Minimal Static Single Assignment Form

Max Wagner  Denis Lohner

April 20, 2020

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 \( \phi \) 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  . . . . . . . . . . . . . . . . . . . . . . . 7

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 \( \phi \) functions such that \( G' \) has less than two \( \phi \) 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 \( \phi \) functions contains at least one redundant SCC. A redundant set of \( \phi \) functions is a set \( P \) of \( \phi \) functions with \( P \cup \{v\} \supseteq A \), where \( A \) is the union over all \( \phi \) 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 \( \phi \) 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 \( \phi \) functions contained inside such a node.

The graph induced by the \( \phi \) 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 \( \Gamma P \equiv \{ (\phi, \phi'), \text{phiArg } \phi \ \phi' \} \cap \ P \times \ P \)

For the purposes of this section, we define a "redundant set" as a nonempty set of \( \phi \) functions with at most one \( \phi \) 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 \( \Gamma P \equiv \{ P \neq \{ \} \land P \subseteq \text{dom } \text{phi } g \} \land (\exists v' \in \text{allVars } g. \ \forall \phi \in P. \ \forall \phi'. \ \text{phiArg } g \ \phi \ \phi' \rightarrow \phi' \in P \cup \{ v' \}) \)
definition redundant-scc \( \Gamma P \text{scc} \equiv \text{redundant-set } \Gamma P \text{scc} \land \text{is-scc } \text{induced-phi-graph } \Gamma P \text{scc} \)

We prove an important lemma via condensation graphs of \( \phi \) networks, so the relevant definitions are introduced here.

definition condensation-nodes \( \Gamma P \equiv \text{sec-of } (\text{induced-phi-graph } \Gamma P) \cdot P \)
definition condensation-edges \( \Gamma P \equiv ((\lambda(x,y). \ (\text{sec-of } (\text{induced-phi-graph } \Gamma P)) x, \ \text{sec-of } (\text{induced-phi-graph } \Gamma P) y)) \cdot \ (\text{induced-phi-graph } \Gamma P) = \text{Id} \)

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

lemma condensation-finite: finite (condensation-edges \( \Gamma P \))

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

proof –
let \( ?\text{phiEdges} = \{(a,b). \ \text{phiArg } a \ a \ b \} \)
have finite \( ?\text{phiEdges} \)
proof –
let \( ?\text{phiDomRan} = (\text{dom } \text{phi } g) \times \bigcup \ (\text{set } \cdot (\text{ran } (\text{phi } g))) \)
from phi-finite
have finite \( ?\text{phiDomRan} \) by (simp add: imageE phi-finite map-dom-mn-finite)
have \( ?\text{phiEdges} \subseteq ?\text{phiDomRan} \)
apply (rule subst[of \( \forall \ a \in ?\text{phiEdges}. \ a \in ?\text{phiDomRan} \))
apply (simp-all add: subset-eq[symmetric] phiArg-def)
by (auto simp: mn-def)
with (finite ?phiDomRan)
show finite ?phiEdges by (rule Finite_Set.rev-finite-subset)
qed

hence \( \bigwedge f . \text{finite} (f ^{\cdot} (\text{phiEdges} \cap (P \times P))) \) by auto
thus finite (condensation-edges g P) unfolding condensation-edges-def induced-phi-graph-def
by auto
qed

auxiliary lemmas for acyclicity

lemma condensation-nodes-edges: (condensation-edges g P) \( \subseteq \) (condensation-nodes g P \times condensation-nodes g P)
unfolding condensation-edges-def condensation-nodes-def induced-phi-graph-def
by auto

lemma condensation-edge-impl-path:
assumes \((a, b) \in (\text{condensation-edges} g P)\)
assumes \((\varphi_a \in a)\)
assumes \((\varphi_b \in b)\)
shows \((\varphi_a, \varphi_b) \in (\text{induced-phi-graph} g P)^*\)
unfolding condensation-edges-def
proof
  from assms(1)
  obtain \(x\ y\) where x-y-props:
  \((x, y) \in (\text{induced-phi-graph} g P)\)
  \(a = \text{sc-c-of} (\text{induced-phi-graph} g P) x\)
  \(b = \text{sc-c-of} (\text{induced-phi-graph} g P) y\)
  unfolding condensation-edges-def by auto
hence \(x \in a\ y \in b\) by auto

All that's left is to combine these paths.

with assms(2) x-y-props(2)
have \((\varphi_a, x) \in (\text{induced-phi-graph} g P)^*\) by (meson is-sec-connected scc-of-is-sec)
moreover with assms(3) x-y-props(3) \(\gamma \in b\)
have \((y, \varphi_b) \in (\text{induced-phi-graph} g P)^*\) by (meson is-sec-connected scc-of-is-sec)
ultimately
show \((\varphi_a, \varphi_b) \in (\text{induced-phi-graph} g P)^*\) using x-y-props(1) by auto
qed

lemma path-in-condensation-impl-path:
assumes \((a, b) \in (\text{condensation-edges} g P)^+\)
assumes \((\varphi_a \in a)\)
assumes \((\varphi_b \in b)\)
shows \((\varphi_a, \varphi_b) \in (\text{induced-phi-graph} g P)^*\)
using assms
proof (induction arbitrary: \(\varphi_b\) rule:trans-induct)
  fix \(y\ z\ \varphi_b\)
  assume \((y, z) \in (\text{condensation-edges} g P)\)
hence \( \text{is-sec} \) (\text{induced-phi-graph} \ g \ P) \ y \ \text{unfolding} \ \text{condensation-edges-def} \ \text{by auto}

hence \( \exists \varphi_y. \varphi_y \in y \) using \( \text{sec-non-empty}' \) by auto

then obtain \( \varphi_y \) where \( \varphi_y \text{-in-} y: \varphi_y \in y \) by auto

\( \varphi_y \) is-element of \( \varphi_y \) in \( y \) using \( \text{non-empty}' \)

\( \varphi_y \) is-element of \( \varphi_y \) in \( y \) using \( \text{condensation-edge-impl-path} \)

by auto

Then there must be a second SCC \( b \) on this path.

from this cyclic

obtain \( b \) where \( b \text{-on-path}: (x, b) \in (\text{condensation-edges} \ g \ P) \) \( (b, x) \in (\text{condensation-edges} \ g \ P)^+ \)

by \( \text{meson converse-tranclE} \)

\( \varphi_y \) is-element of \( \varphi_y \) in \( y \) using \( \text{condensation-edge-impl-path} \)

by auto

however, the existence of this path means all nodes in \( b \) and \( x \) are mutually reachable.

\( \varphi_x \) is-element of \( \varphi_x \) in \( x \) using \( \text{nodes-are-sec} \) \( \text{sec-non-empty}' \) \text{ex-in-conv}

by auto

then obtain \( \varphi_x \) where \( \varphi_x \) is-element of \( \varphi_x \) in \( x \) \( \varphi_b \) is-element of \( \varphi_b \) in \( b \) using \( \text{mesis} \)

\( \varphi_b \) is-element of \( \varphi_b \) in \( b \) using \( \text{nodes-are-sec} \) \( 1 \) \text{b-on-path path-in-condensation-impl-path condensation-edge-impl-path} \)

\( \varphi_b \) is-element of \( \varphi_b \) in \( b \) using \( \text{mesis} \)

\( \varphi_b \) is-element of \( \varphi_b \) in \( b \) using \( \text{nodes-are-sec} \) \( 2 \) \text{b-on-path path-in-condensation-impl-path condensation-edge-impl-path} \)

\( \varphi_b \) is-element of \( \varphi_b \) in \( b \) using \( \text{mesis} \)

\( \varphi_b \) is-element of \( \varphi_b \) in \( b \) using \( \text{nodes-are-sec} \) \( 3 \) \text{b-on-path path-in-condensation-impl-path condensation-edge-impl-path} \)

