Minimal Static Single Assignment Form

Max Wagner       Denis Lohner

October 11, 2017

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 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 $g P \equiv \{ (\varphi, \varphi'), \phiArg g \varphi \varphi' \} \cap P \times P$

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 $g P \equiv P \not= \{} \land P \subseteq dom (\phi g) \land (\exists v' \in allVars g. \forall \varphi \in P. \forall \varphi'. \phiArg g \varphi \varphi' \rightarrow \varphi' \in P \cup \{v'\})$

**definition** redundant-scc $g P scc \equiv redundant-set g scc \land is-scc (induced-phi-graph g P) scc$

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

**definition** condensation-nodes $g P \equiv scc-of (induced-phi-graph g P) \cdot P$

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

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 $\phi$ nodes and their arguments (both of which are finite sets), is finite itself.

**proof**

- **let** $\phiEdges = \{(a,b), \phiArg g a b\}$
- **have** finite $\phiEdges$

**proof**

- **let** $\phiDomRan = (dom (\phi g) \times \bigcup \{set \cdot (ran (\phi g))\})$
- **from** phi-finite
- **have** finite $\phiDomRan$ by (simp add: imageE phi-finite map-dom-mn-finite)
- **have** $\phiEdges \subseteq \phiDomRan$
- **apply** (rule subst[of $\forall a \in \phiEdges. a \in \phiDomRan$])
- **apply** (simp—all add: subset-eq[symmetric] phiArg-def)
by (auto simp: mk-def)
with (finite ?phiDomRan)
show finite ?phiEdges by (rule Finite-Set.rev-finite-subset)
qed
hence ∩ : finite (f : {?phiEdges ∩ (P × 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) ⊆ (condensation-nodes g P × 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) ∈ (condensation-edges g P)
assumes (?ϕa ∈ a)
assumes (?ϕb ∈ b)
show (?ϕa, ?ϕb) ∈ (induced-phi-graph g P)^*
unfolding condensation-edges-def
proof
from asms(1)
obtain x y where x-y-props:
  (x, y) ∈ (induced-phi-graph g P)
a = scc-of (induced-phi-graph g P) x
b = scc-of (induced-phi-graph g P) y
unfolding condensation-edges-def by auto
hence x ∈ a y ∈ b by auto

All that's left is to combine these paths.
with asms(2) x-y-props(2)
have (?ϕa, x) ∈ (induced-phi-graph g P)^* by (meson is-sec-connected scc-of-is-sec)
moreover with asms(3) x-y-props(3) ⟨y ∈ b⟩
have (y, ?ϕb) ∈ (induced-phi-graph g P)^* by (meson is-sec-connected scc-of-is-sec)
ultimately
show (?ϕa, ?ϕb) ∈ (induced-phi-graph g P)^* using x-y-props(1) by auto
qed

lemma path-in-condensation-impl-path:
assumes (a, b) ∈ (condensation-edges g P)^+
assumes (?ϕa ∈ a)
assumes (?ϕb ∈ b)
show (?ϕa, ?ϕb) ∈ (induced-phi-graph g P)^*
using asms
proof (induction arbitrary: ?ϕb rule:trans-induct)
fix y z ?ϕb
assume (y, z) ∈ condensation-edges g P
hence is-sscc \((\text{induced-phi-graph } g P)\) y unfolding condensation-edges-def by auto
hence \(\exists \varphi_y. \varphi_y \in y\) using sec-non-empty' by auto
then obtain \(\varphi_y\) where \(\varphi_y\text{-in-}\ y: \varphi_y \in y\) by auto

assume \(\varphi_b\text{-elem}: \varphi_b \in z\)
assume \(\forall \varphi_b. \varphi_a \in a \implies \varphi_b \in y \implies (\varphi_a, \varphi_b) \in (\text{induced-phi-graph } g P)^+\)
with \(\text{assms}(2) \varphi_y\text{-in-}\ y\)
have \(\varphi_a\text{-to-}\varphi_y: (\varphi_a, \varphi_y) \in (\text{induced-phi-graph } g P)^+\) using condensation-edge-impl-path by auto

from \(\varphi_b\text{-elem } \varphi_y\text{-in-}\ y\ (y, z) \in \text{condensation-edges } g P\)
have \(\langle \varphi_y, \varphi_b \rangle \in (\text{induced-phi-graph } g P)^+\) using condensation-edge-impl-path by auto
with \(\varphi_a\text{-to-}\varphi_y\)
show \(\langle \varphi_a, \varphi_b \rangle \in (\text{induced-phi-graph } g P)^+\) by auto
qed (auto intro: condensation-edge-impl-path)

