Verified Construction of Static Single Assignment Form

Sebastian Buchwald, Denis Lohner and Sebastian Ullrich
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

- **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
Abstract, minimal CFG representation:

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

\[ y = x + 1; \]
\[ z = f(y); \]

\[ \{y\} := \{x\} \]
\[ \{z\} := \{y\} \]
Definition (SSA CFG)
A CFG with $\phi$ functions is an SSA CFG if
- every SSA value is defined at most once
- all $\phi$ functions are well-formed: #arguments = #CFG predecessors
- definite assignment also holds for all $\phi$ 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 $\phi$ functions and renames variables
- $\phi$ functions only reference SSA values of the same variable
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 $\phi$ 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

\[ \text{lemma} \quad \phi\text{DefNodes} \quad v = \{ n. \]
\[ \quad \text{length (predecessors } n \text{)} > 1 \land \exists \ ns \ m. \ n \leftarrow ns \rightarrow m \land \]
\[ \quad v \in \text{uses } m \land \]
\[ \quad \forall \ n \in \ ns. \ v \notin \text{defs } n \}

\[ \text{n is a join point} \]
\[ \text{v is live at } n \]

\[ m \ldots := \{ v \} \]
Aycock and Horspool: for reducible inputs, sufficient to remove all *trivial* $\phi$ functions

\[
\{x_0\} := \ldots \\
\quad \downarrow \\
\quad \downarrow \\
\quad x_1 = \phi(x_0, x_0)
\]

\[
\{x_0\} := \ldots \\
\quad \downarrow \\
\quad \downarrow \\
\quad x_1 = \phi(x_0, x_1)
\]

**Implementation**

Define a graph transformation that removes a single trivial $\phi$ function, then close over it via a fixed-point iteration.
Proof of Minimality

Definition (Convergence property)
There is a $\phi$ 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 $\phi$ functions satisfying the convergence property.

Theorem (Trivial $\phi$ criterion)
$\text{reducible } g \land \neg \text{hasTrivPhis } g \implies \text{cytronMinimal } g$

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 \( \phi \) functions where paths from definitions of a variable converge and the variable is live.

This leads to an even stronger minimality theorem:

**Theorem (\( \phi \)-count minimality)**

A translated SSA CFG in both minimal and pruned SSA form has the minimum number of \( \phi \) functions among all valid translations.
We proved that our formalization of Braun et al.’s algorithm computes
✓ an SSA CFG
✓ a valid translation of the input CFG
  ⇒ Semantic equivalence
✓ pruned SSA form
✓ minimal SSA form for reducible input CFGs
CompCertSSA Integration

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
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

#### Our formalization

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Pruned</th>
<th>Minimization</th>
<th>Glue</th>
<th>Total</th>
<th>#ϕ</th>
</tr>
</thead>
<tbody>
<tr>
<td>177.mesa</td>
<td>0.46 s</td>
<td>0.64 s</td>
<td>0.20 s</td>
<td>1.31 s</td>
<td>4884</td>
</tr>
<tr>
<td>186.crafty</td>
<td>0.16 s</td>
<td>0.16 s</td>
<td>0.15 s</td>
<td>0.47 s</td>
<td>1169</td>
</tr>
<tr>
<td>300.twolf</td>
<td>0.26 s</td>
<td>0.40 s</td>
<td>0.10 s</td>
<td>0.76 s</td>
<td>2259</td>
</tr>
<tr>
<td>spass</td>
<td>0.79 s</td>
<td>1.08 s</td>
<td>0.53 s</td>
<td>2.41 s</td>
<td>15192</td>
</tr>
</tbody>
</table>

#### CompCertSSA

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>LV Analysis</th>
<th>φ Placement</th>
<th>Validation</th>
<th>Total</th>
<th>#ϕ</th>
</tr>
</thead>
<tbody>
<tr>
<td>177.mesa</td>
<td>0.66 s</td>
<td>0.33 s</td>
<td>0.17 s</td>
<td>1.16 s</td>
<td>4884</td>
</tr>
<tr>
<td>186.crafty</td>
<td>0.28 s</td>
<td>0.30 s</td>
<td>0.27 s</td>
<td>0.84 s</td>
<td>1169</td>
</tr>
<tr>
<td>300.twolf</td>
<td>0.42 s</td>
<td>0.24 s</td>
<td>0.16 s</td>
<td>0.82 s</td>
<td>2259</td>
</tr>
<tr>
<td>spass</td>
<td>1.38 s</td>
<td>1.16 s</td>
<td>0.65 s</td>
<td>3.20 s</td>
<td>15168</td>
</tr>
</tbody>
</table>

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

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 $\phi$ criterion is correct
- minimality and prunedness together imply a minimum number of $\phi$ functions

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