Minimal Static Single Assignment Form

Max Wagner  Denis Lohner

February 23, 2021

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 g P ≡ \{(ϕ,ϕ'), \phiArg g ϕ ϕ'\} \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 ≡ P \neq {} \land P \subseteq \text{dom (phi g)} \land (\exists v' \in \text{allVars g. \forall ϕ \in P. \forall ϕ'. \phiArg g ϕ ϕ' \rightarrow ϕ' \in P \cup \{v'\}})
definition redundant-scc g P scc ≡ redundant-set g scc \land \text{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 ≡ \text{scc-of (induced-phi-graph g P) \cup P}
definition condensation-edges g P ≡ ((\lambda(x,y). (\text{scc-of (induced-phi-graph g P) x, scc-of (induced-phi-graph g P) y})) \cup (\text{induced-phi-graph g P})) \setminus \text{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=(\text{dom (phi g)} \cup (\text{set \cup (ran (phi g)))))
  from phi-finite
  have finite ?phiDomRan by (simp add: imageE phi-finite map-dom-ran-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: ran-def)
with (finite ?phiDomRan)
show finite ?phiEdges by (rule Finite-Set.rev-finite-subset)
qed

hence \( \bigwedge f. \) finite \( (f \setminus (\text{phiEdges} \cap (P \times P))) \) 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: \( (\text{condensation-edges g P}) \subseteq (\text{condensation-nodes } g \times \text{condensation-nodes } g) \)

unfolding condensation-edges-def condensation-nodes-def induced-phi-graph-def
by auto

**Lemma** condensation-edge-impl-path: takes assumptions \((a, b) \in (\text{condensation-edges g P})\)
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:
\((x,y) \in (\text{induced-phi-graph g P})\)
\(a = \text{scc-of} (\text{induced-phi-graph g P}) x\)
\(b = \text{scc-of} (\text{induced-phi-graph g P}) y\)
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-scc-connected scc-of-is-scc)
moreover with assms(3) x-y-props(3) \(y \in b\)
have \((y, \varphi_b) \in (\text{induced-phi-graph g P})^*\) by (meson is-scc-connected scc-of-is-scc)
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:
takes assumptions \((a, b) \in (\text{condensation-edges g P})^+\)
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:trancl-induct)
fix \(y z \varphi_b\)
assume \((y,z) \in \text{condensation-edges g P}\)
hence \( \text{is-scc} \) (\text{induced-phi-graph} \ g \ P) \ y \ \text{unfolding} \ \text{condensation-edges-def} \ \text{by auto}

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

assume \( \varphi_b \cdot \varphi_b \in z \)
assume \( \land \varphi_b. \varphi_a \in a \Longrightarrow \varphi_b \in y \Longrightarrow (\varphi_a, \varphi_b) \in (\text{induced-phi-graph} \ g \ P)^* \)
with \( \text{assms(2)} \) \( \varphi_y \cdot \in - y \)
have \( \varphi_a \cdot \varphi_y: (\varphi_a, \varphi_y) \in (\text{induced-phi-graph} \ g \ P)^* \) \text{using condensation-edge-impl-path by auto}

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

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

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

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

from this cyclic
obtain \( b \) \text{where} \( b \cdot \text{on-path}: (x, b) \in (\text{condensation-edges} \ g \ P) \) \( (b, x) \in (\text{condensation-edges} \ g \ P)^+ \)
by (meson converse-tranclE)

hence \( x \in (\text{condensation-nodes} \ g \ P) \) \( b \in (\text{condensation-nodes} \ g \ P) \) \text{using condensation-nodes-edges by auto}

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

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

have \( \exists \varphi_x. \varphi_x \in x \) \( \exists \varphi_b. \varphi_b \in b \) \text{using nodes-are-scc scc-non-empty' ex-in-conv by auto}
then obtain \( \varphi_x \) \( \varphi_b \) \text{where} \( \varphi_x \cdot \varphi_b \in x \) \( \varphi_b \in b \) \text{by metis with nodes-are-scc(1) b-on-path path-in-condensation-impl-path condensation-edge-impl-path}
(\( \varphi_x \cdot \varphi_b \cdot \text{elem}(2) \))
have \( \varphi_b \in x \)
by \((-\text{rule is-scc-closed})\)