\textbf{lemma} condensation-acyclic: acyclic (condensation-edges \(g P\))
\textbf{proof} (rule acyclicI, rule allI, rule contr, simp)
fix \(x\)
Assume there is a cycle in the condensation graph.

assume cyclic: \((x, x) \in (\text{condensation-edges } g P)^+\)
have nonrel: \((x, x) \notin (\text{condensation-edges } g P)^+\) unfolding condensation-edges-def by auto

Then there must be a second SCC \(b\) on this path.
from this cyclic
obtain \(b\) where \(b\text{-on-}\ y:\ (x, b) \in (\text{condensation-edges } g P)\ (b, x) \in (\text{condensation-edges } g P)^+\)
by (meson converse-transclE)

hence \(x \in (\text{condensation-nodes } g P)\ b \in (\text{condensation-nodes } g P)\) using condensation-nodes-edges by auto
hence nodes-are-sscc: is-sscc \((\text{induced-phi-graph } g P)\) \(x\) is-sscc \((\text{induced-phi-graph } g P)\) \(b\)
using sec-of-is-sscc unfolding induced-phi-graph-def condensation-nodes-def by auto

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

have \(\exists \varphi_x. \varphi_x \in x\) \(\exists \varphi_b. \varphi_b \in b\) using nodes-are-sscc sec-non-empty' ex-in-conw by auto
then obtain \(\varphi_x\ \varphi_b\) where \(\varphi_b\text{-elem}: \varphi_x \in x \ \varphi_b \in b\) by metis
with nodes-are-sscc(1) b-on-path path-in-condensation-impl-path condensation-edge-impl-path
\(\varphi_b\text{-elem}(2)\)
have \(\varphi_b \in x\)

4
This however means \( x \) and \( b \) must be the same SCC, which is a contradiction to the nonreexivity of \textit{condensation-edges}.

\begin{verbatim}
by \( - \) (rule is-sec-closed)

with nodes-are-sec ϕb-elem
have \( x = b \) using is-sec-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} \textit{Ex-condensation-leaf}:
\textit{assumes} \( P \neq \{\} \)
\textit{shows} \( \exists \) leaf. \( \text{leaf} \in \text{(condensation-nodes g P)} \land (\forall \text{sec.} (\text{leaf}, \text{sec}) \notin \text{condensation-edges g P}) \)
\textit{proof} –
from \textit{assms} obtain \( x \) where \( x \in \text{condensation-nodes g P} \) unfolding \textit{condensation-nodes-def}
by auto
show \textit{thesis}
proof (rule \textit{wfE-min})
from \textit{condensation-finite} \textit{condensation-acyclic}
show \( \text{wf} ((\text{condensation-edges g P})^{-1}) \) by (rule \textit{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 \text{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}) \text{ by blast}
qed fact
\end{verbatim}

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

with different

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

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

from \text{step}

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

qed simp

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

apply –

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

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

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

qed

lemma \text{redundant-sec-phis}:

assumes \(\text{redundant-set } g P \text{ sec } \in \text{condensation-nodes } g P \text{ } x \in \text{sec}\)

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

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

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.

lemma 1:

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

shows \(\exists \text{sec } \subseteq P. \text{ redundant-sec } g P \text{ sec}\)

proof –

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

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

unfolding \(\text{redundant-set-def by auto}\)

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

moreover

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

moreover

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

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

using \(\text{redundant-sec-phis leaf-props(1) assms(1) by auto}\)

moreover

from \text{assms}

obtain \text{pred} where \text{pred-props}: \text{pred } \in \text{allVars} g \forall \phi \in P. \forall \phi'. \phi \text{Any } g \phi \phi' \rightarrow \phi' \in P \cup \{\text{pred}\} \text{ unfolding } \text{redundant-set-def by auto}\)

{6}
Any argument of a $\phi$ function in the leaf SCC which is not in the leaf SCC itself must be the unique argument of $P$

fix $\varphi, \varphi'$

consider $(\text{in-P}) \varphi' \notin \text{leaf} \land \varphi' \in P \mid (\text{neither}) \varphi' \notin \text{leaf} \land \varphi' \notin P \cup \{\text{pred}\} \mid \varphi' \notin \text{leaf} \land \varphi' \in \{\text{pred}\} \mid \varphi' \in \text{leaf} \text{ by auto}$

hence $\varphi' \in \text{leaf} \cup \{\text{pred}\}$ if $\varphi \in \text{leaf}$ and $\text{phiArg } g \varphi \varphi'$

proof ases

case in-P — In this case leaf wasn’t really a leaf, a contradiction

moreover

from in-P that leaf-prps(1) sec-in-P[of leaf g P]

