Verified Construction of Static Single Assignment Form

Sebastian Buchwald, Denis Lohner and Sebastian Ullrich

Institute for Program Structures and Data Organization, Karlsruhe Institute of Technology (KIT)

Implementation Complexity of Construction Algorithms

- Dominance frontier-based algorithms
  - Introduced in *An Efficient Method of Computing SSA Form* [Cytron et al., TOPLAS '91]
  - Used by GCC, LLVM, . . .
  - High implementation complexity
  - No existing formal verification

- Algorithms designed for simplicity
  - *Simple Generation of SSA Form* [Aycock and Horspool, CC '00]
  - Two-step algorithm:
    1. "Really Crude" phase: maximal SSA form
    2. Minimization phase

SSA Construction in Verified Compilers

Karlsruhe Institute of Technology

- Vellvm [Zhao et al., PLDI '13]
  - Formalization of the LLVM IR
  - Uses Aycock and Horspool's algorithm
    - Proof of semantic correctness
    - No proof of minimality
- CompCertSSA [Barthe et al., PLDI '13]
  - Extends the verified CompCert C compiler with an SSA midend
  - *Translation Validation* approach:
    - Untrusted implementation of Cytron et al.'s algorithm
    - Verified validator
    - No proof/validation of minimality

Construction Algorithm by Braun et al.

Simple and Efficient Construction of Static Single Assignment Form [Braun et al., CC ’13]

Simplicity

- Does not use dominance frontiers or any other analyses

Efficiency

- Shown to be on par with LLVM’s construction pass
- Used in libfirm and the Go compiler

Output size

- Pruned for all inputs
- Minimal for reducible/all inputs

Formalization

A functional variant of Braun et al.'s core algorithm in Isabelle/HOL

- CFG-based transformation
- Minimal only for reducible inputs

Algorithm split into basic parts:

1. Pruned SSA form
2. Minimization

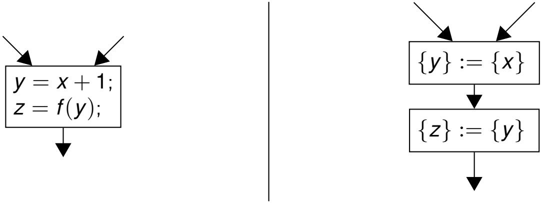
Goal

- Complete verification
- Special focus on quality guarantees

Formalization – CFG Abstraction

Abstract, minimal CFG representation:

- Graph structure
- Defs and uses per basic block
- Assumption: definite assignment
- Assumption: no intra-block data dependencies



Formalization – SSA Definition

Karlsruhe Institute of Technology

Definition (SSA CFG)

A CFG with ϕ functions is an *SSA CFG* if

- every SSA value is defined at most once
- all ϕ functions are well-formed: #arguments = #CFG predecessors
- definite assignment also holds for all ϕ functions (*strict* SSA form)
- it is in *conventional* SSA form (for Cytron et al.'s minimality definition)

Definition (Valid SSA translation)

An SSA CFG is a *valid SSA translation* of a CFG if

- it only adds ϕ functions and renames variables
- ϕ functions only reference SSA values of the same variable

Proof of Correctness

Karlsruhe Institute of Technology

Theorem (Semantics Preservation)

If G' is a valid SSA translation of G, then G and G' are semantically equivalent.

Formalization – Pruned Construction

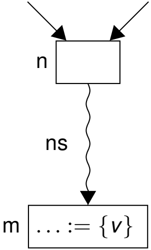
Definition (Prunedness)

An SSA CFG is in *pruned* SSA form if all ϕ functions are live.

- Cytron et al.: iterate dominance frontiers of def sites, use liveness analysis for prunedness
- Braun et al.: backwards search from use sites, implicitly pruned

lemma *phiDefNodes v* $= \{n.$
length (*predecessors n*) $> 1 \land$ — *n* is a join point
$\exists$ *ns m*. *n*–*ns*→*m* $\land$
v $\in$ *uses m* $\land$
$\forall$ *n* $\in$ *ns*. *v* $\notin$ *defs n*
$\}$

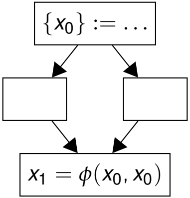
(the last three conditions: *v* is live at *n*)

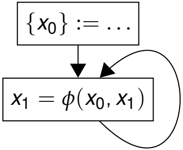


Formalization – Minimization

Karlsruhe Institute of Technology

Aycock and Horspool: for reducible inputs, sufficient to remove all *trivial* ϕ functions





Implementation

Define a graph transformation that removes a single trivial ϕ function, then close over it via a fixed-point iteration.

