Minimal Static Single Assignment Form

Max Wagner Denis Lohner

February 6, 2026

Abstract

This formalization is an extension to [3]. In their work, the authors have shown that Braun et al.'s static single assignment (SSA) construction algorithm [1] produces minimal SSA form for input programs with a reducible control flow graph (CFG). However Braun et al. also proposed an extension to their algorithm that they claim produces minimal SSA form even for irreducible CFGs. In this formalization we support that claim by giving a mechanized proof.

As the extension of Braun et al.'s algorithm aims for removing so-called *redundant strongly connected components* (sccs) of ϕ functions, we show that this suffices to guarantee minimality according to Cytron et al. [2].

Contents

1 Minimality under Irreducible Control Flow **1**
1.1 Proof of Lemma 1 from Braun et al. 2
1.2 Proof of Minimality 3

1 Minimality under Irreducible Control Flow

Braun et al. [1] provide an extension to the original construction algorithm to ensure minimality according to Cytron's definition even in the case of irreducible control flow. This extension establishes the property of being *redundant-scc-free*, i.e. the resulting graph G contains no subsets inducing a strongly connected subgraph G' via ϕ functions such that G' has less than two ϕ arguments in $G \setminus G'$. In this section we will show that a graph with this property is Cytron-minimal.

Our formalization follows the proof sketch given in [1]. We first provide a formal proof of Lemma 1 from [1] which states that every redundant set of ϕ functions contains at least one redundant SCC. A redundant set of ϕ functions is a set P of ϕ functions with $P \cup \{v\} \supseteq A$, where A is the union over all ϕ functions arguments contained in P. I.e. P references at most one SSA value (v) outside P. A redundant SCC is a redundant set that is strongly connected according to the *is-argument* relation.

Next, we show that a CFG in SSA form without redundant sets of ϕ functions is Cytron-minimal.

Finally putting those results together, we conclude that the extension to Braun et al.'s algorithm always produces minimal SSA form.

theory *Irreducible*
imports *Formal-SSA.Minimality*
begin

context *CFG-SSA-Transformed*
begin

1.1 Proof of Lemma 1 from Braun et al.

To preserve readability, we won't distinguish between graph nodes and the ϕ functions contained inside such a node.

The graph induced by the ϕ network contained in the vertex set P. Note that the edges of this graph are not necessarily a subset of the edges of the input graph.

definition *induced-phi-graph g P* $\equiv \{(\varphi,\varphi').$ *phiArg g* φ $\varphi'\} \cap P \times P$

For the purposes of this section, we define a "redundant set" as a nonempty set of ϕ functions with at most one ϕ argument outside itself. A redundant SCC is defined analogously. Note that since any uses of values in a redundant set can be replaced by uses of its singular argument (without modifying program semantics), the name is adequate.

definition *redundant-set g P* $\equiv P \neq \{\} \wedge P \subseteq$ *dom* (*phi g*) $\wedge (\exists\, v' \in$ *allVars g*. $\forall\, \varphi \in P.\ \forall\, \varphi'.$ *phiArg g* $\varphi\ \varphi' \longrightarrow \varphi' \in P \cup \{v'\}$)
definition *redundant-scc g P scc* $\equiv$ *redundant-set g scc* $\wedge$ *is-scc* (*induced-phi-graph g P*) *scc*

We prove an important lemma via condensation graphs of ϕ networks, so the relevant definitions are introduced here.

definition *condensation-nodes g P* $\equiv$ *scc-of* (*induced-phi-graph g P*) ' *P*
definition *condensation-edges g P* $\equiv$ (($\lambda(x,y).$ (*scc-of* (*induced-phi-graph g P*) *x*, *scc-of* (*induced-phi-graph g P*) *y*)) ' (*induced-phi-graph g P*)) $-$ *Id*

For a finite P, the condensation graph induced by P is finite and acyclic.

lemma *condensation-finite*: *finite* (*condensation-edges g P*)

The set of edges of the condensation graph, spanning at most all ϕ nodes and their arguments (both of which are finite sets), is finite itself.