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

ultimately

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

unfolding condensation-edges-def

using leaf-prps(1) that 'is-sec (induced-phi-graph g P) leaf':

apply -

apply clarsimp

apply (rule conjI)

prefer 2

apply auto[1]

unfolding condensation-nodes-def

by (meson (no-types, lifting) is-sec-unique node-in-sec-of-node pair-imageI sec-of-is-sec)

with leaf-prps(2)

show thesis by auto

next

case neither — In which case P itself wasn’t redundant, a contradiction

with that leaf-prps pred-prps

have ¬redundant-set g P unfolding redundant-set-def

by (meson rev-subsetD sec-in-P)

with assms

show thesis by auto

qed auto — the other cases are trivial

} with pred-prps(1)

have $\exists \nu' \in \text{allVars } g, \forall \varphi \in \text{leaf}, \forall \varphi', \text{phiArg } g \varphi \varphi' \rightarrow \varphi' \in \text{leaf} \cup \{\nu'\}$ by auto

ultimately

have redundant-sec g P leaf unfolding redundant-sec-def redundant-set-def by auto

thus thesis using leaf-prps(1) sec-in-P by meson

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.
inductive-set reachable :: 'g ⇒ 'val ⇒ 'val set
  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'
  using assms
  by (induction φ' rule: reachable.induct) auto

  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 (rule contr)
  assume infinite (trueArgs g φ)
  hence a: infinite \{ φ'. ∃ φ'' ∈ reachable g φ. phiArg g φ'' φ' \} by auto
  have phiArg-set: \{ φ'. ∃ φ. phiArg g φ φ' \} = \bigcup (set {b. ∃ a. phi g a = Some b}) unfolding phiArg-set by auto

  If the true arguments of a φ function are infinite in number, there must be an infinite number of φ functions...

  have infinite \{ φ'. ∃ φ. phiArg g φ φ' \}
    by (rule infinite-super[of \{ φ'. ∃ φ'' ∈ reachable g φ. phiArg g φ'' φ' \}]) (auto simp: a)
    with phiArg-set
  have infinite (min (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 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)
  unfolding redundant-set-def
proof (intro conjI)
  from assms
  show reachable g φ ≠ {} 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} \text{cases} \text{subsetI} \text{unnecessary} \text{Phi-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 \ \text{by (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 \( ?\text{thesis} \) by auto

next
  case empty
  from \text{allDefs-in-allVars[of-Node g \varphi]} \text{assms}
  have \( \phi\text{-var}: \ \varphi \in \text{allVars} \ g \text{ unfolding unnecessaryPhi-def phiDefs-def allDefs-def defNode-def phi-def trueArgs-def} \)
  by (clarsimp simp: \text{domIphis-in-\alpha n})
  from \text{empty \text{assms}(1)}
  have \( \text{no-preds: trueArgs } g \varphi = \{\} \) by (subst \text{card-eq}[\text{OF preds-finite, symmetric}]) auto
  show \( ?\text{thesis} \)
  proof (rule \text{rule bexI}, \text{rule ballI}, \text{rule allI}, \text{rule impI})
    fix \( \varphi', \varphi'' \)
    assume \( \phi\text{-props}: \ \varphi' \in \text{reachable } g \varphi. \ \phi\text{Arg } g \varphi' \varphi'' \)
    with \text{no-preds}
    have \( \varphi'' \in \text{reachable } g \varphi \) by auto
  unfolding \( \text{trueArgs-def} \)
  proof
    from \phi\text{-props}
    have \( \varphi'' \in \{\varphi'. \ \exists \varphi'' \in \text{reachable } g \varphi. \ \phi\text{Arg } g \varphi' \varphi''\} \) by auto
    with \phi\text{-props \text{no-preds}}
    show \( \varphi'' \in \text{reachable } g \varphi \) unfolding \( \text{trueArgs-def} \) by auto
  qed
  thus \( \varphi'' \in \text{reachable } g \varphi \cup \{\} \) by simp
  qed (auto simp: \phi-var)
  qed

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

using assms

apply (induction rule: tranclp-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 \( \var \var \phi \) argument to the corresponding \( \var \var \phi \) function while preserving disjointness to a second path.

