NOKIA Bell Labs

Designing Self-Certifying Compilers

Kedar Namjoshi
Bell Labs, Nokia

NYPLSE 2019
25 Feb 2019

NOKIA Bell Labs

Compilers are Everywhere!

- **Language Implementation**

C/C++	to	a.out
Java/Python	to	bytecode
VHDL/Verilog	to	netlist
Statecharts	to	C++

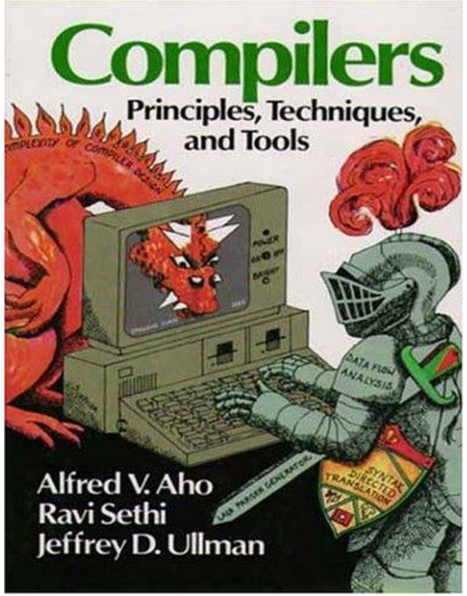
  Target behavior should be a subset of source behavior.

- **Refactoring Programs**

  Source and target behavior should be identical.

- **Layering Program Aspects**

  Target should match source behavior on all but the new aspect.