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

```isar
with nodes-are-scc \(\varphi z b\)-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.

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

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

with different

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

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

from \text{step}

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

qed simp

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

apply –

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

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

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

qed

**Lemma redundant-scc-phis:**

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

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

**using** \text{assms by} (meson domIff redundant-set-def \text{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** redundant-set \(g P\)

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

**proof** –

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

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

unfolding redundant-set-def by auto

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

moreover

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

moreover

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

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

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

moreover

from \text{assms}

obtain \(\text{pred}\) where \(\text{pred-props}: \text{pred} \in \text{allVars } g \forall \varphi \in P. \forall \varphi'. \text{phiArg } g \varphi \varphi' \rightarrow \varphi' \in P \cup \{\text{pred}\}\)

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 $P$

fix $\varphi \varphi'$

consider (in-P) $\varphi' \not\in \text{leaf} \land \varphi' \in P \mid (\text{neither}) \varphi' \not\in \text{leaf} \land \varphi' \in P \cup \{\text{pred}\} \mid 
\varphi' \not\in \text{leaf} \land \varphi' \in \{\text{pred}\} \mid \varphi' \in \text{leaf} \mid \text{by auto}

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

proof cases

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

moreover

from in-P that leaf-props(1) scc-in-P[of leaf g P]

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

ultimately

have (leaf, scc-of (induced-phi-graph g P) $\varphi'$) $\in$ condensation-edges g P unfolding condensation-edges-def

using leaf-props(1) that is-scc (induced-phi-graph g P) leaf

apply —

apply clarsimp

apply (rule conjI)

prefer 2

apply auto[1]

unfolding condensation-nodes-def

by (metis (no-types, lifting) is-scc-unique node-in-scc-of-node pair-imageI scc-of-is-scc)

with leaf-props(2)

show ?thesis by auto

next

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

with that leaf-props pred-props

have $\neg$redundant-set g P unfolding redundant-set-def

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

with assms

show ?thesis by auto

qed auto — the other cases are trivial

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
for g :: 'g and ϕ :: 'val
where refl: unnecessaryPhi g ϕ ⇒ ϕ ∈ reachable g ϕ
| step: ϕ' ∈ reachable g ϕ ⇒ phiArg g ϕ' ϕ'' ⇒ unnecessaryPhi g ϕ'' ⇒ ϕ'' ∈ reachable g ϕ

lemma reachable-props:
assumes ϕ' ∈ reachable g ϕ
shows (phiArg g)' ϕ' 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 ccontr)
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'} = ∪ (set 'b. ∃a phi g a = Some b)} unfolding phiArg-def 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 (ran (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 reachable \( g \varphi \subseteq \text{dom} (\text{phi} \ g) \)
  by (metis domIff reachable_cases subsetI unnecessaryPhi-def)

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

lemma phiArg-trancl-same-var:
assumes (\( \text{phiArg} \ g \)++ \( \varphi \) \( n \))
shows var g φ = 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 φ argument to the corresponding φ function while preserving disjointness to a second path.

lemma phiArg-disjoint-paths-extend:
