From Program Verification to Certified Binaries

Angelos Manousaridis Michalis A. Papakyriakou
Nikolaos S. Papaspyrou

National Technical University of Athens
School of Electrical and Computer Engineering
Software Engineering Laboratory

{amanous, mpapakyr, nickie}@softlab.ntua.gr

Logic and Theory of Algorithms
4th Conference on Computability in Europe
Athens, Greece, June 18, 2008

What is this about?

What is this about?

OVERALL RATING: -4 (unacceptable for presentation)
REVIEWER'S CONFIDENCE: 3 (high)

`----------------------- REVIEW -----------------------`

This short paper replays the decade old vision of proof-carrying code, but aiming to increase the level of ambition from simple memory and control-flow properties to arbitrary program properties.

(snip)

I was unable to spot any research contributions or novelty in the paper.

(snip)

In summary, this work is much too preliminary and is in the current state unacceptable for presentation.

So, what is this all about?

- A position paper, not much of a research paper
- Goal? the construction of certified software
  i.e. that provably satisfies its specifications
- Why? the Holy Grail of software engineering!
- How? by combining formal verification
  and proof-carrying code

Outline

Introduction
    Program verification
    Proof-carrying code

Motivation

A Hybrid System

A Motivating Example

Proof-preserving Compilation

Conclusion

Introduction (i)

- Program verification
  - aims at formally proving program correctness
  - given a formal specification or property

  - long tradition (4 decades)
  - several formal logics (e.g. Hoare Logic)

  - at the source code level

Introduction *(ii)*

- Proof-carrying code (PCC)
  - certified binary: a value together with a proof that the value satisfies a given specification
  - relatively recent approach (∼10 years)
  - essential in modern distributed computer systems
  - executable code is transferred among devices that do not necessarily trust one another
  - at a lower level (e.g. machine language)
  - mainly interested in relatively simple properties: memory safety and control flow

Introduction *(iii)*

- Type-theoretic approaches to PCC

  e.g. Shao *et al.*, POPL 2002, TOPLAS 2005;
  Crary and Vanderwaart, ICFP 2002

  - arbitrary program properties
  - embedding of logic "formulae as types"
  - proof-preserving compilation
  - makes the proof a part of the code
  - type (proof) inference is undecidable

Motivation

	Program verification	PCC
programmer "friendly"		
high-level proofs		
end-user safety		

Motivation

	Program verification	PCC
programmer "friendly"	✓	✗
high-level proofs		
end-user safety		

Motivation

	Program verification	PCC
programmer "friendly"	✓	✗
high-level proofs	✓	✗
end-user safety		

Motivation

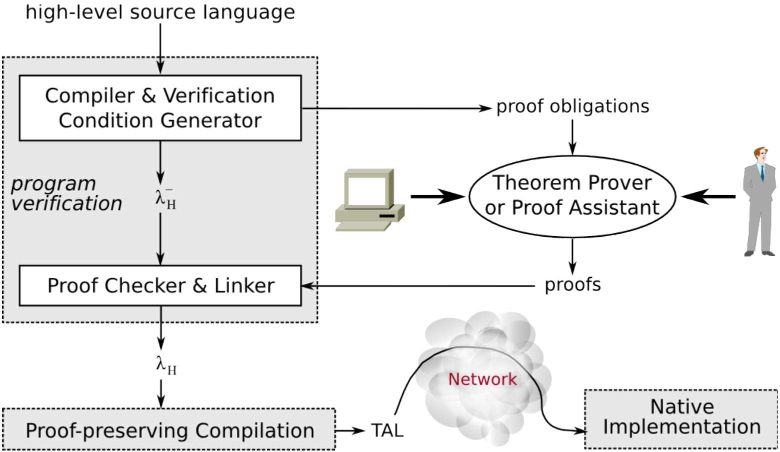
	Program verification	PCC
programmer "friendly"	✓	✗
high-level proofs	✓	✗
end-user safety	✗	✓

Motivation

	Program verification	PCC
programmer "friendly"	✓	✗
high-level proofs	✓	✗
end-user safety	✗	✓

Can we write programs in a high-level language,
provide correctness proofs for them
and then compile them
to provably correct executable code?

A hybrid system



high-level source language

Compiler & Verification Condition Generator

proof obligations

program verification

λ_H^-

Theorem Prover or Proof Assistant

Proof Checker & Linker

proofs

λ_H

Proof-preserving Compilation

TAL

Network

Native Implementation

Integer square root *(i)*

- Given $n \in \mathbb{N}$, find the greatest $r \in \mathbb{N}$ such that $r^2 \leq n$

Integer square root *(i)*

- Given $n \in \mathbb{N}$, find the greatest $r \in \mathbb{N}$ such that $r^2 \leq n$

```
int root (int n) {
  int y = 0;

  while ((y+1)*(y+1) <= n) y++;
  return y;
}
```