Proof of Minimality

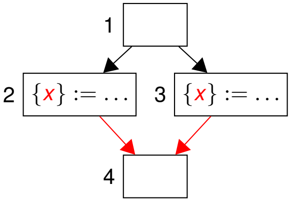
Karlsruhe Institute of Technology

Definition (Convergence property)

There is a ϕ function wherever paths from two definitions of a variable converge.

Definition (Minimality [Cytron et al.])

An SSA CFG is in *minimal* SSA form if it *only* contains ϕ functions satisfying the convergence property.



Theorem (Trivial ϕ criterion)

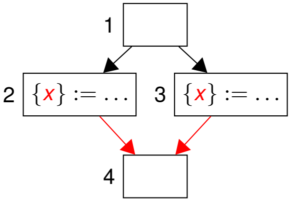
$\textit{reducible } g \wedge \neg \textit{hasTrivPhis } g \Longrightarrow \textit{cytronMinimal } g$

Isabelle proof (~1000 LoC) closely follows the handwritten proof by Braun et al. (~1.5 pages)

Proof of Minimality

A single major modification was needed:

- The handwritten proof uses the convergence property, which does not necessarily hold after pruning
- Corrected version: It is necessary to insert ϕ functions where paths from definitions of a variable converge and the variable is live



This leads to an even stronger minimality theorem:

Theorem (ϕ-count minimality)

A translated SSA CFG in both minimal and pruned SSA form has the minimum number of ϕ functions among all valid translations.

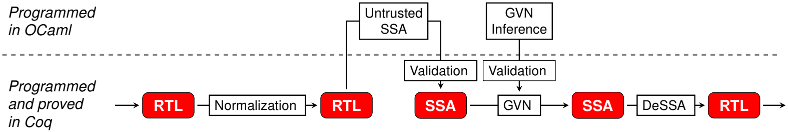
Verification Results

We proved that our formalization of Braun et al.'s algorithm computes

- ✓ an SSA CFG
- ✓ a valid translation of the input CFG
  - $\Rightarrow$ Semantic equivalence
- ✓ pruned SSA form
- ✓ minimal SSA form for reducible input CFGs

CompCertSSA Integration

Barthe et al. [PLDI '13]

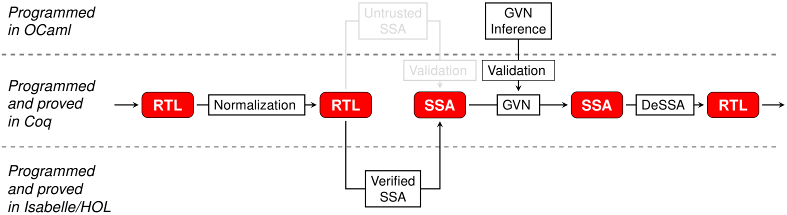


We replaced the construction + validation with an OCaml extraction of our verified Isabelle code

- Refined implementation to optimize asymptotics
- Some unverified OCaml glue code needed for interoperability

CompCertSSA Integration

Barthe et al. [PLDI '13]



We replaced the construction + validation with an OCaml extraction of our verified Isabelle code

- Refined implementation to optimize asymptotics
- Some unverified OCaml glue code needed for interoperability

CompCertSSA Integration – Performance

Benchmark	Our formalization				
	Pruned	Minimization	Glue	Total	#ϕ
177.mesa	0.46 s	0.64 s	0.20 s	1.31 s	4884
186.crafty	0.16 s	0.16 s	0.15 s	0.47 s	1169
300.twolf	0.26 s	0.40 s	0.10 s	0.76 s	2259
spass	0.79 s	1.08 s	0.53 s	2.41 s	15192

Benchmark	CompCertSSA				
	LV Analysis	ϕ Placement	Validation	Total	#ϕ
177.mesa	0.66 s	0.33 s	0.17 s	1.16 s	4884
186.crafty	0.28 s	0.30 s	0.27 s	0.84 s	1169
300.twolf	0.42 s	0.24 s	0.16 s	0.82 s	2259
spass	1.38 s	1.16 s	0.65 s	3.20 s	15168

Runtime on an Intel Core i7-3770 with 3.40 GHz and 16 GB RAM.

Conclusion

Karlsruhe Institute of Technology

Our functional implementation of Braun et al.'s algorithm is

- simple enough for a complete verification in Isabelle/HOL
- efficient for real-world inputs: on par with CompCertSSA's construction pass

We further formally proved that

- Aycock and Horspool's trivial ϕ criterion is correct
- minimality and prunedness together imply a minimum number of ϕ functions

Complete formalization available at
`http://pp.ipd.kit.edu/ssa_construction`