assumes var g r = V and var g s = V and r ∈ allVars g and s ∈ allVars g
and V ∈ oldDefs g n and V ∈ oldDefs g m
and g ⊢ n−ns→defNode g r and g ⊢ m−ms→defNode g s
and set ns ∩ set ms = {}
and phiArg g φr obtains ns′
where g ⊢ n−ns@ns′→defNode g φr
and set (butlast (ns@ns′)) ∩ set ms = {}
proof (cases r = φr)
  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 ⊢ defNode g φr set (butlast (ns@ns′)) ∩ set ms = {}
  by simp-all fastforce
  with that show ?thesis .
  next
  case False
  It suffices to obtain any path from r to φr. However, since we’ll need the corresponding predecessor of φr later, we must do this as follows:
  from assms(10)
  have φr ∈ allVars g unfolding phiArg-def
    by (metis allDefs-in-allVars phiDefs-in-allDefs phi-def phi-phiDefs phis-in-αn)
  with assms(10)
  obtain rs′ pred φr where rs′-props: g ⊢ defNode g r−rs′→predφr old.EntryPath
  g rs′ r ∈ phiUses g predφr predφr ∈ set (old.predecessors g (defNode g φr))
  by (rule phiArg-path-ex′)
  define rs where rs = rs′@[defNode g φr]
  from rs′-props(2,1) old.EntryPath-distinct old.path2-hd
  have rs′-loopfree: defNode g r ∉ set (tl rs′) by (simp add: Misc.distinct-hd-tl)
  from False assms have defNode g φr ≠ defNode g r
    apply –
    apply (rule phiArg-distinct-nodes)
    apply (auto intro:phiArg-in-allVars)[2]
unfolding phiArg-def by (metis allDefs-in-allVars phiDefs-in-allDefs phi-def phi-phiDefsphis-in-xn)

from rs'-props
have rs props: g ⊢ defNode g r → 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 ccontr, 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-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-tlD 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 ⊢ defNode g r → i 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: $g \vdash m \rightarrow \text{prefix } ms' ms i \notin \text{set (butlast } ms')$ 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 $\varphi$ argument to the $\varphi$ itself leads to a contradiction. However, we still need to distinguish the cases of whether $m = i$

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

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

  moreover
  have $\text{set (butlast } (\text{ns } (\text{tl } ri))) \cap \text{set } ms = \{}$
  proof (rule ccontr)
    assume $\text{contr}$: $\text{set (butlast } (\text{ns } (\text{tl } ri))) \cap \text{set } ms \neq \{}$
    from this
    obtain $x$ where $x$-props: $x \in \text{set (butlast } (\text{ns } (\text{tl } ri))) x \in \text{set } ms$ by auto
    with assms(9) have $x \notin \text{set } ns$ by auto
    with $x$-props $(g \vdash n \rightarrow \text{ns } (\text{tl } ri) \rightarrow \text{defNode } g \varphi_r) \langle \text{defNode } g \varphi_r \neq \text{defNode } g$
    r), assms(7)
    have $x \in \text{set (butlast } (\text{tl } ri))$
      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-tlD List.butlast-tl)
  qed
ultimately
show thesis by (rule that)
next
  case $m$-same

  If $m = i$, we have, with $m$, a variable definition on the path from a $\varphi$ 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 $r \in \text{allUses } g \text{pred}_\varphi$ unfolding allUses-def
by simp

moreover
from rs-props(1) m-i-same rs-def ri-props(1,2,4) \langle \text{defNode } g \varphi_r \neq \text{defNode } g \rangle \text{ assms(7,9)}

have \( m \in \text{set (tl } rs') \)

by (metis disjoint-elem hd-append in-hd-or-tl-conv in-prefix list.sel(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 \ \text{var } g \ \text{def}_m = V

unfolding oldDefs-def using defss-in-allDefs by blast

ultimately
have \( \text{var } g \ \text{def}_m \neq \text{var } g \ r \) by -(rule conventional, simp-all)

with \( \langle \text{var } g \ \text{def}_m = V \rangle \text{ assms(1)} \)

have False by simp

thus ?thesis by simp

next
case m-i-differ

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-differ ms'-props old.path2-nontriv by blast

next

show 1 < length (ns @ tl ri) using ri-props 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 ccontr)

