 1100
- Remarks
  1. Unambiguous parser
  2. Representing tokens as facts

Outline

- LF as a logic programming framework
- Example: Type-system with subtyping
- Basics of tabled higher-order logic programming
- Experimental evaluation:
  1. Refinement type-checking:
     Depth-first vs tabled search
  2. Parsing into higher-order abstract syntax:
     Iterative deepening vs tabled search
- Related work
- Conclusion and future work

Higher-order theorem proving

- Tactics and tacticals
  - Isabelle [Paulson86], λProlog[Miller91,Felty93]
  - Need to be rewritten for each specification
  - Requires understanding of prover
  - Proving correctness of tactics often hard
- Memoization-based search
  - User concentrates on specification
  - Generic proof search mechanism
  - Table may contain useful failure information

Deterministic search: an alternative?

- Safe cut: finds exactly one solution
- In general: incomplete
- If there are only ground goals, then deterministic search is complete.
- Less general than memoization-based search
- No overhead

Conclusion

Memoization-based search allows

- generic efficient theorem proving
- execution of more declarative specification
- more efficient execution of implementations
- more flexibility
- small proofs

Memoization has some overhead

- Mixing tabled and non-tabled computation
- Table access
- Table size

Future work

- Higher-order indexing
- Different table strategies
- Incorporate into meta-theorem prover *Twelf* [Schürmann,Pfenning99]
- Applying tabelling to linear logic programming

Finally ...

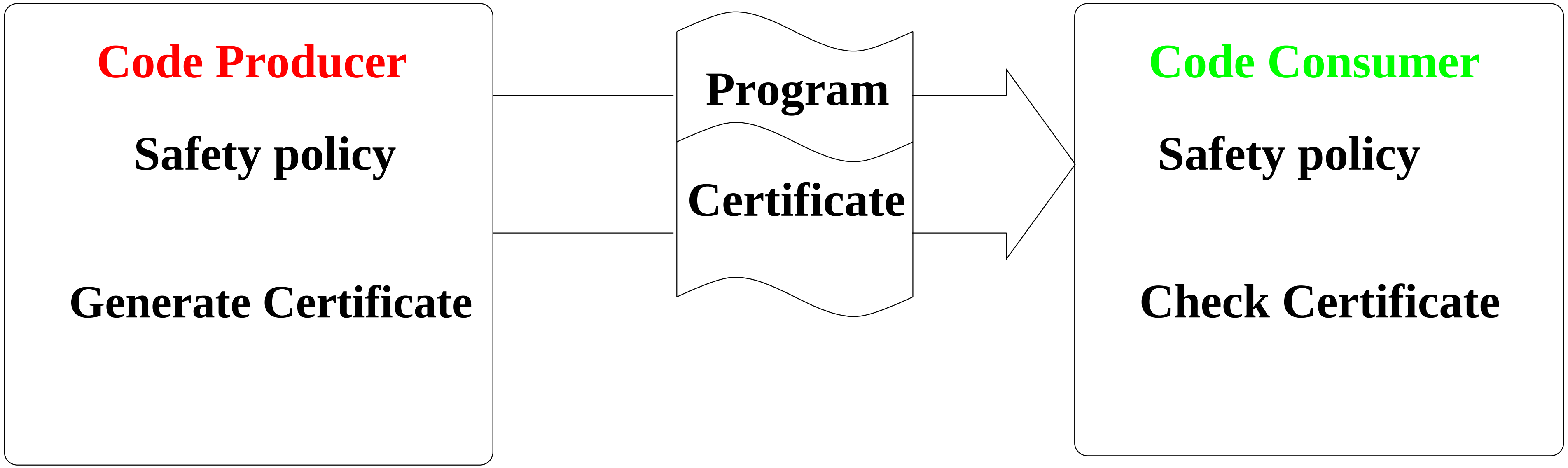
Acknowledgements: Frank Pfenning

if you want to find out more:

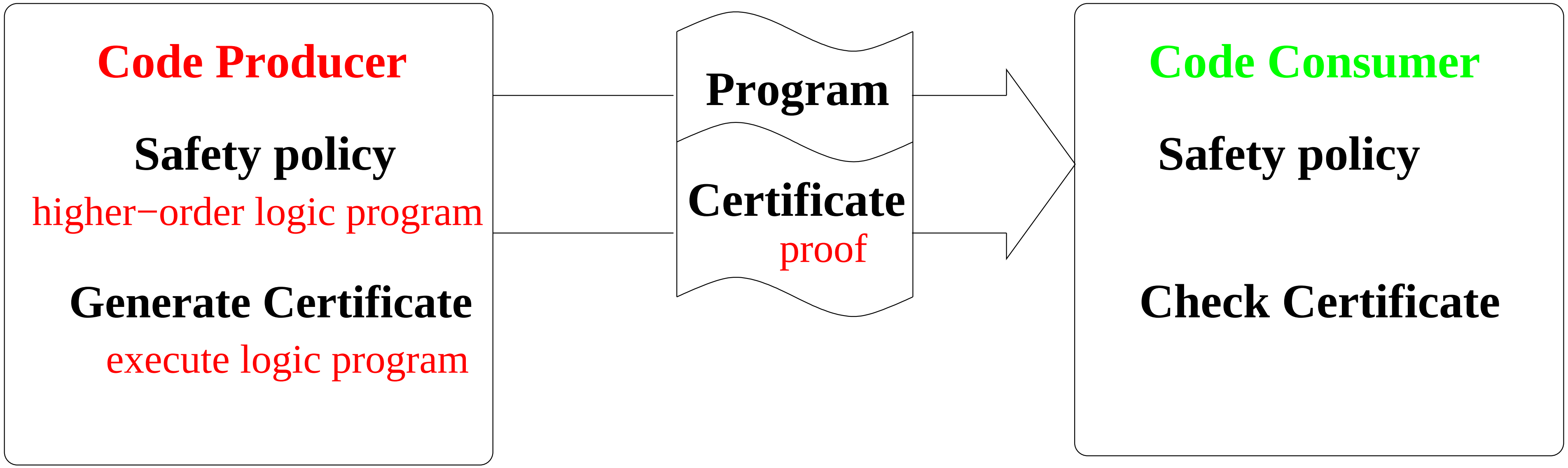
Demo after workshop

http://www.cs.cmu.edu/˜bp
email: bp@cs.cmu.edu

Application:Certified code

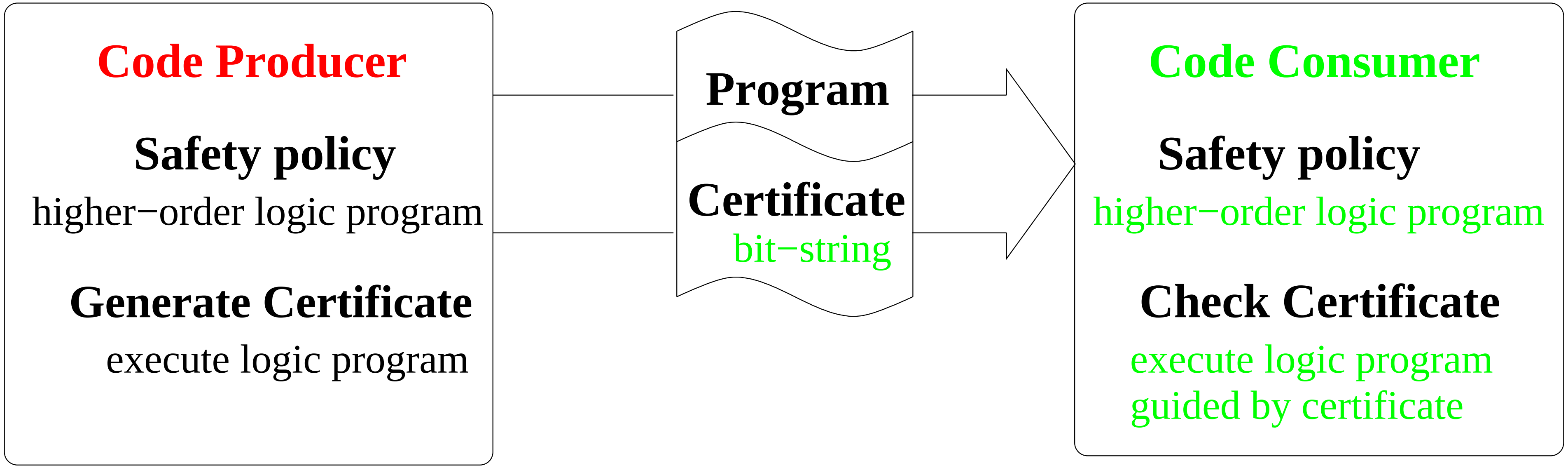


Application:Certified code



- Foundational proof-carrying code : [Appel, Felty 00]
- Proof-carrying authentication: [Felten, Appel 99]

Application:Certified code



- Foundational proof-carrying code : [Appel, Felty 00]
- Proof-carrying authentication: [Felten, Appel 99]
- Proof-checking via bit-strings: [Necula, Rahul 01]

Runtime, #Entries, #SuspGoals

Program	Run-Time	#Entries	#SuspGoals
plus'4	2.330 sec	151	48
plus4	3.150 sec	171	74
plus4(np)	1.95 sec	143	56
sub'1a	0.240 sec	58	11
sub1b	7.560 sec	252	138
sub3b	11.200 sec	278	170
mult1(np)	4.690 sec	217	83
mult1a	4.730 sec	211	78
mult4	17.900 sec	298	270
mult4(np)	13.140 sec	275	194

Time, #Entries, #SuspGoals

Length of input	Memoization	#Entries	#SuspGoals
5	0.010 sec	15	11
20	0.260 sec	60	54
32	2.020 sec	176	197
56	7.980 sec	371	439
107	86.320 sec	929	1185