lemma phiArg-disjoint-paths-extend:

assumes \( \var g r = V \) and \( \var g s = V \) and \( r \in \text{allVars } g \) and \( s \in \text{allVars } g \)

and \( V \in \text{oldDefs } g n \) and \( V \in \text{oldDefs } g m \)

and \( g \vdash n \rightarrow \text{defNode } g r \) and \( g \vdash m \rightarrow \text{defNode } g s \)

and \( \text{set } ns \cap \text{set } ms = \{\} \)

and \( \phi r g = \var \phi r \) obtains \( ns' \)

where \( g \vdash n \rightarrow \text{defNode } g r \)

and \( \text{set } (\text{butlast } (\text{ns} @ ns')) \cap \text{set } ms = \{\} \)

proof (cases \( r = \var \phi r \))

  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 \( g \vdash n \rightarrow \text{defNode } g r \) set (butlast (\text{ns} @ ns')) \cap \text{set } ms = \{\}

  by simp-all fastforce

  with that show \( ?\text{thesis} \).

  next

  case False

  It suffices to obtain any path from \( r \) to \( \var \phi r \). However, since we’ll need the corresponding predecessor of \( \var \phi r \) later, we must do this as follows:

  from assms(10)

  have \( \var \phi r \in \text{allVars } g \) unfolding phiArg-def

  by (metis allDefs-in-allVars phiDefs-in-allDefs phi-def phi-phiDefs phis-in-\( \alpha \))

  with assms(10)

  obtain \( rs' \vdash \text{pred } r \) where \( rs'-props: g \vdash \text{defNode } g r \rightarrow rs' \rightarrow \text{pred } \var \phi r \) old.EntryPath

  g \( rs' \) \( r \in \text{phiUses } g \) pred_\( \var \phi r \) pred_\( \var \phi r \) \( \in \text{set } (\text{old.precdecessors } g ) \text{ (defNode } g \var \phi r ) \)

  by (rule phiArg-path-ex')

  def \( rs \equiv rs'@[\text{defNode } g \var \phi r] \)

  from \( rs'-props(2,1) \) old.EntryPath-distinct old.path2-hd

  have \( rs' \text{-loop-free: defNode } g r \notin \text{set } (\text{tl } rs') \) by (simp add: Misc.distinct-hd-\( \text{tl} \))

  from False assms have defNode \( g \var \phi r \neq \text{defNode } g r \)

  apply

  apply (rule phiArg-distinct-nodes)

  apply (auto intro:phiArg-in-allVars)[2]
unfolding phi-phiDefs by (metis allDefs-in-allVars phiDefs-in-allDefs phi-def phi-phiDefs phi-in-cc)

from rs'-props
have rs'props: g ⊢ defer g rs → defer g rs' length rs > 1 defer g rs' r /∈ set tl rs
  apply (subgoal-tac defer g rs = hd rs')
  prefer 2 using rs'props(I)
  apply (rule old.path2-hd)
  using old.path2-snoc old.path2-def rs'props(1) rs-def rs'loopfree (defer g rs' r ≠ defer g rs)
  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 ∃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-rs) x ∈ set ns | (in-rs) x ∈ set (butlast (tl rs)) by (metis Un-iff butlast-append in-set-butlastD set-append)
      thus False
      apply (asises)
      using x'props(2) assms(9)
      apply (simp add: disjoint-elem)
      by (metis x'props(2) inter-empty in-set-HD List.butlast-tl 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 ⊢ defer g ri → i ∈ set ms ∀n ∈ set (butlast ri). n /∈ set ms prefix ri rs
    apply –
    apply (rule old.path2-split-first-prop[of g defer g ri rs defer g ri 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

11
obtain \( m's \) where \( m's\)-props: \( g \vdash m - m's \rightarrow i \) prefix \( m's \) \( m \notin \text{set} \) (butlast \( m's \)) 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 \( (ri\text{-is-valid}) i = \text{defNode} g \varphi_r \mid (m\text{-same}) i \neq \text{defNode} g \varphi_r \) \( m = i \)

| (m\text{-differ}) \( i \neq \text{defNode} g \varphi_r \) \( m \neq i \) by auto

thus thesis

proof (cases)
  case \( ri\text{-is-valid} \)
  \( ri \) is a valid path extension.
  with \( \text{assms}(7) \) \( ri\text{-props}(1) \)
  have \( g \vdash n - n\circ tl ri \rightarrow \text{defNode} g \varphi_r \) by auto