\( \varphi_b \) is-element of \( \varphi_b \) in \( b \) using \( \text{mesis} \)

\( \varphi_b \) is-element of \( \varphi_b \) in \( b \) using \( \text{nodes-are-sec} \) \( 4 \) \text{b-on-path path-in-condensation-impl-path condensation-edge-impl-path} \)

\( \varphi_b \) is-element of \( \varphi_b \) in \( b \) using \( \text{mesis} \)
This however means $x$ and $b$ must be the same SCC, which is a contradiction to the nonreflexivity of \textit{condensation-edges}.

with nodes-are-scc ϕb-elem
have $x = b$ using is-scc-unique[of induced-phi-graph $g P$] by simp
hence $(x, x) \in (\text{condensation-edges } g P)$ using b-on-path by simp
with nonrefl
show False by simp
qed

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

\textbf{lemma} $\text{Ex-condensation-leaf}$:
\begin{itemize}
  \item assumes $P \neq \{\}$
  \item shows $\exists \text{leaf}. \text{leaf} \in (\text{condensation-nodes } g P) \land (\forall \text{sec}. (\text{leaf}, \text{sec}) \notin \text{condensation-edges } g P)$
\end{itemize}
\begin{proof}
from assms obtain $x$ where $x \in \text{condensation-nodes } g P$ unfolding \text{condensation-nodes-def} by auto
show ?thesis
proof (rule wfE-min)
  from condensation-finite condensation-acyclic
  show wf ((\text{condensation-edges } g P)^{-1}) by (rule finite-acyclic-wf-converse)
next
fix \text{leaf}
assume \text{leaf-node}: \text{leaf} \in \text{condensation-nodes } g P
moreover
assume \text{leaf-is-leaf}: \text{sec} \notin \text{condensation-nodes } g P \text{ if } (\text{sec}, \text{leaf}) \in (\text{condensation-edges } g P)^{-1} \text{ for sec}
ultimately
have \text{leaf} \in \text{condensation-nodes } g P \land (\forall \text{sec}. (\text{leaf}, \text{sec}) \notin \text{condensation-edges } g P) \text{ using condensation-nodes-edges by blast}
thus $\exists \text{leaf}. \text{leaf} \in \text{condensation-nodes } g P \land (\forall \text{sec}. (\text{leaf}, \text{sec}) \notin \text{condensation-edges } g P)$ by blast
qed
fact
qed

\textbf{lemma} $\text{sec-in-P}$:
\begin{itemize}
  \item assumes $\text{sec} \in \text{condensation-nodes } g P$
  \item shows $\text{sec} \subseteq P$
\end{itemize}
\begin{proof}
have $\text{sec} \subseteq P$ if \text{y-props}: $\text{sec} = \text{sec-of } (\text{induced-phi-graph } g P)$ $n \in P$ for $n$
proof --
from \text{y-props}
show $\text{sec} \subseteq P$
proof (clarsimp simp:y-props(1); case-tac $n = x$)
fix $x$
assume different: $n \neq x$
assume $x \in \text{sec-of } (\text{induced-phi-graph } g P)$ $n$
hence \((n, x) \in (\text{induced-phi-graph } g P)^*\) by \((\text{metis is-scc-connected scc-of-is-scc node-in-scc-of-node})\)

with different

have \((n, x) \in (\text{induced-phi-graph } g P)^+\) by \((\text{metis rtranclD})\)

then obtain \(z\) where step: \((z, x) \in (\text{induced-phi-graph } g P)\) by \((\text{meson rtranclE})\)

from step

show \(x \in P\) unfolding \(\text{induced-phi-graph-def}\) by auto

qed simp

from this assms(1) have \(x \in P\) if \(x\)-node: \(x \in \text{sec}\) for \(x\)

apply –

apply \((\text{rule imageE[of sec scc-of (induced-phi-graph } g P)])\)

using condensation-nodes-def \(x\)-node by blast+

thus \(?thesis\) by clarify

qed

\textbf{Lemma redundant-scc-phis}:

\textbf{assumes} redundant-set \(g P \subseteq \text{condensation-nodes}\)

\textbf{shows} \(\phi g \neq \text{None}\)

\textbf{using} assms by \((\text{meson domI redundant-set-def scc-in-P subsetCE})\)

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.

\textbf{Lemma 1}:

\textbf{assumes} redundant-set \(g P\)

\textbf{shows} \(\exists \text{scc} \subseteq P, \text{redundant-scc } g P \subseteq \text{scc}\)

\textbf{proof} –

from assms Ex-condensation-leaf[of \(P g\)]

obtain leaf where leaf-props: leaf \(\in (\text{condensation-nodes } g P) \forall \text{ scc. (leaf, scc}) \notin \text{condensation-edges } g P\)

unfolding redundant-set-def by auto

hence is-scc \((\text{induced-phi-graph } g P)\) leaf unfolding condensation-nodes-def by auto

moreover

hence leaf \(\neq \{\}\) by \((\text{rule scc-non-empty'})\)

moreover

have leaf \(\subseteq \text{dom } (\phi g)\)

apply \((\text{rule subst-set}, \text{subset-eq})\)

using redundant-scc-phis leaf-props(1) assms(1) by auto

moreover

from assms

obtain pred where pred-props: pred \(\in \text{allVars } g \forall \phi \in P \forall \phi', \phi \text{Any } g \phi \phi' \rightarrow \phi' \in P \cup \{\text{pred}\}\)

unfolding redundant-set-def by auto

{
Any argument of a $\phi$ function in the leaf SCC which is not in the leaf SCC itself must be the unique argument of $P$

\[ \text{fix } \varphi, \varphi' \]

\[ \text{consider (in-P) } \varphi' \notin \text{leaf } \land \varphi' \in P \lor (\text{neither}) \varphi' \notin \text{leaf } \land \varphi' \notin P \cup \{\text{pred}\} \]

\[ \varphi' \notin \text{leaf } \land \varphi' \in \{\text{pred}\} \lor \varphi' \in \text{leaf by auto} \]

\[ \text{hence } \varphi' \in \text{leaf } \lor \{\text{pred}\} \text{ if } \varphi \in \text{leaf and } \phi\text{Arg } g \varphi, \varphi' \]

\[ \text{proof assses} \]

\[ \text{case in-P --- In this case leaf wasn't really a leaf, a contradiction} \]

\[ \text{moreover} \]

\[ \text{from in-P that leaf-props(1) sec-in-P[of leaf g P]} \]

\[ \text{have } (\varphi, \varphi') \in \text{induced-phi-graph g P unfolding induced-phi-graph-def by auto} \]

\[ \text{ultimately} \]

\[ \text{have } (\text{leaf, sec-of } (\text{induced-phi-graph g P } ) \varphi') \in \text{condensation-edges g P unfolding condensation-edges-def} \]

\[ \text{using leaf-props(1) that 'is-sec (induced-phi-graph g P) leaf'} \]

\[ \text{apply -} \]

\[ \text{apply clarsimp} \]

\[ \text{apply (rule conjI)} \]

\[ \text{prefer 2} \]

\[ \text{apply auto[1]} \]

\[ \text{unfolding condensation-nodes-def} \]

\[ \text{by (meson (no-types, lifting is-sec-unique node-in-sec-of-node pair-imageI sec-of-is-sec))} \]

\[ \text{with leaf-props(2)} \]

\[ \text{show ?thesis by auto} \]

\[ \text{next} \]

\[ \text{case neither --- In which case P itself wasn't redundant, a contradiction} \]

\[ \text{with that leaf-props pred-props} \]

\[ \text{have } \neg \text{redundant-set g P unfolding redundant-set-def} \]

\[ \text{by (meson rev-subsetD sec-in-P)} \]

\[ \text{with asms} \]

\[ \text{show ?thesis by auto} \]

\[ \text{qed auto --- the other cases are trivial} \]

\[ \} \]

\[ \text{with pred-props(1)} \]

\[ \text{have } \exists v' \in \text{allVars g. } \forall \varphi \in \text{leaf}. \forall \varphi', \phi\text{Arg g } \varphi, \varphi' \rightarrow \varphi' \in \text{leaf } \cup \{v'\} \text{ by auto} \]

\[ \text{ultimately} \]

\[ \text{have redundant-sec g P leaf unfolding redundant-sec-def redundant-set-def by auto} \]

\[ \text{thus ?thesis using leaf-props(1) sec-in-P by meson} \]

\[ \text{qed} \]