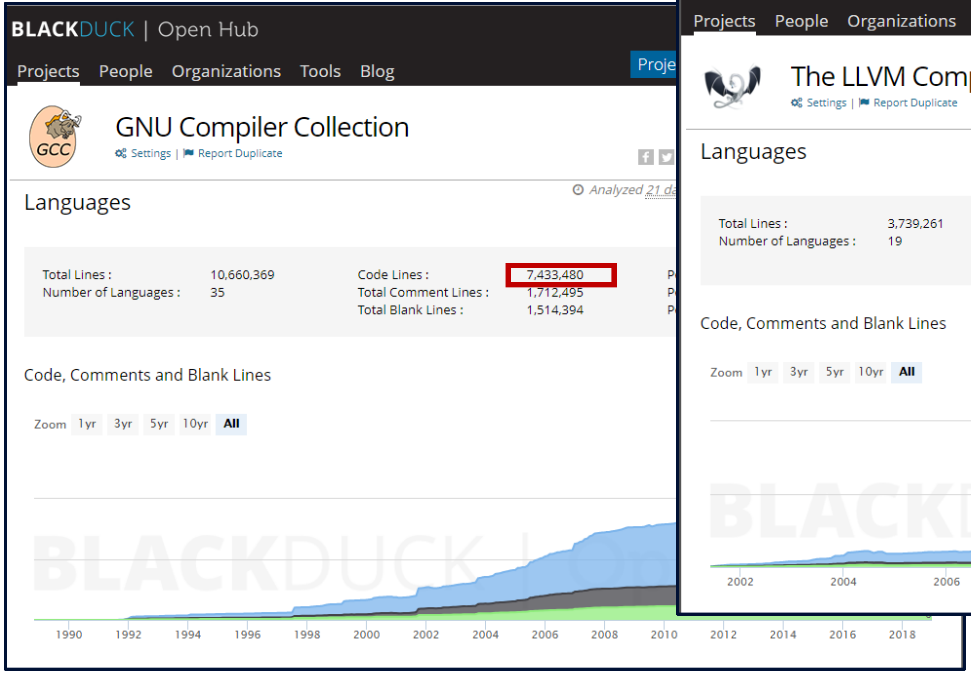


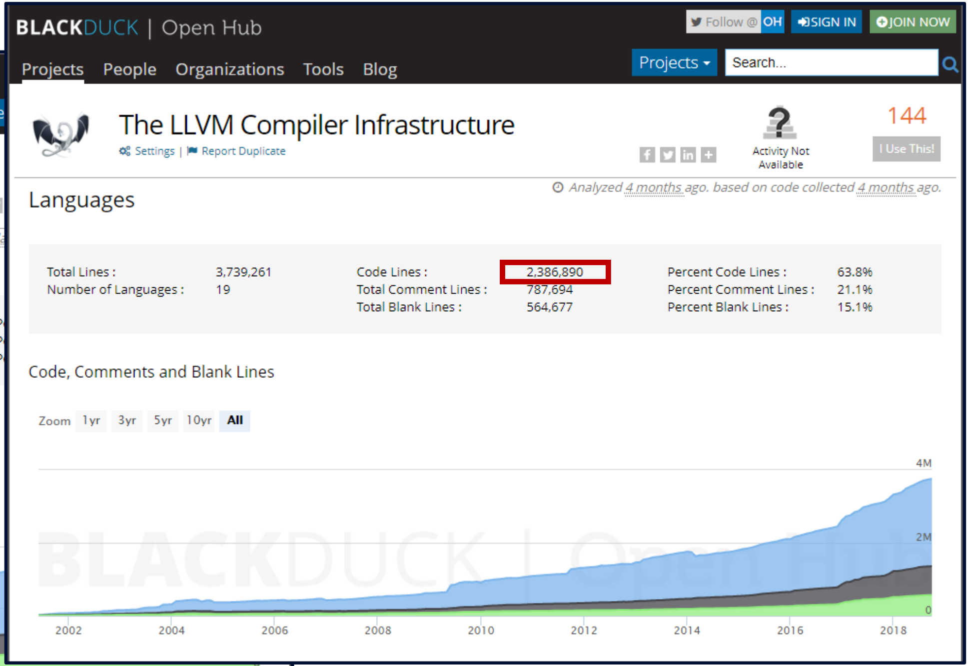
  Public

NOKIA Bell Labs

What is the Problem?

Compilers are too large and far too complex for *ex post facto* verification.





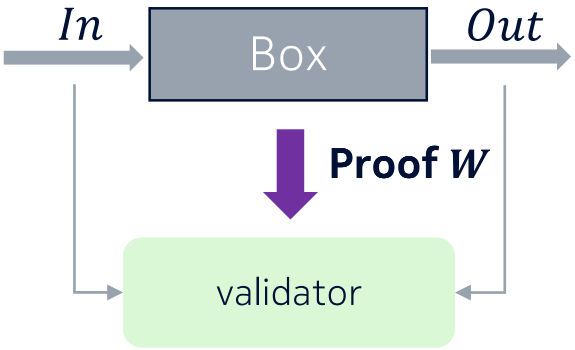
This is not a new problem!

But I called this a minor cause; the major cause is ... that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.

(*The Humble Programmer*, Edsger W. Dijkstra, 1972)

This talk: A Less-Than-Ideal Solution

Design software to be self-certifying.



Proof/Witness/Certificate

With the property: **If $valid(In, W, Out)$ then Box is operating correctly.**

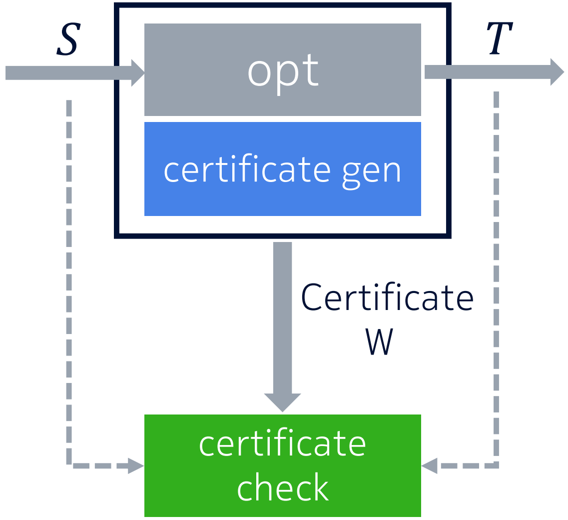
NOKIA Bell Labs

Does self-certification work for compiling?

Common optimizations have simple certificates

- Dead Store Elimination
- Constant Propagation and Folding
- Loop Unrolling (replicate loop body)
- Loop Peeling (expand first K iterations)
- Loop Invariant Code Motion
- Static Single Assignment (SSA) conversion
- CFG simplification
- Instruction Combination

The trusted code base shrinks from a few million to a few thousand lines of code.



Credible Compilation [Rinard-Marinov, 1999]

Witnessing [Namjoshi-Zuck, 2013]

What is a Certificate?