Integer square root *(i)*

- Given $n \in \mathbb{N}$, find the greatest $r \in \mathbb{N}$ such that $r^2 \le n$

```
//@ predicate leRoot(int r, int x) { r ≥ 0 ∧ r² ≤ x }
//@ predicate isRoot(int r, int x) { leRoot(r, x) ∧ (r + 1)² > x }
```

```
int root (int n) {
  int y = 0;

  while ((y+1)*(y+1) <= n) y++;
  return y;
}
```

Integer square root (i)

- Given $n \in \mathbb{N}$, find the greatest $r \in \mathbb{N}$ such that $r^2 \leq n$

`//@` $\mathit{predicate}\ \ \mathit{leRoot}(\mathit{int}\, r, \mathit{int}\ \ x)\ \{\ r \geq 0 \land r^2 \leq x\ \}$

`//@` $\mathit{predicate}\ \ \mathit{isRoot}(\mathit{int}\, r, \mathit{int}\ \ x)\ \{\ \mathit{leRoot}(r, x) \land (r+1)^2 > x\ \}$

`/*@ requires` $n \geq 0$

`  @ ensures` $\mathit{isRoot}(\backslash \mathit{result}, n)$

```
  @*/
int root (int n) {
  int y = 0;

  while ((y+1)*(y+1) <= n) y++;
  return y;
}
```

Integer square root *(i)*

- Given $n \in \mathbb{N}$, find the greatest $r \in \mathbb{N}$ such that $r^2 \leq n$

```
//@ predicate leRoot(int r, int x) { $r \geq 0 \wedge r^2 \leq x$ }
//@ predicate isRoot(int r, int x) { $leRoot(r, x) \wedge (r + 1)^2 > x$ }

/*@ requires $n \geq 0$
  @ ensures  $isRoot(\backslash result, n)$
  @*/
int root (int n) {
  int y = 0;
  //@ invariant leRoot(y, n)
  while ((y+1)*(y+1) <= n) y++;
  return y;
}
```

Integer square root (ii)

Proof obligations in Why/Caduceus

1. $\forall n \in \mathbb{Z}.$
   $n \geq 0 \;\Rightarrow\; leRoot(0, n)$
2. $\forall n, y \in \mathbb{Z}.$
   $n \geq 0 \;\wedge\; leRoot(y, n) \;\wedge\; (y+1)^2 \leq n \;\Rightarrow$
   $leRoot(y+1, n)$
3. $\forall n, y \in \mathbb{Z}.$
   $n \geq 0 \;\wedge\; leRoot(y, n) \;\wedge\; (y+1)^2 > n \;\Rightarrow$
   $isRoot(y, n)$

They are all automatically discharged, using the definitions of leRoot and isRoot

Integer square root *(ii)*

Proof obligations translation to $\overline{H}$

$$
\begin{array}{lll}
\texttt{root} & \rhd & \forall n{:}\mathsf{Z}.\,\forall n^*{:}(n \geq 0).\,\mathsf{sint}\,n \twoheadrightarrow \exists x{:}\mathsf{Z}.\,\exists x^*{:}\mathsf{isRoot}\,x\,n.\,\mathsf{sint}\,x \\
 & = & \texttt{poly}\ n{:}\mathsf{Z}.\ \texttt{poly}\ n^*{:}(n \geq 0).\ \texttt{lambda}\ \texttt{n}{:}\mathsf{sint}\,n. \\
 & & (\texttt{fix loop}{:}\forall y{:}\mathsf{Z}.\,\forall y^*{:}\mathsf{leRoot}\,y\,n.\,\mathsf{sint}\,y \twoheadrightarrow \\
 & & \qquad\qquad\qquad \exists x{:}\mathsf{Z}.\,\exists x^*{:}\mathsf{isRoot}\,x\,n.\,\mathsf{sint}\,x. \\
 & & \quad \texttt{poly}\ y{:}\mathsf{Z}.\ \texttt{poly}\ y^*{:}\mathsf{leRoot}\,y\,n.\ \texttt{lambda}\ \texttt{y}{:}\mathsf{sint}\,y. \\
 & & \qquad \texttt{if}\,[\clubsuit,\clubsuit]\,((\mathtt{y} + \texttt{cint}\,[1])^2 > \mathtt{n}, \\
 & & \qquad\quad p_1^*\ .\,\texttt{pack}\,(y, \texttt{pack}\,(\clubsuit, y)\ \mathsf{as}\ \exists y^*{:}\mathsf{isRoot}\,y\,n.\,\mathsf{sint}\,y)\ \mathsf{as} \\
 & & \qquad\qquad\qquad \exists x{:}\mathsf{Z}.\,\exists x^*{:}\mathsf{isRoot}\,x\,n.\,\mathsf{sint}\,x, \\
 & & \qquad\quad {p_2}^*\,.\,\texttt{loop}\ \ [y+1]\ \ [\clubsuit]\ \ (\mathtt{y} + \texttt{cint}\,[1]))) \\
 & & \quad [0]\ \ [\clubsuit]\ \texttt{cint}\,[0]
\end{array}
$$

Integer square root (iii)