### 1.2 Proof of Minimality

We inductively define the reachable-set of a $\phi$ function as all $\phi$ functions reachable from a given node via an unbroken chain of $\phi$ argument edges to unnecessary $\phi$ functions.
**Lemma** reachable-props:

- **Assumes:** \( \varphi' \in \text{reachable } g \varphi \)
- **Shows:** \((\text{phiArg } g)^* \varphi \varphi'\) and \(\text{unnecessaryPhi } g \varphi'\)
- **Using assms by (induction \( \varphi'\) rule: reachable.induct) auto

We call the transitive arguments of a \( \varphi \) function not in its reachable-set the "true arguments" of this \( \varphi \) function.

**Definition** [simp]: \(\text{trueArgs } g \varphi \equiv \{ \varphi' . \varphi' \notin \text{reachable } g \varphi \} \cap \{ \varphi' . \exists \varphi'' \in \text{reachable } g \varphi' . \text{phiArg } g \varphi' \varphi'' \}\)

**Lemma** preds-finite: finite (trueArgs \( g \varphi \))

**Proof** (rule condr)

- **Assume:** infinite (trueArgs \( g \varphi \))
- **Hence:** \( a : \text{infinite } \{ \varphi' . \exists \varphi'' \in \text{reachable } g \varphi . \text{phiArg } g \varphi'' \varphi' \}\) by auto
- **Have:** \(\text{phiArg-set } \{ \varphi' . \exists \varphi . \text{phiArg } g \varphi' \varphi' \} = \bigcup \{ \text{set } \{ b . \exists a . \text{phi } g a = \text{Some } b \} \}\)

**Unfolding** phiArg-def by auto

- If the true arguments of a \( \varphi \) function are infinite in number, there must be an infinite number of \( \varphi \) functions... \(\text{have infinite } \{ \varphi' . \exists \varphi . \text{phiArg } g \varphi' \varphi' \}\)
  - **By:** (rule infinite-super[of \{ \varphi' . \exists \varphi'' \in \text{reachable } g \varphi . \text{phiArg } g \varphi'' \varphi' \}]) (auto simp: a)
- **With:** phiArg-set
- **Have:** infinite (mn (phi g)) unfolding ran-def phiArg-def by clarsimp

Which cannot be.

**Thus** False by (simp add:phi-finite map-dom-ran-finite)

**Qed**

Any unnecessary \( \varphi \) with less than 2 true arguments induces with reachable \( g \varphi \) a redundant set itself.

**Lemma** few-preds-redundant:

- **Assumes:** card (trueArgs \( g \varphi \)) < 2 unnecessaryPhi \( g \varphi \)
- **Shows:** redundant-set \( g \) (reachable \( g \varphi \))
- **Unfolding** redundant-set-def

**Proof** (intro conjI)

- **From assms show:** reachable \( g \varphi \neq \{ \}\)
  - **Using:** empty-iff reachable.intros(1) by auto
- **Next from assms(2)**
show \( \text{reachable} \ g \ \varphi \subseteq \text{dom} (\phi g) \)
by (metis \text{domIff} \text{reachable.cases subsetI unnecessaryPhi-def})
next
from \text{assms}(1)
consider \((\text{single}) \text{ card} \ (\text{trueArgs} \ g \ \varphi) = 1 \ | \ (\text{empty}) \text{ card} \ (\text{trueArgs} \ g \ \varphi) = 0\)
by force
thus \( \exists \text{pred} \in \text{allVars} \ g . \ \forall \varphi' \in \text{reachable} \ g \ \varphi . \ \forall \varphi''. \ \phi \text{Arg} \ g \ \varphi' \ \varphi'' \rightarrow \varphi'' \in \text{reachable} \ g \ \varphi \cup \{\text{pred}\}\)
proof cases
  case single
  then obtain \text{pred} \ where \ \text{pred}: \text{trueArgs} \ g \ \varphi = \{\text{pred}\} \ using \text{card-eq-1-singleton}
  by blast
  hence \text{pred} \in \text{allVars} \ g \ by \ (\text{auto intro: Int-Collect phiArg-in-allVars})
  moreover
  from \text{pred-prop}
  have \( \forall \varphi' \in \text{reachable} \ g \ \varphi . \ \forall \varphi''. \ \phi \text{Arg} \ g \ \varphi' \ \varphi'' \rightarrow \varphi'' \in \text{reachable} \ g \ \varphi \cup \{\text{pred}\}\)
  by auto
  ultimately
  show \( \exists \text{thesis} \ by \text{auto} \)
next
  case empty
  from \text{allDefs-in-allVars[of - defNode g \varphi] assms}
  have \text{phi-var}: \varphi \in \text{allVars} \ g \ \text{unfolding unnecessaryPhi-def phiDefs-def allDefs-def defNode-def phi-def trueArgs-def}
  by (clarsimp simp: \text{domIff phi-in-an})
  from \text{empty assms}(1)
  have \text{no-preds}: \text{trueArgs} \ g \ \varphi = \{\} \ by \ (\text{subst card-eq[OF \text{preds-finite}, symmetric]) auto}
  show \( \exists \text{thesis} \)
proof
  (rule \text{btexI}, rule ballI, rule allI, rule impI)
  fix \( \varphi' \ \varphi'' \)
  assume \( \text{phis-props}: \varphi' \in \text{reachable} \ g \ \phi \text{Arg} \ g \ \varphi' \ \varphi'' \)
  with \text{no-preds}
  have \( \varphi'' \in \text{reachable} \ g \ \varphi \)
  unfolding \text{trueArgs-def}
  proof
  from \text{phis-props}
  have \( \varphi'' \in \{\varphi', \exists \varphi''' \in \text{reachable} \ g \ \phi \text{Arg} \ g \ \varphi'' \ \varphi'\} \ by \text{auto} \)
  with \text{phis-props no-preds}
  show \( \varphi'' \in \text{reachable} \ g \ \varphi \) \ unfolding \text{trueArgs-def} \ by \text{auto}
  qed
  thus \( \varphi'' \in \text{reachable} \ g \ \varphi \cup \{\varphi\} \) \ by \text{simp}
  qed \ (\text{auto simp: phi-var})
  qed

lemma \text{phiArg-trancel_same-var:}
assumes \((\phi \text{Arg} \ g)^{++} \ \varphi \ n\)
shows \( \nu g \varphi = \nu g n \)

using assms

apply (induction rule: translp-induct)
  apply (rule phiArg-same-var[symmetric])
  apply simp
using phiArg-same-var by auto

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 \( \varphi \) argument to the corresponding \( \varphi \) function while preserving disjointness to a second path.

lemma phiArg-disjoint-paths-extend:
assumes \( \nu g r = V \) and \( \nu g s = V \) and \( r \in \text{allVars } g \) and \( s \in \text{allVars } g \) and \( V \in \text{oldDefs } g n \) and \( V \in \text{oldDefs } g m \) and \( g \vdash n - \text{defNode } g \ r \) and \( g \vdash m - \text{defNode } g \ s \) and \( \text{set } ns \cap \text{set } ms = \{ \} \) and \( \phi_A \ r g \) obtains \( ns' \) where \( g \vdash n - \text{defNode } g \ r \) and \( \text{set } (\text{butlast } (ns@ns')) \cap \text{set } ms = \{ \} \)
proof (cases \( r = \varphi_r \))
  case (True)
  If the node to extend the path to is already the endpoint, the lemma is trivial.