Program S		Program T
	$x = x' \;\; and \;\; y = y'$	
`x := x+y;`		`skip;`
	$y = y'$	
`y := y+1;`		`y := y+1;`
	$y = y'$	
`x := y*2;`		`x := y*2;`
	$x = x' \;\; and \;\; y = y'$	
`print(x,y)`		`print(x,y)`
	$x = x' \;\; and \;\; y = y'$	

Dead Store Elimination

There is a simple pattern to the certificate:

-- relate states with the same program location, and

-- include term $v = v'$ if and only if variable v is **live** at that location in the source program

What is a Certificate?

Program S		Program T
	$x, y, z = x', y', z'$	
`x := 3;`		`x := 3;`
	$x, y, z = x', y', z'$ $\mathit{and}\ x = 3$	
`y := x+1;`		`y := 4;`
	$x, y, z = x', y', z'$ $\mathit{and}\ x = 3\ \mathit{and}\ y = 4$	
`x := y*z;`		`x := 4*z;`
	$x, y, z = x', y', z'$ $\mathit{and}\ y = 4$	
`print(x)`		`print(x)`
	$x, y, z = x', y', z'$ $\mathit{and}\ y = 4$	

Constant Propagation

This certificate also follows a simple pattern:

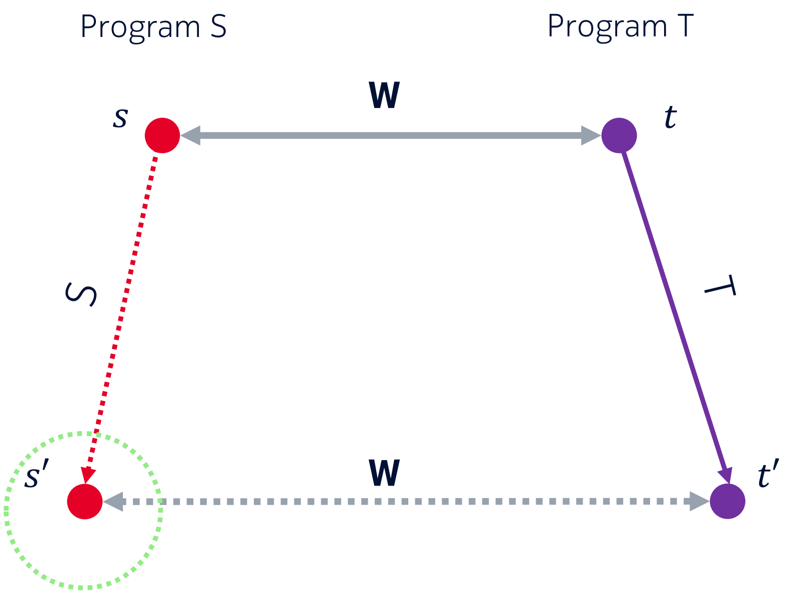
-- relate states that have the same location, and

-- include term $v = v'$ for all variables v,
and term $v = \mathrm{c}$ if v has a known constant value c at that location in the source

  Public

NOKIA Bell Labs

Validating Certificates



Soundness: Every computation of T has a W-related computation in S with identical i/o behavior.

Completeness: Every correct transformation has a valid certificate.

The certificate may include **history** (summarize past actions), allow **stuttering** (ignore inessential actions), and provide **prophecy** (guess future nondeterminism)

Generating Certificates

Program S		Program T
	$x = x' \;\; and \;\; y = y'$	
`x := x+y;`		`skip;`
	$y = y'$	
`y := y+1;`	candidate certificate	`y := y+1;`
`x := y*2;`		`x := y*2;`
	$x = x' \;\; and \;\; y = y'$	
`print(x)`		`print(x)`
	$x = x' \;\; and \;\; y = y'$	

Dead Store Elimination

There is a simple pattern to the certificate:

-- link states with the same program location, and

-- include term $v = v'$ if and only if variable v is live at that location in the source program

certificate generation scheme

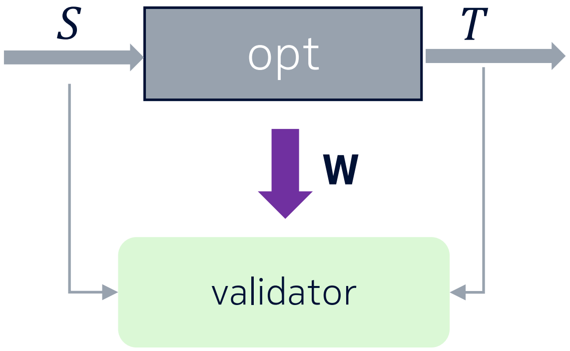
... And Checking Them

The checker turns the validity constraints into SMT queries.