moreover
  have \( \text{set} \) (butlast \( n\circ tl ri \)) \( \cap \) \( \text{set} m s = \{ \} \)
  proof (rule contr)
    assume \( \text{contr} : \text{set} \) (butlast \( n\circ tl ri \)) \( \cap \) \( \text{set} m s \neq \{ \} \)
    from this
    obtain \( x \) where \( x\text{-props} : x \in \text{set} \) (butlast \( n\circ tl ri \)) \( x \in \text{set} m s \) by auto
    with \( \text{assms}(9) \) have \( x \notin \text{set} n s \) by auto
    with \( x\text{-props} (g \vdash n - n\circ tl ri \rightarrow \text{defNode} g \varphi_r) \) \( \langle \text{defNode} g \varphi_r \neq \text{defNode} g \rangle \)
    with \( \text{assms}(7) \)
    have \( x \in \text{set} \) (butlast \( n\circ tl ri \))
    by (metis Un-iff append-Nil2 butlast-append old.path2-last set-append)
    with \( x\text{-props}(2) \) \( ri\text{-props}(3) \)
    show False by (metis FormalSSA-Misc.in-set-TD List.butlast-tl)
  qed

ultimately
  show thesis by (rule that)

next
  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 \( rs'\text{-props}(1) \) \( rs'\text{-loop-free} \)

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

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

by simp

moreover
from `rs-pr ops(1)` \(m\)-i-same `rs-def ri-pr ops(1, 2, A)` `defNode g \varphi_r \neq defNode g r` \(\langle\text{defNode g}\rangle\)
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 allDefs g m var g \varphi_m = V`
from `rs-pr ops(1)` \(m\)-i-difer

ultimately
have `\varphi_g \varphi_m \neq \varphi_g r` by (rule conventional, simp-all)
with `\langle \varphi_g \varphi_m = V \rangle` \(\text{assms(1)}\)
have `False` by simp
thus `?thesis` by simp

next
case `m-i-difer`
If \(m \neq i\), \(i\) constitutes a proper path convergence point.
have `old.path2Converge g m ms' n (ns @ tl ri) i`
proof (rule `old.path2ConvergeI`)
show `1 < length ms'` using `m-i-difer ms'-props old.path2-nontriv` by blast
next
show `1 < length (ns @ tl ri)`
using `ri-pr ops old.path2-nontriv` \(\text{assms(9)}\)
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 `contra`)
assume `set (butlast ms') \cap set (butlast (ns @ tl ri)) \neq \{\}`
then obtain `i'` where `i'-props: i' \in set (butlast ms') i' \in set (butlast (ns @ tl ri))`
by auto
with `ms'-props(2)`
have `i'-not-in-ms: i' \in set (butlast ms)` by (metis `in-set-butlast-appI` `prefixE`)
with `assms(9)`
show `False`
proof (cases `i' \notin set ns`)
case `True`
with `i'-props(2)`
have `i' \in set (butlast (tl ri))`
by (metis `Un-iff buttlast-append in-set-buttlastD set-append`)
hence `i' \in set (butlast ri)` by (simp add: `in-set-tlD List.buttlast-tl`)
with `i'-not-in-ms ri-pr ops(3)`
show `False` by (auto dest: `in-set-buttlastD`)
qed (meson disjoint-elem `in-set-buttlastD`)
qed
qed (auto intro: `assms(7)` `ri-pr ops(1)` `old.path2-app ms'-props(1)`)

13
At this intersection of paths we can find a $\phi$ function.

from this assms(6,5)
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...

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

moreover
from 'defNode $g \; r$ $\notin$ set (tl rs')': rs'-rest-def
have 'defNode $g \; r$ $\notin$ set rs'-rest by auto
with $g \vdash i - i#rs'-rest \rightarrow pred_{\phi_{\sigma}}$
have $\forall x. \; x \in set rs'-rest \implies r \notin \text{allDefs } g \; x$
by (metis defNode-eq list.distinct(1) list.sel(3) list.set-cases old.path2-cases
old.path2-in-cn)
moreover
from assms(7,9) ($g \vdash i - i#rs'-rest \rightarrow pred_{\phi_{\sigma}}$: ri-props(2)
have $r \notin \text{defs } g \; i$
by (metis defNode-eq defs-in-allDefs disjoint-elem old.path2-hd-in-cn old.path2-last-in-ns)
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 $r$.

have $\exists y. \; \text{phis } g \; (i, r) = \text{Some } y$
by (rule convergence-prop [where $g=g$ and $n=i$ and $v=r$ and $ns=i#rs'-rest$, simplified])
moreover
from $g \vdash n-ns \rightarrow \text{defNode } g \; r$: have 'defNode $g \; r$ $\in$ set ns by auto

14
with \( \{ i \in \text{set } ms \} \) have \( i \neq \text{defNode } g \) by auto

 moreover

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

 moreover

 have \( \text{defNode } g \) \( r \in \text{set } (\alpha n \ g) \) 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 False