with assms (7,8,9) in-set-butlastD have \( g \vdash n - \text{defNode } g \ r \) set (butlast (ns@ns')) \( \cap \) set ms = \{ \} by simp-all fastforce with that show ?thesis .
next case False
It suffices to obtain any path from \( r \) to \( \varphi_r \). However, since we'll need the corresponding predecessor of \( \varphi_r \) later, we must do this as follows:

from assms (10) have \( \varphi_r \in \text{allVars } g \) unfolding phiArg-def
  by (metis allDefs-in-allVars phiDefs-in-allDefs phi-def phi-phiDefs phis-in-=
with assms (10) obtain \( rs' \) pred_r where \( rs' \)-props: \( g \vdash \text{defNode } g \ r - rs' \rightarrow \text{pred}_{\varphi_r} \) old.EntryPath \( g \ r \in \text{phiUses } g \) pred_{\varphi_r} \( \in \) set (old.predecessors \( g \) (defNode \( g \) \( \varphi_r \)))
  by (rule phiArg-path-ex')

define \( rs \) where \( rs = rs@[\text{defNode } g \ \varphi_r] \)
from \( rs' \)-props (2,1) old.EntryPath-distinct old.path2-hd
have \( rs' \)-loopfree: \( \text{defNode } g \ r \notin \text{set } (tl rs') \) by (simp add: Miscdistinct-hd-tl)

from False assms have \( \text{defNode } g \ \varphi_r \neq \text{defNode } g \ r \)
apply --
apply (rule phiArg-distinct-nodes)
apply (auto intro:phiArg-in-allVars)[2]

10
unfolding phiArY-def by (metis allDefs-in-allVars phiDefs-in-allDefs phi-def phi-phiDefs phi-in-cn)

from rs'-props
have rs-props: g ⊢ defNode g r − rs → defNode g φ_r length rs > 1 defNode g r /∈ set (tl rs)
  apply (subgoal-tac defNode g r = hd rs')
  prefer 2 using rs'-props(1)
  apply (rule old.path2-hd)
  using old.path2-snoc old.path2-def rs'-props(1) rs-def rs'-loopfree (defNode g φ_r ≠ defNode g r) by auto

show thesis
proof (cases set (butlast rs) ∩ set ms = { })
  case inter-empty: True
  if the intersection of these is empty, tl rs is already the extension we're looking for

  show thesis
  proof (rule that)
    show set (butlast (ns @ tl rs)) ∩ set ms = { }
    proof (rule contr, simp only: ex-in-conv[symmetric])
      assume 3 x. x ∈ set (butlast (ns @ tl rs)) ∩ set ms
      then obtain x where x-props: x ∈ set (butlast (ns @ tl rs)) x ∈ set ms
        by auto
        with rs-props(2)
        consider (in-ns) x ∈ set ns | (in-rs) x ∈ set (butlast (tl rs)) by (metis Un-iff butlast-append in-set-butlastD set-append)
        thus False
        apply (cases)
        using x-props(2) assms(9)
        apply (simp add: disjoint-elem)
        by (metis x-props(2) inter-empty in-set-I list butlast-td disjoint-iff-not-equal)
    qed
    qed (auto intro:assms(7) rs-props(1) old.path2-app)
  next
  case inter-ex: False
  if the intersection is nonempty, there must be a first point of intersection i

  from inter-ex assms(7,8) rs-props
  obtain i ri where ri-props: g ⊢ defNode g r−ri → i ∈ set ms ∀ n ∈ set (butlast ri). n /∈ set ms prefix ri rs
    apply −
    apply (rule old.path2-split-first-prop[of g defNode g r rs defNode g φ_r, where P=λm. m ∈ set ms])
    apply blast
    apply (metis disjoint-iff-not-equal in-set-butlastD)
    by blast
    with assms(8) old.path2-prefix-ex
obtain \( ms' \) where \( ms' \)-props: \( g \vdash m \rightarrow -ms' \rightarrow i \) prefix \( ms' \) \( ms \notin \text{set} (\text{butlast} ms') \) by blast

We proceed by case distinction:

- if \( i = \text{defNode} g \varphi_r \), the path \( ri \) is already the path extension we're looking for
- Otherwise, the fact that \( i \) is on the path from \( \phi \) argument to the \( \phi \) itself leads to a contradiction. However, we still need to distinguish the cases of whether \( m = i \)

consider \( (ri \text{-is-valid}) \ i = \text{defNode} g \varphi_r \ | \ (m \text{-same}) \ i \neq \text{defNode} g \varphi_r \ m = i \ | \ (m \text{-differ}) \ i \neq \text{defNode} g \varphi_r \ m \neq i \) by auto

thus thesis

proof (cases)

\text{case } ri \text{-is-valid}

\( ri \) is a valid path extension.

with assms(7) \( ri \)-props(1)

have \( g \vdash n \rightarrow ns \circ (\text{tl} ri) \rightarrow \text{defNode} g \varphi_r \) by auto

moreover

have \( \text{set} (\text{butlast} (ns \circ (\text{tl} ri))) \cap \text{set} ms = \{\} \)

proof (rule contr)

assume contr: \( \text{set} (\text{butlast} (ns \circ (\text{tl} ri))) \cap \text{set} ms \neq \{\} \)

from this

obtain \( x \) where \( x \)-props: \( x \in \text{set} (\text{butlast} (ns \circ (\text{tl} ri))) \ x \in \text{set} ms \) by auto

with assms(9) have \( x \notin \text{set} ns \) by auto

with \( x \)-props \( (g \vdash n \rightarrow ns \circ (\text{tl} ri) \rightarrow \text{defNode} g \varphi_r) \ (\text{defNode} g \varphi_r \neq \text{defNode} g \ r) \) assms(7)

have \( x \in \text{set} (\text{butlast} (\text{tl} ri)) \)

by (metis Un-iff append-Nil2 butlast-append old.path2-last set-append)

with \( x \)-props(2) \( ri \)-props(3)

show False by (metis FormalSSA-Misc.in-set-LD List.butlast-tl)

qed

ultimately

show thesis by (rule that)

next

\text{case } m \text{-same}

If \( m = i \), we have, with \( m \), a variable definition on the path from a \( \phi \) function to its argument. This constitutes a contradiction to the conventional property.

note \( rs' \)-props(1) \( rs' \)-loopfree

moreover have \( r \in \text{allDefs} g \ (\text{defNode} g r) \) by (simp add: assms(3))

moreover from \( rs' \)-props(3) have \( r \in \text{allUses} g \text{pred}_{\varphi_r} \text{unfolding allUses-def} \)

by simp

moreover
from rs-pr ops(1) m-i-same rs-def ri-pr ops(1,2,4) \(\text{defNode} \ g \ \varphi, \neq \ \text{defNode} \ g \ \mid\)
have \(m \in \text{set} \ (tl \ rs)\)
by (metis disjoint-elem hd-append in-hd-or-tl-conv in-prefix list.set(1) old.path2-hd old.path2-last old.path2-last-in-ns prefix-snoc)

moreover
from assms(6) obtain def_m where def_m \in \text{allDefs} \ g \ m \ \var \ g \ \text{def_m} = V
unfolding oldDefs-def using defs-in-allDefs by blast
ultimately
have \(\var \ g \ \text{def_m} \neq \ \var \ g \ r\) by (rule conventional, simp-all)
with \(\langle \var \ g \ \text{def_m} = V \rangle\) assms(1)
have False by simp
thus \(?thesis\) by simp

next
  case m-i-diifer
If \(m \neq i\), \(i\) constitutes a proper path convergence point.