$x = x' \;\; and \;\; y = y'$

x := x+y **skip**

$y = y'$

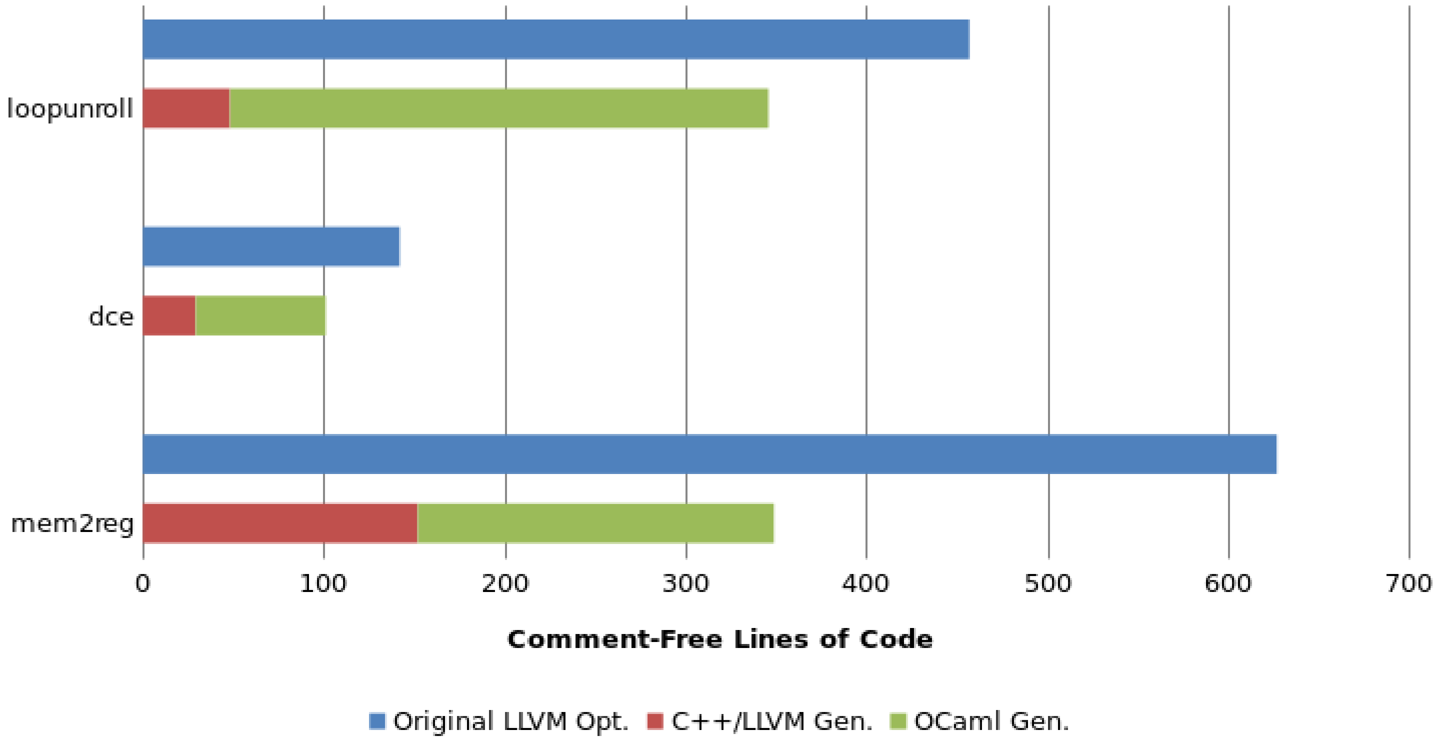


The induced SMT query is

$$
\begin{array}{ll}
x = x' \;\; and \;\; y = y' & \text{// top W} \\
and \;\; next(x') = x' \;\; and \;\; next(y') = y' & \text{// } \mathtt{skip} \\
and \;\; next(x) = x + y \;\; and \;\; next(y) = y & \text{// } \mathtt{x := x+y} \\
\Rightarrow \;\; next(y) = next(y') & \text{// bottom W}
\end{array}
$$

Current LLVM validator is ~2000 lines of OCaml code, links to the Z3 SMT solver

Certificate Generation for LLVM



  Public

NOKIA Bell Labs

Open: Concurrency-aware Optimization

Standard optimizations may be incorrect for concurrent execution.

```
x := 1;           y := 1;
signal A;   ||    await A;
await B;          print x;
print y;          signal B;
```

Dead Store Elimination

```
skip;             skip;
signal A;   ||    await A;
await B;          print x;
print y;          signal B;
```

Prints x=1, y=1

Prints x=0, y=0

How to certify concurrency-aware transformations?