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-appendI 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 butlast-append in-set-butlastD set-append)

hence i' \in set (butlast ri) by (simp add:in-set-tlD List.butlast-tl)

with i'-not-in-ms ri-props(3)

show False by (auto dest:in-set-butlastD)

qed (meson disjoint-elem in-set-butlastD)

qed

qed (auto intro: assms(7) ri-props(1) old.path2-app ms'-props(1))
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-rs'-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 $rs'$-rest-prop: $rs' = ri@rs'$-rest by (simp add:old.path2-split)
moreover note $\langle \text{var } g \quad r = V \rangle$ from $rs'$-props(3) have $r \in \text{allUses } g \text{ pred}_{\phi_r}$ unfolding allUses-def by simp

moreover from $\langle \text{defNode } g \quad r \notin \text{ set } (tl rs') \rangle \text{ rs'}-rest-def have defNode g v $\notin$ set $rs'$-rest by auto with (g $\vdash \quad i - i#rs'$-rest $\rightarrow$ pred$_{\phi_r}$) have $\forall x. x \in \text{ set } rs'$-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) $g \vdash \quad i - i#rs'$-rest $\rightarrow$ pred$_{\phi_r}$ 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-cn) 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 $g \vdash \quad n$-ns $\rightarrow$ defNode g $r$ have defNode g $r \in$ set ns by auto
with \( \{ i \in \text{set} \, \text{ms} \} \) have \( i \neq \text{defNode} \, g \, r \) by auto

moreover

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

moreover

have \( \text{defNode} \, g \, r \in \text{set} \, (\alpha \, g) \) by (simp add: assum(2))

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

ultimately have \( \text{False} \)

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

unfolding \( \text{allDefs-def} \, \phiDefs-def \)

apply clarsimp

apply (erule-tac c=r in equalityCE)

using \( \phi-def \, \phi-\phiDefs \)

thus \( \phi ? \) by simp

qed

qed

qed

lemma \text{reachable-same-var}:
assumes \( \phi' \in \text{reachable} \, g \, \phi \)
shows \( \var \, g \, \phi = \var \, g \, \phi' \)

using assum by (metis \text{Nitpick.rtranclp-unfold} \text{phiArg-trancl-same-var} \text{reachable-props(1)})

lemma \( \phi\)-node-no-defs:
assumes unnecessaryPhi \( g \) \( \phi \, \phi \in \text{allVars} \, g \, \var \, g \, \phi \in \text{oldDefs} \, g \, n \)
shows \( \text{defNode} \, g \, \phi \neq n \)

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

unfolding unnecessaryPhi-\text{def by auto}

lemma \( \text{defNode-differ-aux} \):
assumes \( \phi_s \in \text{reachable} \, g \, \phi \, \phi \in \text{allVars} \, g \, \var \, g \, \phi_s \neq s \, \var \, g \, \phi = \var \, g \, s \)
shows \( \text{defNode} \, g \, \phi_s \neq \text{defNode} \, g \, s \) \text{unfolding} \text{reachable-def}

proof (rule contr)

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

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

from assum(1)

have \( \text{vars-eq} \): \( \var \, g \, \phi = \var \, g \, \phi_s \)

apply --

apply (cases \( \phi = \phi_s \))

apply simp

apply (rule phiArg-trancl-same-var)

apply (drule reachable-props)

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

proof (cases $\phi = \phi_s$)
  case False
    with assms(1)
    obtain $\phi'$ where phiArg g $\phi'$ $\phi_s$ by (metis rtranclp.cases reachable-props(1))
    thus $\phi_s \in \text{allVars } g$ by (rule phiArg-in-allVars)
  next
    case eq: True
    with assms(2)
    show $\phi_s \in \text{allVars } g$ by (subst eq[symmetric])
  qed

from eq $\phi_s$-in-allVars assms(3,4)
have var g $\phi_s \neq \text{var } s$ by (rule defNode-var-disjoint)
with vars-eq assms(5)
show False by 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: $\neg(\exists P. \text{redundant-set } g P)$
  shows cytronMinimal g
proof (rule ccontr)
  assume $\neg$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 $\phi$-props: unnecessaryPhi g $\phi$ $\phi_s \in \text{allVars } g \phi \in \text{reachable } g \phi$
    using cytronMinimal-def unnecessaryPhi-def reachable-def unnecessaryPhi-def reachable.intros 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 $\phi$-props(1)