  have old.pathsConverge g m ms' n (ns @ tl ri) i
  proof (rule old.pathsConvergeI)
    show \(1 < \text{length} \ ms'\) using m-i-diifer ms'-props old.path2-nontriv by blast
  next
    show \(1 < \text{length} \ (ns @ tl ri)\)
      using ri-pr ops old.path2-nontriv assms(9) by (metis assms(7) disjoint-elem old.path2-app old.path2-hd-in-ns)
    next
      show set (butlast ms') \cap set (butlast (ns @ tl ri)) = {}
    proof (rule condr)
      assume set (butlast ms') \cap set (butlast (ns @ tl ri)) \neq {}
      then obtain \(i'\) where i'-props: \(i' \in \text{set} \ (butlast ms')\) i' \in set (butlast (ns @ tl ri)) by auto
      with ms'-props(2)
      have i'-not-in-ms: \(i' \in \text{set} \ (butlast ms)\) by (metis in-set-butlast-appendI prefixE)
    with assms(9)
    show False
    proof (cases i' \neq set ns)
      case True
      with i'-props(2)
      have i' \in set (butlast (tl ri))
        by (metis Un_iff butlast-append in-set-butlastD set-append)
      hence i' \in set (butlast ri) by (simp add:in-set-ld List.butlast-ld)
      with i'-not-in-ms ri-pr ops(3)
      show False by (auto dest: in-set-butlastD)
    qed (meson disjoint-elem in-set-butlastD)
    qed
  qed (auto intro: assms(7) ri-pr ops(1) old.path2-app ms'-props(1))
At this intersection of paths we can find a $\phi$ function.

\[ \text{from this assms(6,5)} \]
\[ \text{have necessaryPhi g V i by (rule necessaryPhiI)} \]

Before we can conclude that there is indeed a $\phi$ at $i$, we have to prove a couple of technicalities...

\[ \text{moreover} \]
\[ \text{from m-i-differ ri-props(1,4) rs-def old.path2-last prefix-snoc} \]
\[ \text{have ri-\textquoteleft{}-prefix: prefix ri rs'} by fastforce} \]
\[ \text{then obtain rs'\textquoteleft{}-rest where rs'\textquoteleft{}-rest-prop: rs'} = ri@rs'\textquoteleft{}-rest using prefixE by auto} \]
\[ \text{from old.path2-last[OF ri-props(1)] last-snoc[of - i] obtain tmp where ri = tmp@[i]} \]
\[ \text{apply (subgoal-tac ri \neq [])} \]
\[ \text{prefer \$2} \]
\[ \text{using ri-props(1) apply (simp add: old.path2-not-Nil)} \]
\[ \text{apply (rule-tac that)} \]
\[ \text{using append-butlast-last-id[symmetric] by auto} \]
\[ \text{with rs'\textquoteleft{}-rest-prop have rs'\textquoteleft{}-rest-def: rs'} = tmp@[i\#rs'\textquoteleft{}-rest by auto} \]
\[ \text{with rs'\textquoteleft{}-props(1) have g \vdash i -i\#rs'\textquoteleft{}-rest \rightarrow pred_{\phi_r}} \]
\[ \text{by (simp add: old.path2-split)} \]
\[ \text{moreover} \]
\[ \text{note \langle var g r = V \rangle simp} \]
\[ \text{from rs'\textquoteleft{}-props(3)} \]
\[ \text{have r \in allUses g pred_{\phi_r}, unfolding allUses-def by simp} \]

\[ \text{moreover} \]
\[ \text{from 'defNode g r \notin set (tl rs')} : rs'\textquoteleft{}-rest-def} \]
\[ \text{have defNode g r \notin set rs'\textquoteleft{}-rest by auto} \]
\[ \text{with \langle g \vdash i -i\#rs'\textquoteleft{}-rest \rightarrow pred_{\phi_r} \rangle} \]
\[ \text{have } \bigwedge x. x \in set rs'\textquoteleft{}-rest \implies r \notin allDefs g x \]
\[ \text{by (metis defNode-eq list.distinct(1) list.sel(3) list.set-cases old.path2-cases old.path2-in-cn)} \]

\[ \text{moreover} \]
\[ \text{from assms(7,9) \langle g \vdash i -i\#rs'\textquoteleft{}-rest \rightarrow pred_{\phi_r} \rangle ri-props(2)} \]
\[ \text{have r \notin defs g i} \]
\[ \text{by (metis defNode-eq defs-in-allDefs disjoint-elem old.path2-hd-in-cn old.path2-last-in-ns)} \]

\[ \text{ultimately} \]

The convergence property gives us that there is a $\phi$ in the last node fulfilling necessaryPhi on a path to a use of $r$ without a definition of $r$. Thus $i$ bears a $\phi$ function for the value of $r$.

\[ \text{have } \exists y. \text{phis (g, i, r) = Some y} \]
\[ \text{by (rule convergence-prop \textquoteleft{}where g=g and n=i and v=r and ns=i\#rs'\textquoteleft{}-rest, simplified\textquoteleft{)}} \]

\[ \text{moreover} \]
\[ \text{from \langle g \vdash n\#ns \rightarrow defNode g r \rangle have defNode g r \in set ns by auto} \]
with \( \{ i \in \text{set } ms \mid i \cap \text{set } ms = \{\} \} \) have \( i \neq \text{defNode } g \) by auto

moreover

from \( \text{ms}'-\text{props}(1) \) have \( i \in \text{set } (\alpha\text{ng}) \) by auto

moreover

have \( \text{defNode } g \in \text{set } (\alpha\text{ng}) \) by (simp add: assms(3))

However, we now have two definitions of \( r \): one in \( i \), and one in \( \text{defNode } g \) \( r \), which we know to be distinct. This is a contradiction to the \text{allDefs-disjoint}-property.

ultimately have \( \text{False} \)

using \text{allDefs-disjoint} \[ \text{where } g=g \text{ and } n=i \text{ and } m=\text{defNode } g \text{ r} \]

unfolding \text{allDefs-def} \text{ phiDefs-def}

apply clarsimp

apply (erule_tac c=r in equalityCE)

using \text{phi-def} \text{ phis-phi} by auto

thus \( ?\text{thesis} \) by simp

qed

qed

lemma \text{reachable-same-var}:

assumes \( \psi' \in \text{reachable } g \psi \)

shows \( \text{var } g \psi = \text{var } g \psi' \)

using \text{assms} by (metis \text{Nitpick瞻rtranclp-unfold phiArg-trancel-same-var reachable-props}(1))

lemma \( \psi\text{-node-no-defs} \):

assumes \( \text{unnecessaryPhi } g \psi \psi \in \text{allVars } g \psi \text{ var } g \psi \in \text{oldDefs } g \psi n \)

shows \( \text{defNode } g \psi \neq \psi n \)

using \text{assms} simpleDefs-phiDefs-var-disjoint \text{ defNode}(1) \text{ not-None-eq phi-phiDefs}

unfolding \text{unnecessaryPhi-def} by auto

lemma \text{defNode-differ-aux}:

assumes \( \varphi_s \in \text{reachable } g \varphi \varphi \in \text{allVars } g \varphi s \in \text{allVars } g \varphi_s \neq \psi s \text{ var } g \varphi = \text{var } g s \)

shows \( \text{defNode } g \varphi_s \neq \text{defNode } g s \) unfolding reachable-def

proof (rule ccontr)

assume \( \neg \text{defNode } g \varphi_s \neq \text{defNode } g s \)

hence \( \text{eq: defNode } g \varphi_s = \text{defNode } g s \) by simp

from \text{assms}(1)

have \( \text{vars-eq: var } g \psi = \text{var } g \varphi_s \)

apply ...

apply (cases \( \varphi = \varphi_s \))

apply simp

apply (rule \text{phiArg-trancel-same-var})

apply (drule reachable-props)

unfolding reachable-def by (meson \text{IntD1 mem-Collect-eq rtranclpD})
have \( \varphi_s \in \text{allVars} \): \( \varphi_s \) \( \in \text{allVars} \) unfolding \( \text{reachable-def} \)

proof (cases \( \varphi = \varphi_s \))