⟨*proof*⟩
auxiliary lemmas for acyclicity

lemma *condensation-nodes-edges*: (*condensation-edges g P*) $\subseteq$ (*condensation-nodes g P* $\times$ *condensation-nodes g P*)
⟨*proof*⟩

lemma *condensation-edge-impl-path*:
assumes (*a*, *b*) $\in$ (*condensation-edges g P*)
assumes ($\varphi_a \in a$)

assumes $(\varphi_b \in b)$
shows $(\varphi_a, \varphi_b) \in (\textit{induced-phi-graph } g\ P)^*$
⟨*proof*⟩

lemma *path-in-condensation-impl-path*:
assumes $(a, b) \in (\textit{condensation-edges } g\ P)^+$
assumes $(\varphi_a \in a)$
assumes $(\varphi_b \in b)$
shows $(\varphi_a, \varphi_b) \in (\textit{induced-phi-graph } g\ P)^*$
⟨*proof*⟩

lemma *condensation-acyclic*: *acyclic* (*condensation-edges g P*)
⟨*proof*⟩

Since the condensation graph of a set is acyclic and finite, it must have a leaf.

lemma *Ex-condensation-leaf*:
assumes $P \neq \{\}$
shows $\exists\, \textit{leaf}.\ \textit{leaf} \in (\textit{condensation-nodes } g\ P) \wedge (\forall\ \textit{scc}.(\textit{leaf}, \textit{scc}) \notin \textit{condensation-edges } g\ P)$
⟨*proof*⟩

lemma *scc-in-P*:
assumes *scc* ∈ *condensation-nodes g P*
shows *scc* ⊆ *P*
⟨*proof*⟩

lemma *redundant-scc-phis*:
assumes *redundant-set g P scc* ∈ *condensation-nodes g P x* ∈ *scc*
shows *phi g x* ≠ *None*
⟨*proof*⟩

The following lemma will be important for the main proof of this section. If P is redundant, a leaf in the condensation graph induced by P corresponds to a strongly connected set with at most one argument, thus a redundant strongly connected set exists.

Lemma 1. Every redundant set contains a redundant SCC.

lemma *1*:
assumes *redundant-set g P*
shows $\exists\, \textit{scc} \subseteq P.\ \textit{redundant-scc } g\ P\ \textit{scc}$
⟨*proof*⟩

1.2 Proof of Minimality

We inductively define the reachable-set of a ϕ function as all ϕ functions reachable from a given node via an unbroken chain of ϕ argument edges to unnecessary ϕ functions.

inductive-set *reachable* :: *'g* ⇒ *'val* ⇒ *'val set*
for *g* :: *'g* **and** *φ* :: *'val*
where *refl*: *unnecessaryPhi g φ* ⟹ *φ* ∈ *reachable g φ*
| *step*: *φ'* ∈ *reachable g φ* ⟹ *phiArg g φ' φ''* ⟹ *unnecessaryPhi g φ''* ⟹ *φ''* ∈ *reachable g φ*

lemma *reachable-props*:
assumes *φ'* ∈ *reachable g φ*
shows (*phiArg g*)** *φ φ'* **and** *unnecessaryPhi g φ'*
⟨*proof*⟩

We call the transitive arguments of a ϕ function not in its reachable-set the "true arguments" of this ϕ function.

definition [*simp*]: *trueArgs g φ* ≡ {*φ'*. *φ'* ∉ *reachable g φ*} ∩ {*φ'*. ∃ *φ''* ∈ *reachable g φ*. *phiArg g φ'' φ'*}

lemma *preds-finite*: *finite* (*trueArgs g φ*)
⟨*proof*⟩

Any unnecessary ϕ with less than 2 true arguments induces with *reachable g φ* a redundant set itself.