      have redundant-set g (reachable g $\phi$) by (rule few-preds-redundant)
      with no-redundant-set
      show False by 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:

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

apply simp
apply (rule set-take-two[OF nontrivial])
apply simp

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

moreover from assign-nodes-props this
have $\varphi\text{-r-s-uneq}: \varphi \neq r \ \varphi \neq s$ using $\varphi$-props by auto

moreover from assign-nodes-props this
have $r\text{-s-in-tranclp}: (\text{phiArg } g)^++ \ \varphi \ r \ (\text{phiArg } g)^++ \ \varphi \ s$
by (meson mem-Collect-eq rtranclpD) (meson assign-nodes-props(7) $\varphi\text{-r-s-uneq}(2)$ mem-Collect-eq rtranclpD)

from this
obtain $V$ where $V$-props: $\var r \ g = V \ \var s \ g = V \ \var \var g = V$ by (metis phiArg-trancl-same-var)

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

moreover from $V$-props $\text{defNode-var-disjoint } r\text{-s-allVars}$ assign-nodes-props(1)
have $r\text{-s-defNode-distinct}: \text{defNode } g \ r \neq \text{defNode } g \ s$ by auto

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

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

have $n\text{-m-distinct}: n \neq m$

proof (rule ccontr)
  assume $n\text{-m}: \neg n \neq m$
  with $r\text{-s-path-props}(2) \ \text{old.path2-hd-in-ns}$
  have $n \in \text{set } ns$ by blast
  moreover from $n\text{-m} \ r\text{-s-path-props}(4) \ \text{old.path2-hd-in-ns}$
  have $n \in \text{set } ms$ by blast
  ultimately show False using $r\text{-s-path-props}(5)$ by auto

qed

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

from $V$-props $r\text{-s-allVars}$ $r\text{-s-path-props}$ assign-nodes-props
obtain $rs$ where $rs$-props: $g \vdash n - ns\odot rs \to \text{defNode } g \varphi_r$ set (butlast (ns@rs)) \nset ms = \{}$

using phiArg-disjoint-paths-extend by blast

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

have \nset ms = \{} set ms \quad proof (rule ccontr)
  assume $\varphi_r$-in-ms: $\neg \text{defNode } g \varphi_r$ \nset ms

from this r-s-path-props(4)

obtain $ms'$ where $ms'$-props: $g \vdash m - ms' \to \text{defNode } g \varphi_r$ prefix $ms'$ ms by

-(rule old.path2-prefix-ex[of g m ms defNode g s defNode g $\varphi_r$], auto)

have old.pathsConverge $n$ (ns@rs) m $ms'$ (defNode g $\varphi_r$)

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:

\nset (butlast ns @ rs)
\nset (butlast ms') by auto

from this(2) $ms'$-props(2)

have $c$ \nset ms by (simp add: in-prefix in-set-butlastD)

with $c$-props(1) $rs$-props(2)

show $False$ by auto

qed

next

have $m - n$, $\varphi_r$-differ: $n \neq \text{defNode } g \varphi_r$, $m \neq \text{defNode } g \varphi_r$

using assign-nodes-props(2,3,4,5) V-props r-s-path-props $\varphi_r$-in-ms

apply fastforce

using V-props(1) $\varphi_r$-in-ms assign-nodes-props(8) old.path2-in-an phiArg-def

phiArg-same-var r-s-path-props(3,4) simpleDefs-phiDefs-var-disjoint

by auto

with $ms'$-props(1)

show $1 < \text{length } ms'$ using old.path2-nontriv by simp

from $m - n$, $\varphi_r$-differ rs-props(1)

show $1 < \text{length } (ns@rs)$ 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 $g \varphi_r$, unfolding necessaryPhi-def using assign-nodes-props(8)

phiArg-same-var $g \varphi_r$, 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 (metis disjoint-iff-not-equal not-in-butlast old.path2-last)

have $\varphi_r$-V: var $g \varphi_r = V$

using V-props(1) assign-nodes-props(8) phiArg-same-var by auto
have \( \varphi_r \in \text{allVars} \) \( \varphi_r \in \text{allVars} \)

by (meson \( \text{phiArg-def assign-nodes-props(8)} \) \( \text{allDefs-in-allVars old.path2-tl-in-cn phiDefs-in-allDefs phi-phiDefs rs-props} \))

define \( p_m \) where \( p_m = \text{ms@ss} \)

define \( p_n \) where \( p_n = \text{ns@rs} \)

have \( \text{ind-props: } g \vdash m \rightarrow \text{defNode} g \: \varphi_s \: g \vdash n \rightarrow \text{defNode} g \: \varphi_r \: \) \( \text{set (butlast } p_m) \cap \text{set (butlast } p_n) = \{\} \)