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

 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 \?thesis by simp

 qed

 qed

 lemma \text{reachable-same-var}:

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

 shows \( \varphi \ \varphi = \varphi \ \varphi' \)

 using assms by (metis Nitpick.\text{rtranclp-unfold} \text{phi-Arg-bunch-same-var} \text{reachable-props}(1))

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

 assumes unnecessary\( \text{Phi } g \ \varphi \ \varphi \in \text{allVars } g \ \varphi \ \varphi \in \text{oldDefs } g \ n \)

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

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

 unfolding unnecessary\( \text{Phi-def} \) by auto

 lemma \text{defNode-differ-aux}:

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

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

 proof (rule contr)

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

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

 from assms(1)

 have vars-eq: \( \varphi \ \varphi = \varphi \ \varphi_s \)

 apply

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

 apply simp

 apply (rule \text{phi-Arg-bunch-same-var})

 apply (drule \text{reachable-props})

 unfolding \text{reachable-def} by (meson \text{IntD1} \text{mem-Collect-eq} \text{rtranclpD})
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: \( \neg(\exists P. \text{redundant-set } g P) \)

**shows** cytronMinimal \( g \)

**proof** (rule contr)

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

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

then obtain \( \phi \) where \( \phi\text{-props: unnecessaryPhi } g \phi \in \text{allVars } g \phi \in \text{reachable} \)

using cytronMinimal-def unnecessaryPhi-def reachable-def unnecessaryPhi-def reachable-infos by auto

We consider the reachable-set of \( \phi \). If \( \phi \) 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 \( \phi \).

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

thus False

**proof** cases

**case trivial**

If there are less than 2 true arguments of this set, the set is trivially redundant (see few-preds-redundant).

from this \( \phi\text{-props}(1) \)

have redundant-set \( g \) (reachable \( g \phi \)) by (rule few-preds-redundant)

with no-redundant-set

show False by simp
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:

- $r \neq s \ \varphi_r \in \text{reachable \ g} \ \varphi_s \in \text{reachable \ g} \ \varphi$
- $\neg \text{unnecessaryPhi} g \ r \ \neg \text{unnecessaryPhi} g \ s$
- $r \in \{n. \ (\text{phiArg} g)\^{++} \ \varphi \ n\} \ s \in \{n. \ (\text{phiArg} g)\^{++} \ \varphi \ n\}$
- $\text{phiArg} g \ \varphi_r \ r \ \text{phiArg} g \ \varphi_s \ s$

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-rtr anclp-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: $\var g \ r = V \ \var g \ s = V \ \var g \ \varphi = V$ by (metis phiArg-trancl-same-var)

moreover from r-s-in-tranclp
have r-s-allVars: $r \in \text{allVars} \ g \ s \in \text{allVars} \ g \ s$ 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: defNode $g \ r \neq \text{defNode} \ g \ s$ by auto

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

set ns $\cap$ set ms $=$ $\{\}$ by (auto intro: unnecessaryPhis-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-prefixes(2) old.path2-hd-in-ns
have $n \in \text{set ns}$ by blast
moreover from n-m r-s-path-prefixes(4) old.path2-hd-in-ns
have $n \in \text{set ms}$ by blast
ultimately
show False using r-s-path-prefixes(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-prefixes assign-nodes-props
obtain rs where rs-props: g ⊢ n − ϕr @ set (butlast (ns @ rs))
\[ \Delta set ms = \{ \} \]
using phiArg-disjoint-paths-extend by blast
(In fact, we can prove that set (ns @ rs) \( \cap \) set ms = \{\}, which we need for the next path extension.)

have defNode g \( \phi_r \) ∈ set ms
proof (rule ccontr)
assume \( \phi_r \)-in-ms: ¬ defNode g \( \phi_r \) ∈ set ms
from this r-s-path-props(4)
obtain ms’ where ms’-props: g ⊢ m − ϕr prefix ms’ ms by
-(rule old.path2-prefix-ex [of g m ms defNode g \( \phi_r \)], auto)

have old.pathsConverge g n (ns @ rs) m ms’ (defNode g \( \phi_r \))
proof (rule old.pathsConvergeI)
show set (butlast (ns @ rs)) \( \cap \) set (butlast ms’) = \{\}
proof (rule ccontr)
assume set (butlast (ns @ rs)) \( \cap \) set (butlast ms’) \( \neq \) \{\}
then obtain c where c-props: c ∈ set (butlast (ns @ rs)) c ∈ set (butlast ms’) by auto
from this(2) ms’-props(2)
have c ∈ set ms by (simp add: in-prefix in-set-butlastD)
with c-props(1) rs-props(2)
show False by auto
qed