Open: Secure Compilation

Correct optimizations can introduce security holes.

```
x := read_password();
use(x);
x := 0; // clear memory
// password is secure
```

Dead Store Elimination →

```
x := read_password();
use(x);
~~x := 0~~;  // dead store
// password leaks via x
```

How to certify secure compilation?

  Public NOKIA Bell Labs

To Sum Up

- Much software is too large and far too complex to be verified *ex post facto*.

  It is necessary to **build verifiability into software**.

- The design of self-certifying software is an art .... but one that is informed by deductive proof principles.

Design your software to be self-certifying!

With Many Thanks To

- Lenore Zuck and V. N. Venkatakrishnan (UIC)

- Summer interns at Bell Labs

Tim King (NYU)	Oswaldo Olivo (UT Austin)	Chaoqiang Deng (NYU)
Nimit Singhania (U. Penn)	Zvonimir Pavlinovic (NYU)	Yiji Zhang (UIC)

- DARPA and the NSF, for supporting this research

- You, for listening to this talk! ☺

Formal Acknowledgements

This work was supported, in part, by DARPA under agreement number FA8750-12-C-0166. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

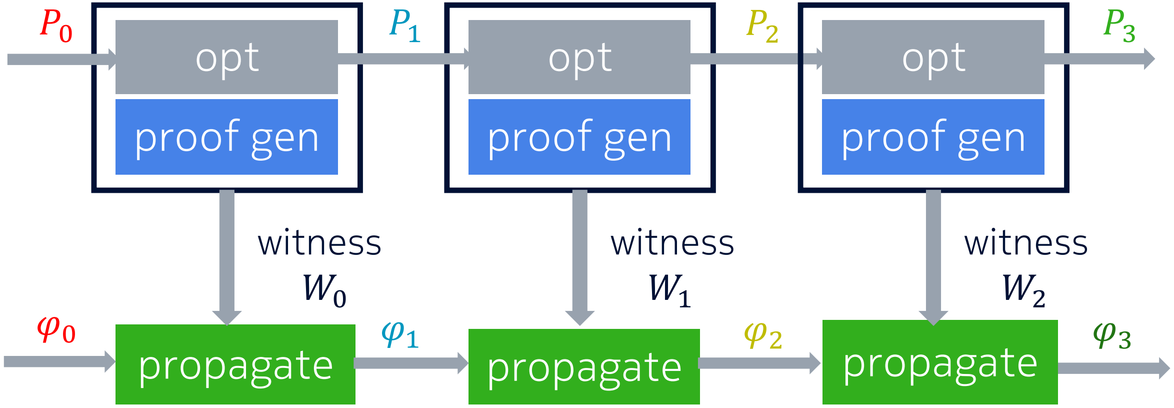
This material is based upon work supported, in part, by the National Science Foundation under Grants No. (NSF CCF-0341658, NSF CCF-1563393). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

NOKIA Bell Labs

NOKIA

Spinoff: Defensive Optimizing Compilation

Can an optimizing compiler make good use of externally generated invariants?



Theorem: If R is a refinement from T to S and φ is invariant in S, then $post(R^{-1}, \varphi)$ is invariant in T.

Compiler Validation Methods

1. **Automated Test Generation** (E.g., Csmith [2011] and EMI [2014])
   - Must generate test programs AND test inputs for those programs
   - All the advantages and the disadvantages of testing

2. **Proving correctness** once and for all (E.g., CompCert [Leroy 2006])
   - I.e., establish the theorem: $\left(\forall P\ \forall i \quad P(i) = compile(P)(i)\right)$
   - Requires considerable effort and expertise; best when designing compiler with its proof

3. **Translation Validation**: Proving correctness per program (E.g., TVOC [2005])
   - I.e., Given P, establish the theorem $\left(\forall i \quad P(i) = compile(P)(i)\right)$
   - A different heuristic per optimization; validator correctness becomes an issue