using \( \text{rs-props(1)} \) \( \text{ss-props} p_m, \text{def p_n-def by auto} \)

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 \( \text{reachable} \ g \: \varphi \), yielding the path convergence we seek.

have \( \text{path-crossing-yields-convergence: } \exists \varphi_z \in \text{reachable} \ g \: \varphi \: \exists \text{ns ms. old.pathsConverge} g \: n \: s \: m \: s | m | s | n | m | n | \) \( \text{defNode} g \: \varphi_z \) \n
if \( \varphi_r \in \text{reachable} \ g \: \varphi \: \) \( \varphi_s \in \text{reachable} \ g \: \varphi \: \text{and } g \vdash n \rightarrow \text{defNode} g \: \varphi_r \)

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

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

for \( \varphi_r \: \varphi_s \: p_m \: p_n \)

proof –

from that(6) \( \text{split-list-first-propE} \)

obtain \( p_m' \: n_2 \: p_n' \) where \( \text{n2-props: } n_2 \in \text{set } p_n \: p_m = p_m' \: @ \: \text{n2} \: # \: p_m' \)

\forall n \in \text{set } p_m' \: n \notin \text{set } p_n

by (auto intro: \( \text{split-list-first-propE} \))

with that(3,4)

obtain \( p_n' \) where \( \text{p'n-props: } g \vdash n \rightarrow \text{prefix } p_n' \: p_n' \text{ n2 } \notin \text{set (butlast } p_n) \)

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

from \( \text{V-props(3)} \) \( \text{reachable-same-var[OF that(1)] reachable-same-var[OF that(2)]} \)

have \( \text{phis-V: var g } \varphi_r \: = \: V \: \text{var g } \varphi_s \: = \: V \) by simp-all

from \( \text{reachable-props(1) that(1,2) \phi-props(2) phiArg-in-allVars} \)

have \( \text{phis-allVars: } \varphi_r \in \text{allVars} \: \varphi_s \in \text{allVars} \: g \) by (metis \( \text{rtranclp.cases} \))

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 \(\varphi\)-node-no-defs reachable-props(2) that(2) r-s-path-props(1,3) phis-V(2)
that phis-allVars

**have** \(m \neq \text{defNode}\ g \varphi_s \land n \neq \text{defNode}\ g \varphi_r\) by blast+

With this scenario, since \(\text{set}\ (\text{butlast}\ p_n) \cap \text{set}\ (\text{butlast}\ p_m) = \{\}\), 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 \(n_z\)-props
**consider** \((p_n\text{-ends-in-}p_m)\) \(n_z = \text{defNode}\ g \varphi_s\ \land\ (p_m\text{-ends-in-}p_n)\) \(n_z = \text{defNode}\ g \varphi_r\)

**proof** (cases \(n_z = \text{last}\ p_n\))

**case** True

**with** \(g \vdash n - p_n \rightarrow \text{defNode}\ g \varphi_r\)

**have** \(n_z = \text{defNode}\ g \varphi_r\), using \(\text{old.path2-last by auto}\)

**with** that(2) show \(?\text{thesis}\).

**next**

**case** False

**from** \(n_z\)-props(2)

**have** \(n_z \in \text{set}\ p_m\) by simp

**with** \(\text{False}\ n_z\)-props(1) \(\vdash\ (\text{butlast}\ p_m) \cap \text{set}\ (\text{butlast}\ p_n) = \{\}\) \(\vdash g \vdash m - p_m \rightarrow \text{defNode}\ g \varphi_s\)

**have** \(n_z = \text{defNode}\ g \varphi_s\), by (meson disjoint-elem not-in-butlast old.path2-last)

**with** that(1) show \(?\text{thesis}\).

**qed**

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

**proof** (cases)

**case** \(p_n\text{-ends-in-}p_m\)

**have** \(\text{old.pathsConverge}\ g\ n\ p_n\ m\ p_m\ (\text{defNode}\ g \varphi_s)\)

**proof** (rule \(\text{old.pathsConvergeI}\))

**from** \(p_n\text{-ends-in-}p_m\) \(p_n\text{-props}(1)\) show \(g \vdash n - p_n \rightarrow \text{defNode}\ g \varphi_s\) by simp

**from** \(n \neq \text{defNode}\ g \varphi_s\) \(p_n\text{-ends-in-}p_m\) \(p_n\text{-props}(1)\) \(\text{old.path2-nontriv}\)

**show** \(1 < \text{length}\ p_n\) by auto

**from** that(4) show \(g \vdash m - p_m \rightarrow \text{defNode}\ g \varphi_s\).

**with** \(\text{m} \neq \text{defNode}\ g \varphi_s\) \(\text{old.path2-nontriv}\) show \(1 < \text{length}\ p_m\) by simp

**from** that \(p_n\text{-props}(3)\) show \(\text{set}\ (\text{butlast}\ p_n)\) \(\cap\ \text{set}\ (\text{butlast}\ p_m) = \{\}\)

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** \(\text{old.pathsConverge}\ g\ n\ p_m\ m\ (p_m\ ?[n_z])\ (\text{defNode}\ g \varphi_r)\)