- case \( \text{False} \)
  - with \( \text{assms}(1) \)
  - obtain \( \varphi' \) where \( \text{phiArg \ g \ \varphi' \ \varphi_s} \) by (metis \( \text{rtmayclp.cases \ reachable-defs}(1) \))
  - thus \( \varphi_s \in \text{allVars} \) \( \text{g \ \by (rule \phiArg-in-allVars) } \)

  qed

next

- case eq: \( \text{True} \)
  - with \( \text{assms}(2) \)
  - show \( \varphi_s \in \text{allVars} \) \( \text{g \ \by (subst eq[symmetric]) } \)
  qed

from eq \( \varphi_s \in \text{allVars} \) \( \text{assms}(3.4) \)

have \( \text{var} \ g \ \varphi_s \neq \text{var} \ g \ s \) \( \text{by } - \ (\text{rule } \text{defNode-var-disjoint}) \)

with \( \text{vars-eq \ assms}(5) \)

show \( \text{False} \) \( \text{by } \text{auto} \)
qed

**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: \( \neg(\exists P. \text{redundant-set } \text{g } P) \)

**shows** cytronMinimal \( g \)

**proof** (rule \text{ccontr})

- assume \( \neg \text{cytronMinimal } g \)

  Assume the graph is not Cytron-minimal. Thus there is a \( \varphi \) function which does not sit at the convergence point of multiple liveness intervals.

  then obtain \( \varphi \) where \( \varphi \text{-props: unnecessaryPhi } \text{g } \varphi \in \text{allVars } \varphi \in \text{reachable } g \ \varphi \)
  using cytronMinimal-def unnecessaryPhi-def reachable-def unnecessaryPhi-def reachable-infos by \text{auto}

  We consider the reachable-set of \( \varphi \). If \( \varphi \) has less than two true arguments, we know it to be a redundant set, a contradiction. Otherwise, we know there to be at least two paths from different definitions leading into the reachable-set of \( \varphi \).

  consider (nontrivial) \( \text{card (trueArgs g } \varphi) \geq 2 \) \( \text{| (trivial) card (trueArgs g } \varphi) < 2 \) using \text{linorder-not-le by auto}

  thus \( \text{False} \)

  proof cases
  - case trivial
    - from this \( \varphi \text{-props}(1) \)
      - have redundant-set \( g \) \( \text{(reachable } g \ \varphi) \) \( \text{by } \text{(rule few-preds-redundant) } \)
      - with no-redundant-set
      - show \( \text{False} \) \( \text{by } \text{simp} \)
  next
  - case nontrivial
If there are two or more necessary arguments, there must be disjoint paths from Defs to two of these $\phi$ functions.

then obtain $r s \varphi_r, \varphi_s$ where assign-nodes-props:

$r \neq s \varphi_r \in \text{reachable } g \varphi_s \in \text{reachable } g \varphi$

$\neg \text{unnecessaryPhi } g r \neg \text{unnecessaryPhi } g s$

$r \in \{ n. (\text{phiArg } g)^{++} \varphi n \} s \in \{ n. (\text{phiArg } g)^{++} \varphi n \}$

$\text{phiArg } g \varphi_r \text{ phiArg } g \varphi_s s$

apply simp

apply (rule set-take-two[of nontrivial])

apply simp

by (meson reachable-intros(2) reachable-props(1) tranclp-tranclp-tranclp tranclp-r-into-trancl tranclp-tranclp-tranclp-tranclp)

moreover from assign-nodes-props

have $\varphi-r-s-uneq: \varphi \neq r \varphi \neq s$ using $\varphi$-props by auto

moreover from assign-nodes-props this

have $r-s-in-tranclp: (\text{phiArg } g)^{++} \varphi r (\text{phiArg } g)^{++} \varphi s$

by (meson mem-Collect-eq tranclpD) (meson assign-nodes-props(7) $\varphi-r-s-uneq(2)$ mem-Collect-eq tranclpD)

from this

obtain $V$ where $V$-props: $\forall g r = V \forall g s = V \forall g \varphi = V$ by (metis phiArg-tranclp-same-var)

moreover from $r-s-in-tranclp$

have $r-s-allVars: r \in \text{allVars } g s \in \text{allVars } g$ by (metis phiArg-in-allVars tranclp-cases)+

moreover from $V$-props defNode-var-disjoint $r-s-allVars$ assign-nodes-props(1)

have $r-s-defNode-distinct: \text{defNode } g r \neq \text{defNode } g s$ by auto

ultimately obtain $n ns m ms$ where $r-s-path$-props: $V \in \text{oldDefs } g n g \vdash n - ns \rightarrow \text{defNode } g r V \in \text{oldDefs } g m g \vdash m - ms \rightarrow \text{defNode } g s$

set $ns \cap \text{set } ms = \{ \}$ by (auto intro: unnecessaryPhi-disjoint-paths[of $g r$ $s$])

have $n-m-distinct: n \neq m$

proof (rule ccontr)

assume $n-m: \neg n \neq m$

with $r-s-path$-props(2) old.path2-hd-in-ns

have $n \in \text{set } ns$ by blast

moreover from $n-m$ $r-s-path$-props(4) old.path2-hd-in-ns

have $n \in \text{set } ms$ by blast

ultimately show False using $r-s-path$-props(5) by auto

qed

These paths can be extended into paths reaching $\phi$ functions in our set.

from $V$-props $r-s-allVars$ $r-s-path$-props assign-nodes-props
obtain \textbf{rs} where \textbf{rs-props}: \( g \vdash n - \text{ns}@\text{rs} \rightarrow \text{defNode } g \phi_r \) \( \text{set } (\text{butlast } (\text{ns}@\text{rs})) \)
\( \cap \text{set } \text{ms} = \{ \} \)
\textbf{using \phiArg-disjoint-paths-extend by blast}

(In fact, we can prove that \( \text{set } (\text{ns}@\text{rs}) \cap \text{set } \text{ms} = \{ \} \), which we need for the next path extension.)

\textbf{have \text{defNode } g \phi_r \notin \text{set } \text{ms}}
\textbf{proof (rule ccontr)}
\textbf{assume \phi_r-in-ms: \( \neg \text{defNode } g \phi_r \notin \text{set } \text{ms} \)}
\textbf{from this r-s-path-props(4) obtain \text{ms'} where ms'-props: \( g \vdash m - \text{ms'} \rightarrow \text{defNode } g \phi_r \) prefix ms' by}
\( -(\text{rule old.path2-prefix-ex } \text{of } g m \text{ ms defNode } g s \text{ defNode } g \phi_r), \text{auto} \)

\textbf{have \text{old.path2Converge } g \cap \text{set } (\text{butlast } (\text{ns}@\text{rs})) \cap \text{set } (\text{butlast } \text{ms'}) = \{ \} \)}
\textbf{proof (rule old.path2ConvergeI)}
\textbf{show \text{set } (\text{butlast } (\text{ns}@\text{rs})) \cap \text{set } (\text{butlast } \text{ms'}) = \{ \} \)}
\textbf{proof (rule ccontr)}
\textbf{assume \text{set } (\text{butlast } (\text{ns}@\text{rs})) \cap \text{set } (\text{butlast } \text{ms'}) \neq \{ \} \)}
\textbf{then obtain \text{c where c-props: c \in \text{set } (\text{butlast } (\text{ns}@\text{rs})) \cap \text{set } (\text{butlast } \text{ms'}) \text{ by auto}}}
\textbf{from this(2) \text{ms}-props(2) have \text{c \in \text{set } ms} by (simp add: in-prefix in-set-butlastD)}
\textbf{with \text{c-props}(1) \text{rs-props}(2) show \text{False by auto}}
\textbf{qed}