lemma *few-preds-redundant*:
assumes *card* (*trueArgs g φ*) < *2 unnecessaryPhi g φ*
shows *redundant-set g* (*reachable g φ*)
⟨*proof*⟩

lemma *phiArg-trancl-same-var*:
assumes (*phiArg g*)++ *φ n*
shows *var g φ* = *var g n*
⟨*proof*⟩

The following path extension lemma will be used a number of times in the inner induction of the main proof. Basically, the idea is to extend a path ending in a ϕ argument to the corresponding ϕ function while preserving disjointness to a second path.

lemma *phiArg-disjoint-paths-extend*:
assumes *var g r* = *V* **and** *var g s* = *V* **and** *r* ∈ *allVars g* **and** *s* ∈ *allVars g*
and *V* ∈ *oldDefs g n* **and** *V* ∈ *oldDefs g m*
and *g* ⊢ *n*−*ns*→*defNode g r* **and** *g* ⊢ *m*−*ms*→*defNode g s*
and *set ns* ∩ *set ms* = {}
and *phiArg g φ*$_r$ *r*
obtains *ns'*
where *g* ⊢ *n*−*ns*@*ns'*→*defNode g φ*$_r$
and *set* (*butlast* (*ns*@*ns'*)) ∩ *set ms* = {}
⟨*proof*⟩

lemma *reachable-same-var*:
assumes $\varphi' \in reachable\ g\ \varphi$
shows $var\ g\ \varphi = var\ g\ \varphi'$
⟨*proof*⟩

lemma *φ-node-no-defs*:
assumes $unnecessaryPhi\ g\ \varphi\ \varphi \in allVars\ g\ var\ g\ \varphi \in oldDefs\ g\ n$
shows $defNode\ g\ \varphi \neq n$
⟨*proof*⟩

lemma *defNode-differ-aux*:
assumes $\varphi_s \in reachable\ g\ \varphi\ \varphi \in allVars\ g\ s \in allVars\ g\ \varphi_s \neq s\ var\ g\ \varphi = var\ g\ s$
shows $defNode\ g\ \varphi_s \neq defNode\ g\ s$ ⟨*proof*⟩

Theorem 1. A graph which does not contain any redundant set is minimal according to Cytron et al.'s definition of minimality.

theorem *no-redundant-set-minimal*:
assumes *no-redundant-set*: $\neg(\exists\, P.\ redundant\text{-}set\ g\ P)$
shows $cytronMinimal\ g$
⟨*proof*⟩

Together with lemma 1, we thus have that a CFG without redundant SCCs is cytron-minimal, proving that the property established by Braun et al.'s algorithm suffices.

corollary *no-redundant-SCC-minimal*:
assumes $\neg(\exists\, P\ scc.\ redundant\text{-}scc\ g\ P\ scc)$
shows $cytronMinimal\ g$
⟨*proof*⟩

Finally, to conclude, we'll show that the above theorem is indeed a stronger assertion about a graph than the lack of trivial ϕ functions. Intuitively, this is because a set containing only a trivial ϕ function is a redundant set.

corollary
assumes $\neg(\exists\, P.\ redundant\text{-}set\ g\ P)$
shows $\neg redundant\ g$
⟨*proof*⟩

end

end

References

[1] M. Braun, S. Buchwald, S. Hack, R. Leißa, C. Mallon, and A. Zwinkau. Simple and efficient construction of static single assignment form. In R. Jhala and K. Bosschere, editors, *Compiler Construction*, volume 7791

of *Lecture Notes in Computer Science*, pages 102–122. Springer Berlin Heidelberg, 2013.

[2] R. Cytron, J. Ferrante, B. K. Rosen, M. N. Wegman, and F. K. Zadeck. Efficiently computing static single assignment form and the control dependence graph. *ACM Transactions on Programming Languages and Systems*, 13(4):451–490, Oct. 1991.

[3] S. Ullrich and D. Lohner. Verified construction of static single assignment form. *Archive of Formal Proofs*, Feb. 2016. http://isa-afp.org/entries/Formal_SSA.shtml, Formal proof development.