**proof** (rule \(\text{old.pathsConvergeI}\))

**from** \(p_m\text{-ends-in-}p_m\) \(p_m\text{-props}(1,2)\) show \(g \vdash n - p_n \rightarrow \text{defNode}\ g \varphi_r\) \(g \vdash m - p_m\ ? [n_z] \rightarrow \text{defNode}\ g \varphi_r\) by simp-all

**with** \(n \neq \text{defNode}\ g \varphi_r\) \(\text{m} \neq \text{defNode}\ g \varphi_r\) show \(1 < \text{length}\ p_n\) \(1 < \text{length}\ (p_m\ ?[n_z])\)

using \(\text{old.path2-nontriv}\[\text{of}\ g\ m\ p_m\ ? [n_z]]\ \text{old.path2-nontriv}\[\text{of}\ g\ n]\) by
simp-all

from nz-props p n \cdot-props(3) show set (butlast p n) \cap set (butlast (p m \cdot \circ [n_z])) = {}
  using butlast-snoc disjointI 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 \( \phi \), these paths must at some point converge within reachable \( g \ \phi \).

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 ns ms. \ \text{old.pathsConverge } g \ n ns m ms \ (\text{defNode } g \ \varphi_z) \\
  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.pathsConverge g n p n m p m (\text{defNode } g \ \varphi)
      proof (rule old.pathsConvergeI)
        show \ 1 < length p n \ 1 < length p m
          using refl \ V-props simpleDefs-phiDefs-var-disjoint unfolding unneces-saryPhi-def
          by (metis domD domIfI old.path2-hd-in-an old.path2-nontriv phi-phiDefs \\
            r-s-path-props(1) r-s-path-props(3)+
            show \( g \vdash n-p_n\rightarrow\text{defNode } g \ \varphi \) \( g \vdash m-p_m\rightarrow\text{defNode } g \ \varphi \) \text{set (butlast p m)} \\
              \( \cap \) set (butlast p n) = {}
              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 disjointI disjoint-elem)
      next
      case True

21
If the paths are intersection-free, we can apply our path extension lemma to obtain the path needed.

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

from this(2) have set \( \text{butlast } p_m \) \( \cap \) set \( \text{butlast} (p_n \land ns) = \{ \} \)
  using in-set-butlastD by fastforce
moreover
from phiArg-same-var step.hyps(2) step.prems(5) have \( \varphi' = V \)
  by auto
moreover
have \( \varphi' \in \text{allVars } g \)
  by (metis \( \varphi\)-props V-props \langle \text{defNode } g \varphi' \rangle)
ultimately
show \( \exists \varphi \in \text{reachable } g. \exists ns ms. \text{oldPathsConverge } g n ns m ms \)
  using step.prems(1) \( \varphi\)-props V-props \( g \vdash n - p_n \land ns \rightarrow \text{defNode } g \varphi' \)
  by -(rule step.IH; blast)
qed
next
  case (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 \( \varphi_s \rightarrow V \) var \( g \varphi_s = V \)
    using step(1,2,9,10) reachable-same-var
    by (simp add: phiArg-same-var)
  from step(2) have \( \varphi_s \rightarrow \text{allVars } g \) \( \varphi_s \in \text{allVars } g \)
    by (rule phiArg-in-allVars)
  obtain \( p_m' \) where tmp: \( g \vdash m - p_m \land p_m' \rightarrow \text{defNode } g \varphi' \) set \( \text{butlast} (p_m \land p_m') \) \( \cap \) set \( \text{butlast} p_n = \{ \} \)
    by (rule phiArg-disjoint-paths-extend[of \( g \varphi_s \) V \( \varphi_r \) m n p_m p_n \( \varphi' \)])
    (metis \( \varphi_s \rightarrow V \varphi_s \rightarrow \text{allVars } g \) \( \text{r-s-path-props}(1,3) \) 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.IH[of \( p_m \land p_m' \) 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 $g$ n ns m ms (defNode $g \varphi_z$)
by blast
moreover
with reachable-props have $\text{var } g \varphi_z = V$ by (metis V-props(3) phiArg-trancl-same-var rtranclpD)
ultimately have necessaryPhi' $g \varphi_z$ using r-s-path-props
unfolding necessaryPhi-def by blast
moreover with $\langle \varphi_z \in \text{reachable } g \varphi \rangle$ 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 $\neg (\exists P \text{ scc. redundant-scc } g P \text{ scc})$
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 $\varphi$ functions. Intuitively, this is because a set containing only a trivial $\varphi$ 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