\textbf{next}
\textbf{have m-n-\phi_r-differ: n \neq \text{defNode } g \phi_r m \neq \text{defNode } g \phi_r}
\textbf{using assign-nodes-props(2,3,4,5) V-vars r-s-path-props \phi_r-in-ms}
\textbf{apply fastforce}
\textbf{using V-vars(1) \phi_r-in-ms assign-nodes-props(8) old.path2-in-v \phiArg-def phiArg-same-var r-s-path-props(3,4) simpleDefs-phiDefs-var-disjoint}
\textbf{by auto}
\textbf{with \text{ms}-props(1) show 1 < \text{length } \text{ms'} using old.path2-nontriv by simp}
\textbf{from m-n-\phi_r-differ rs-props(1) show 1 < \text{length } (\text{ns}@\text{rs}) using old.path2-nontriv by blast}
\textbf{qed (auto intro: rs-props set-mono-prefix ms'-props)}
\textbf{with V-vars r-s-path-props have necessaryPhi' g \phi_r unfolding necessaryPhi-def using assign-nodes-props(8) phiArg-same-var by auto}
\textbf{with reachable-vars(2)(OF assign-nodes-props(2)) show False unfolding unnecessaryPhi-def by simp}
\textbf{qed}

\textbf{with \text{rs-props}}
\textbf{have aux: \text{set } \text{ms} \cap \text{set } (\text{ns}@\text{rs}) = \{ \} }
\textbf{by (mesis disjoint-iff-not-equal not-in-butlast old.path2-last)}

\textbf{have \phi_r-V: var g \phi_r = V}
\textbf{using V-vars(1) assign-nodes-props(8) phiArg-same-var by auto}
have $\forall_{-allVars} \varphi_r \in allVars \ g$

by (meson phiArg-def assign-nodes-props(8) allDefs-in-allVars old.path2-tl-in-on phiDefs-in-allDefs phi-phiDefs rs-props)

from $V$-props(2) $\varphi_r \cdot V$-s-allVars(2) $\varphi_r$-allVars r-s-path-props(3) r-s-path-props(1)

obtain $ss$ where $ss$-props: $g \vdash m -ms@ss \rightarrow$ defNode $g \varphi_s$ set (butlast (ms@ss))

$\cap$ set (butlast (ns@rs)) = {} by (rule phiArg-disjoint-paths-extend) (metis disjoint-iff-not-equal in-set-butlastD)

define $p_m$ where $p_m = ms@ss$

define $p_n$ where $p_n = ns@rs$

have ind-props: $g \vdash m -p_m \rightarrow$ defNode $g \varphi_s \ g \vdash n -p_n \rightarrow$ defNode $g \varphi_r$ set (butlast $p_m$) $\cap$ set (butlast $p_n$) = {} using $rs$-props(1) ss-props $p_m$-def $p_n$-def by auto

The following case will occur twice in the induction, with swapped identifiers, so we’re proving it outside. Basically, if the paths $p_m$ and $p_n$ intersect, the first such intersection point must be a $\phi$ function in reachable $g \varphi$, yielding the path convergence we seek.

have path-crossing-yields-convergence:

$\exists \varphi_z \in \text{reachable} \ g \varphi. \exists ns ms. old.pathsConverge g n ns m ms (\text{defNode} g \varphi_z)$

if $\varphi_r \in \text{reachable} g \varphi \ and \ \varphi_s \in \text{reachable} g \varphi \ and \ g \vdash n -p_n \rightarrow$ defNode $g \varphi_s$

$\ and \ g \vdash m -p_m \rightarrow$ defNode $g \varphi_s \ and \ set \ (\text{butlast} p_m) \cap \ set \ (\text{butlast} p_n) = {}$

and set $p_m \cap \ set p_n \neq {}$

for $\varphi_r \varphi_s p_m p_n$

proof –

from that(6) split-list-first-propE

obtain $p_m I n_z p_m 2$ where $n_z$-props: $n_z \in set p_n$ $p_m = p_m I \lnot n_z \neq p_m 2$

$\forall n \in set p_m I. \ n \notin set p_n$

by (auto intro: split-list-first-propE)

with that(3,4)

obtain $p_n'$ where $p_n'$-props: $g \vdash n -p_n' \rightarrow n_z g \vdash m -p_n I [n_z] \rightarrow n_z$ prefix

$p_n' p_n n_z \notin set (\text{butlast} p_n)$

by (meson old.path2-prefix-ex old.path2-split(1))

from $V$-props(3) reachable-same-var[OF that(1)] reachable-same-var[OF that(2)]

have phis-V: var $g \varphi_r = V \ var g \varphi_s = V$ by simp-all

from reachable-props(1) that(1,2) $\varphi$-props(2) phiArg-in-allVars

have phis-allVars: $\varphi_r \in allVars g \varphi_s \in allVars g$ by (metis rtranclp.cases)+

Various inequalities for proving paths aren’t trivial.

have $n \neq$ defNode $g \varphi_r m \neq$ defNode $g \varphi_r$
using $\varphi$-node-no-defs phis-V(1) phis-allVars(1) r-s-path-props(1,3) reachable-props(2)
that(1) by blast+

from $\varphi$-node-no-defs reachable-props(2) that(2) r-s-path-props(1,3) phis-V(2)
that phis-allVars
have $m \neq \text{defNode } g \varphi_s \ n \neq \text{defNode } g \varphi_s$ by blast+

With this scenario, since set (butlast $p_n$) $\cap$ set (butlast $p_m$) = $\emptyset$, one of the paths $p_n$ and $p_m$ must end somewhere within the other, however this means the $\phi$ function in that node must either be $\varphi$ or $\varphi_r$

from assms $n_z$-props
consider ($p_n$-ends-in-$p_m$) $n_z = \text{defNode } g \varphi_s \ | \ (p_m$-ends-in-$p_n$) $n_z = \text{defNode } g \varphi_r$

proof (cases $n_z =$ last $p_n$)
  case True
  with ($g \vdash n \rightarrow \text{defNode } g \varphi_r$)
  have $n_z = \text{defNode } g \varphi_r$, using old.path2-last by auto
  with that(2) show ?thesis.
next
  case False
  from $n_z$-props(2)
  have $n_z \in \text{set } p_m$ by simp
  with False $n_z$-props(1) (set (butlast $p_m$) $\cap$ set (butlast $p_n$) = $\emptyset$) : $g \vdash m
  -p_m \rightarrow \text{defNode } g \varphi_s$
  have $n_z = \text{defNode } g \varphi_s$, by (metis disjoint-elem not-in-butlast old.path2-last)
  with that(1) show ?thesis.
qed

thus $\exists \varphi_z \in \text{reachable } g \varphi. \exists ns ms. \text{old.pathsConverge } g n ns m ms \ (\text{defNode } g \varphi_z)$

proof (cases)
  case $p_n$-ends-in-$p_m$
  have old.pathsConverge $g n p_n \ m p_m$ (defNode $g \varphi_s$)
  proof (rule old.path2Converge1)
    from $p_n$-ends-in-$p_m$ p_n'-props(1) show $g \vdash n-p_n \rightarrow \text{defNode } g \varphi_s$ by simp
    from ($n \neq \text{defNode } g \varphi_s$) $p_n$-ends-in-$p_m$ p_n'-props(1) old.path2-nontriv
    show $I < \text{length } p_n \ m$ by auto
    from that(4) show $g \vdash m \rightarrow \text{defNode } g \varphi_s$.
    with ($m \neq \text{defNode } g \varphi_s$) old.path2-nontriv show $I < \text{length } p_m$ by simp
    from that $p_n$'-props(3) show set (butlast $p_n$) $\cap$ set (butlast $p_m$) = $\emptyset$
    by (meson butlast-prefix disjointI disjoint-elem in-prefix)
    qed
    with that(1,2,3) show ?thesis by (auto intro:reachable.intros(2))
next
  case $p_m$-ends-in-$p_n$
  have old.pathsConverge $g n p_n \ m (p_m \circ [n_z])$ (defNode $g \varphi_r$)
  proof (rule old.path2Converge1)
    from $p_m$-ends-in-$p_n$ p_n'-props(1,2) show $g \vdash n-p_n \rightarrow \text{defNode } g \varphi_r$ $g \vdash m-p_m \circ [n_z]\rightarrow \text{defNode } g \varphi_r$ by simp-all