Proof obligations from $\overset{-}{H}$ to H

$$
\begin{array}{lll}
\texttt{root} & \triangleright & \forall n{:}\mathsf{Z}.\,\forall n^*{:}(n \geq 0).\,\mathsf{sint}\,n \twoheadrightarrow \exists x{:}\mathsf{Z}.\,\exists x^*{:}\mathsf{isRoot}\,x\,n.\,\mathsf{sint}\,x \\
 & = & \texttt{poly}\ n{:}\mathsf{Z}.\ \texttt{poly}\ n^*{:}(n\geq 0).\ \texttt{lambda}\ \texttt{n}{:}\mathsf{sint}\,n. \\
 & & (\texttt{fix loop}{:}\forall y{:}\mathsf{Z}.\,\forall y^*{:}\mathsf{leRoot}\,y\,n.\,\mathsf{sint}\,y\ \twoheadrightarrow \\
 & & \qquad\qquad\qquad \exists x{:}\mathsf{Z}.\,\exists x^*{:}\mathsf{isRoot}\,x\,n.\,\mathsf{sint}\,x. \\
 & & \quad \texttt{poly}\ y{:}\mathsf{Z}.\ \texttt{poly}\ y^*{:}\mathsf{leRoot}\,y\,n.\ \texttt{lambda}\ \texttt{y}{:}\mathsf{sint}\,y. \\
 & & \qquad \texttt{if}\ [\mathsf{decidable}\,((y+1)^2 > n), \mathsf{gtDecidablePrf}\,(y+1)^2\,n]\ (\\
 & & \qquad\quad (\texttt{y} + \texttt{cint}\,[1])^2 > \texttt{n}, \\
 & & \qquad\quad p_1^*\ .\texttt{pack}\,(y, \texttt{pack}\,(\mathsf{conj}\,y^*\,p_1{}^*, y)\ \texttt{as} \\
 & & \qquad\qquad\qquad\qquad \exists y^*{:}\mathsf{isRoot}\,y\,n.\,\mathsf{sint}\,y)\ \texttt{as} \\
 & & \qquad\qquad\qquad \exists x{:}\mathsf{Z}.\,\exists x^*{:}\mathsf{isRoot}\,x\,n.\,\mathsf{sint}\,x, \\
 & & \qquad\quad p_2{}^*\ .\texttt{loop}\ \ [y+1]\ \ [\mathsf{conj}\,(\mathsf{Zplus\_ge\_compat}\,y\,0\,1\,0 \\
 & & \qquad\qquad\qquad\qquad\qquad (\mathsf{proj1}\,y^*)\,(\mathsf{geDecidablePrf}\,1\,0)) \\
 & & \qquad\qquad\qquad\qquad (\mathsf{Znot\_gt\_le}\,(y+1)^2\,n\,p_2{}^*)] \\
 & & \qquad\qquad\qquad (\texttt{y} + \texttt{cint}\,[1]))) \\
 & & \quad [0]\ \ [\mathsf{conj}\,(\mathsf{Z\_ge\_refl}\,0)\,(\mathsf{Zge\_le}\,n\,0\,n^*)]\ \texttt{cint}\,[0]
\end{array}
$$

Proof-preserving compilation (i)

Continuation passing style (CPS) from $_H$ to $_K$

- Functions do not return
- One more parameter: the continuation
- Jumps instead of calls
- Control flow is explicit
- Optimizing transformations can be applied

- ∼20 lines for the square root example

Proof-preserving compilation (ii)

Closure conversion from $_{K}$ to $_{C}$

- Functions only use local data
- One more parameter: the closure
- More optimizing transformations can be applied

- ∼200 lines for the square root example

Proof-preserving compilation *(iii)*

Hoisting from $_C$ to $_A$

- All functions become top-level blocks
- Memory allocation is explicit
- $\sim$200 lines for the square root example

Proof-preserving compilation *(iv)*

Typed assembly language (TAL) from $_A$ to TAL

- RISC
- We assume infinite registers
    - no spilling phase
    - trivial register allocation
- We assume a garbage collector
- $\sim$500 lines for the square root example

Proof-preserving compilation (v)

Beyond compilation

- Low Level Virtual Machine (LLVM)
- Direct translation from TAL to LLVM
- Direct translations from LLVM to native code for many architectures

x86, x86-64, PowerPC 32/64, ARM, Thumb,
IA-64, Alpha, SPARC, MIPS, CellSPU

Conclusion

- Long, long way to go...

Conclusion

- Long, long way to go...
- Most problems on the source level
  - what language(s)?
  - what logic(s)?
  - can programmers easily prove things?
  - scalability?

Conclusion

- Long, long way to go...
- Most problems on the source level
    - what language(s)?
    - what logic(s)?
    - can programmers easily prove things?
    - scalability?
- Efficient representations for certified binaries?

Conclusion

- Long, long way to go...
- Most problems on the source level
    - what language(s)?
    - what logic(s)?
    - can programmers easily prove things?
    - scalability?
- Efficient representations for certified binaries?
- Can modern compilers be proof-preserving?

Conclusion

Thank you!

- Long, long way to go...
- Most problems on the source level
  - what language(s)?
  - what logic(s)?
  - can programmers easily prove things?
  - scalability?
- Efficient representations for certified binaries?
- Can modern compilers be proof-preserving?