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

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

have \( \phi_r \)-V: var g \( \phi_r \) = V
using V-props(1) assign-nodes-props(8) phiArg-same-var by auto
have \( \varphi_r \in \text{allVars} \) 
by (meson phiArg-def assign-nodes-props(8) allDefs-in-allVars old.path2-\text{tl-in-con phi-phiDefs in-alphaDefs phi-phiDefs rs-props})

from V-props(2) \( \varphi_r \in \text{allVars} \) rs-path-props(3) rs-path-props(1) 
rs-path-props(4) rs-props(1) aux assign-nodes-props(9)

obtain ss where ss-props: \( g \vdash m - ms@ss \rightarrow \text{defNode} g \varphi_s \) set (butlast \((ms@ss)\)) 
\( \cap \text{set} \) (butlast \((ns@rs)\)) = \{
by (rule phiArg-disjoint-paths-extend) (metis disjoint-iff-not-equal in-set-butlastD)

def \( p_m \equiv ms@ss \) 
def \( p_n \equiv ns@rs \)

have \( \text{ind-props}: g \vdash m - p_m \rightarrow \text{defNode} g \varphi_s g \vdash n - p_n \rightarrow \text{defNode} g \varphi_r \) set (butlast \( p_m \)) \( \cap \text{set} \) (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 \( \text{path-crossing-yields-convergence}: \)
\( \exists \varphi_z \in \text{reachable} g \varphi. \exists ns ms. \text{old.pathConverge} g n ns m ms (\text{defNode} g \varphi_z) \)
if \( \varphi_r \in \text{reachable} g \varphi \) and \( \varphi_s \in \text{reachable} g \varphi \) and \( g \vdash n - p_n \rightarrow \text{defNode} g \varphi_r \)
and \( g \vdash m - p_m \rightarrow \text{defNode} g \varphi_s \) and set (butlast \( p_m \)) \( \cap \text{set} \) (butlast \( p_n \)) = \{

and set \( p_m \cap \text{set} p_n \neq \) \}
for \( \varphi_r \varphi_s p_m p_n \)

proof –
from that(6) split-list-first-propE
obtain \( p_m l n_z p_m 2 \) where \( n_z \)-props: \( n_z \in \text{set} p_n p_m = p_m l \odot n_z \# p_m 2 \)
\( \forall n \in \text{set} p_m l. n \notin \text{set} p_n \)
by (auto intro: split-list-first-propE)

with that(3,4)
obtain \( p_n \) where \( p_n \)-props: \( g \vdash n - p_n \rightarrow n_z g \vdash m - p_m l[n_z] \rightarrow n_z \) prefix \( p_n p_n n_z \notin \text{set} \) (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 \( \text{phis-V}: \text{var} g \varphi_r = V \text{var} g \varphi_s = V \) by simp-all
from reachable-props(1) that(1,2) \( \varphi \)-props(2) phiArg-in-allVars
have \( \text{phis-allVars}: \varphi_r \in \text{allVars} g \varphi_s \in \text{allVars} g \) by (metis rtranclp.cases)+

Various inequalities for proving paths aren’t trivial.

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

from \( \varphi\text{-node-no-defs} \) 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 \( p_m \)) = \{\}, 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 \( n_z\text{-props} \)
consider (\( p_n\text{-ends-in-}p_m \)) \( n_z = \text{defNode}\ g \varphi_s \) | (\( p_m\text{-ends-in-}p_n \)) \( n_z = \text{defNode}\ g \varphi_r \)

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

thus \( \exists \varphi_z \in \text{reachable}\ g \varphi \). \( \exists n\text{ ms. old.path\text{\-}Converge}\ g n \text{ ms}\ ms\text{ (}\text{defNode}\ g \varphi_z\text{)} \)

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

next
  case \( p_m\text{-ends-in-}p_n \)
  have old.path\text{\-}Converge g n \( p_n \text{'}\ m \) \( \text{defNode}\ g \varphi_r \)
  proof (rule old.path\text{\-}Converge \( I \))
    from \( p_m\text{-ends-in-}p_n \) \( p_n \text{'}\text{-props}(1,2) \) show \( g \vdash n - p_n \text{'} \rightarrow \text{defNode}\ g \varphi_r \) \( g \vdash m - p_m \text{ } \&\text{ } \left[ n_z = \text{defNode}\ g \varphi_r \right] \) by simp-all