20
with \( n \neq \text{defNode } g \varphi_r \) \( m \neq \text{defNode } g \varphi_r \) show 1 < length \( p_n' \) 1 < length \( \{p_m I \ @ \ [n_z]\} \)
using old.path2-nontriv[of \( g m p_m I \ @ \ [n_z]\) old.path2-nontriv[of \( g n\) by simp-all
from \( n_z\)-props \( p_n'\)-props(3) \) show set (butlast \( p_n'\)) \cap set (butlast (\( p_m I \ @ \ [n_z]\))) = {} \)
using butlast-snoc disjoint1 in-prefix in-set-butlastD by fastforce
qed
with that(1) show \(?thesis\) by (auto intro:reachable.intros)
qed
qed

Since the reachable-set was built starting at a single \( \varphi \), these paths must at some point converge within reachable \( g \varphi \).

from assign-nodes-props(3,2) ind-props V-props(3) \( \varphi_r\)-V \( \varphi_r\)-allVars
have \( \exists \varphi_z \in \text{reachable } g \varphi . \exists \text{ms. } \text{old.pathConverge } g \text{ n } \text{ ns m ms (defNode } g \varphi \)\)
proof (induction arbitrary: \( p_m \) \( p_n \) rule: reachable.induct)
case refl

In the induction basis, we know that \( \varphi = \varphi_s \), and a path to \( \varphi_r \) must be obtained – for this we need a second induction.

from refl.prems refl.hyps show \(?case\)
proof (induction arbitrary: \( p_m \) \( p_n \) rule: reachable.induct)
case refl

The first case, in which \( \varphi_r = \varphi_s = \varphi \), is trivial – \( \varphi \) suffices.

have old.pathConverge \( g \text{ n } p_n \) \( m \) \( p_m \) (defNode \( g \varphi \))
proof (rule old.pathConverge1)
show 1 < length \( p_n \) 1 < length \( p_m \)
using refl V-props simpleDefs-phiDefs-var-disjoint unfolding unnecessaryPhi-def
by (metis domD domI old.path2-hd-in-cancel old.path2-nontriv phi-phiDefs r-s-path-props(1) r-s-path-props(3)\)+
show \( g \vdash n-p_n \rightarrow \text{defNode } g \varphi g \varphi \vdash m-p_m \rightarrow \text{defNode } g \varphi\) set (butlast \( p_n \)) \cap set (butlast \( p_m \)) = {}
using refl by auto
qed
with \( \varphi \in \text{reachable } g \varphi \) show \(?case\) by auto
next
case (step \( \varphi' \) \( \varphi_r \))

In this case we have that \( \varphi = \varphi_s \) and need to acquire a path going to \( \varphi_r \), however with the aux. lemma we have, we still need that \( p_n \) and \( p_m \) are disjoint.

thus \(?case\)
proof (cases set \( p_n \) \cap set \( p_m \) = \{\})
case paths-cross: False
with step reachable.intros
show \(?thesis\) using path-crossing-yields-convergence[of \( \varphi_r \) \( \varphi p_n \) \( p_m \)] by (metis disjoint1 disjoint-elem)
next

21
case True

If the paths are intersection-free, we can apply our path extension lemma to obtain the path needed.

from step(9,8,10) \( \varphi \in \text{allVars } g \) r-s-path-props(1,3) step(6,5) True

step(2)

obtain ns where \( g \vdash n \rightarrow _{ns} \text{defNode } g \ \varphi' \) set (butlast \( p_n \)) \( \cap \)
set \( p_m = \{\} \) by (rule phiArg-disjoint-paths-extend)

from this(2) have set (butlast \( p_m ) \) \( \cap \) set (butlast \( p_n \)) = \{\}
using in-set-butlastD by fastforce
moreover
from phiArg-same-var step.hyps(2) step.prems(5) have var \( g \ \varphi' = V \)
by auto
moreover
have \( \varphi' \in \text{allVars } g \)
by (metis \( \varphi\)-props(2) phiArg-in-allVars reachable.cases step.hyps(1))
ultimately
show \( \exists \varphi \in \text{reachable } g \ \varphi \). \( \exists ns \). \( \text{old.ppathsConverge } g \ n \ ns \ m \ ms (\text{defNode } g \ \varphi) \)
using step.prems(1) \( \varphi\)-props V-props \( g \vdash n \rightarrow _{ns} \text{defNode } g \ \varphi' \)
by \(-\text{(rule step.II ; blast)}\)

qed

next

(\( \text{step } \varphi' \) \( \varphi_s \))

With the induction basis handled, we can finally move on to the induction proper.

show \(?\)thesis
proof (cases set \( p_m \) \( \cap \) set \( p_n = \{\} \))

case True

have \( \varphi_s \cdot V \) : var \( g \ \varphi_s = V \) using step(1,2,3,9) reachable-same-var by (simp add: phiArg-same-var)
from step(2) have \( \varphi_s \cdot \text{allVars} \) : \( \varphi_s \in \text{allVars } g \) by (rule phiArg-in-allVars)

obtain \( p_m' \) where \( \text{tmp } g \vdash m \rightarrow _{p_m'} \text{defNode } g \ \varphi' \) set (butlast \( p_m' \)) \( \cap \) set (butlast \( p_n ) \) = \{\}
by (rule phiArg-disjoint-paths-extend[of \( g \ \varphi_s \ V \ \varphi_r \ mn p_n p_m \ \varphi' \])
(metis \( \varphi_s \cdot V \) -allVars step r-s-path-props(1,3) True disjoint-iff-not-equal
in-set-butlastD)+

from step(5) this(1) step(7) this(2) step(9) step(10) step(11)
show \(?\)thesis by (rule step.II[of \( p_m \$ p_m' \) \( p_n)])

next

case paths-cross: False
with step reachable.intros

show \(?\)thesis using path-crossing-yields-convergence[of \( \varphi_r \ \varphi_s \ p_n \ p_m \) ] by blast

qed

22
then obtain $\varphi_z$ ns ms where $\varphi_z \in \text{reachable } g \varphi$ and old.pathsConverge $g$ ns m ms (defNode $g \varphi_z$)

by blast

moreover

with reachable-props have var $g \varphi_z = V$ by (metis V-props(3) phiArg-trancd-same-var rtranchpD)

ultimately have necessaryPhi' $g \varphi_z$ using r-s-path-props

unfolding necessaryPhi-def by blast

moreover with $(\varphi_z \in \text{reachable } g \varphi)$ have unnecessaryPhi $g \varphi_z$ by -(rule reachable-props)

ultimately show False unfolding unnecessaryPhi-def by blast

qed

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 $\neg (\exists P \text{ sec}. \text{redundant-sec } g P \text{ sec})$

shows cytronMinimal $g$

using assms 1 no-redundant-set-minimal by blast

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

corollary

assumes $\neg (\exists P. \text{redundant-set } g P)$

shows $\neg \text{redundant } g$

proof –

have redundant $g \implies \exists P. \text{redundant-set } g P$

proof –

assume redundant $g$

then obtain $\varphi$ where phi $g \varphi \neq \text{None trivial } g \varphi$

unfolding redundant-def redundant-set-def dom-def phiArg-def trivial-def isTrivialPhi-def

by (clarsimp split: option.splits) fastforce

hence redundant-set $g \{\varphi\}$

unfolding redundant-set-def dom-def phiArg-def trivial-def isTrivialPhi-def

by auto

thus ?thesis by auto

qed

with assms show ?thesis by auto

qed

end

end
References