20
with \( n \neq \text{defNode} g \varphi_r \) (\( m \neq \text{defNode} g \varphi_r \)) show \( 1 < \text{length} p_n' \ 1 < \text{length} (p_m @ [n_z]) \) using old.path2-nontriv[of g m p_m @ [n_z]] old.path2-nontriv[of g n] by simp-all
from \( n_z \cdot \text{props} p_n' \cdot \text{props}(3) \) show \( \text{set} (\text{butlast} p_n') \cap \text{set} (\text{butlast} (p_m @ [n_z])) = \{\} \) using butlast-snoc disjointI in-prefix in-set-butlastI by fastforce
qed

with that(1) show \(?\)thesis by (auto intro:reachable.intro)
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) \( \varphi_r \cdot V \varphi_r \cdot \text{allVars} \) have \( \exists \varphi_z \in \text{reachable} g \varphi. \exists n s m s. \text{old.pathsConverge} g n n s m m s (\text{defNode} g \varphi_z) \)

proof (induction arbitrary: \( p_m \ p_n \ \text{rule: reachable.induct} \))
case refl

In the induction basis, we know that \( \varphi = \varphi_z \) 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: \( p_m \ p_n \ \text{rule: reachable.induct} \))
case refl
The first case, in which \( \varphi_r = \varphi_z = \varphi \), is trivial — \( \varphi \) suffices.

have old.pathsConverge g n p_n m p_m (defNode g \varphi)
proof (rule old.pathsConvergeI)
  show \( 1 < \text{length} p_n \ 1 < \text{length} p_m \)
  using refl V-props simpleDefs-phiDefs-var-disjoint unfolding unnecessaryPhi-def by (metis domD domIff old.path2-hd-in-on old.path2-nontriv phi-phiDefs r-s-path-props(1) r-s-path-props(3))
  show \( g \vdash n \cdot p_n \cdot \text{defNode} g \varphi \) \( g \vdash m \cdot p_m \cdot \text{defNode} g \varphi \) \( \text{set} (\text{butlast} p_n) \cap \text{set} (\text{butlast} p_m) = \{\} \)
  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 \( \varphi = \varphi_z \) 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 disjointI disjoint-elem)
next

next
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) \ (\var \in \text{allVars } g) \ r-s-path-props(1,3) \ step(6,5) \ True

step(2)  

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

from this(2) have set (butlast p_m) \cap set (p_n \at \ ns) = \{\} 
using in-set-butlastD by fastforce 

moreover from phiArg-same-var step.hyps(2) step.prems(5) have var g \var' = V 
by auto 

moreover have \var' \in \text{allVars } g 
by (metis \ \var' \in \text{allVars } g \ \var') 
ultimately show \exists \var_z \in \text{reachable } g. \ \exists \text{ms. old.pathsConverge } g \text{ ns m ms } (\text{defNode } g \ \var_z) 
using step.prems(1) \ \var' \text{-props } V-props \ \g \vdash n \rightarrow \text{defNode } g \ \var'
by - (rule step.IH ; blast)

qed 
next


case (step \var' \var_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 \var_s : V. \ \var_s = V 
using step(1,2,3,9) \ reachable-same-var 
by (simp add: phiArg-same-var)

from step(2) have \var_s : allVars. \ \var_s \in \text{allVars } g 
by (rule phiArg-in-allVars)

obtain \ p_m' \ \text{ where \ tmp: } \ g \vdash m \rightarrow \text{defNode } g \ \var' \ \text{set (butlast \ (p_m \at \ p_m'))} \cap \set (butlast p_n) = \{\} 
by (rule phiArg-disjoint-paths-extend [of \ g \ \var_s V \ \var_r \ m \ n \ p_m \ p_n \ \var']) 
(metis \ \var_s : V \ \var_s : allVars \ step r-s-path-props(1,3) True disjoint-if-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.IH [of \ p_m \at \ p_m' \ p_n])

next

case paths-cross: False 

with step reachable.intros 

show \ ?thesis 
using path-crossing-yields-convergence [of \ \var_r \ \var_s \ p_n \ p_m] 
by blast

qed
then obtain $\varphi_z$ ns ms where $\varphi_z \in \text{reachable } g \varphi$ and old.pathsConverge $g$ n
ns m ms (defNode $g \varphi_z$)
by blast
moreover
with reachable-props have var $g \varphi_z = V$ by (metis V-prefixes (3) phiArg-tranc-same-var
rtrancpD)
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 $\neg (\exists P \text{ sec. redundant-sec } g P \text{ sec})$
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 $g \Longrightarrow \exists P . \text{ redundant-set } g P$
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

