Minimal Static Single Assignment Form

Max Wagner          Denis Lohner

October 27, 2022

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

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 \( g P \equiv P \neq \emptyset \land P \subseteq \text{dom}(\phi g) \land (\exists v' \in \text{allVars } g. \forall \varphi \in P. \forall \varphi'. \phi \text{Arg } g \varphi \varphi' \rightarrow \varphi' \in P \cup \{v'\}) \)
definition redundant-scc \( g P \text{ scc } \equiv \text{redundant-set } g \text{ scc } \land \text{is-scc } (\text{induced-phi-graph } g P) \text{ scc} \)

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

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

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

hence (∀ f. 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 (ϕ_a ∈ a)
assumes (ϕ_b ∈ b)
shows (ϕ_a, ϕ_b) ∈ (induced-phi-graph g P)^*
unfolding condensation-edges-def
proof —
from assms(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
hence x ∈ a y ∈ b by auto

All that’s left is to combine these paths.

with assms(2) x-y-props(2)
have (ϕ_a, x) ∈ (induced-phi-graph g P)^* by (meson is-scc-connected scc-of-is-scc)
moreover with assms(3) x-y-props(3) y ∈ b
have (y, ϕ_b) ∈ (induced-phi-graph g P)^* by (meson is-scc-connected scc-of-is-scc)
ultimately
show (ϕ_a, ϕ_b) ∈ (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 (ϕ_a ∈ a)
assumes (ϕ_b ∈ b)
shows (ϕ_a, ϕ_b) ∈ (induced-phi-graph g P)^*
using assms
proof (induction arbitrary: ϕ_b rule:trancl-induct)
fix y z ϕ_b
assume (y, z) ∈ condensation-edges g P
hence is-scc \((\text{induced-phi-graph } g P)\) \text{ unfolding} condensation-edges-def by auto
  
  hence \(\exists \varphi_y. \varphi_y \in y\) using scc-non-empty' by auto
  then obtain \(\varphi_y\) where \(\varphi_y\)-in-y: \(\varphi_y \in y\) by auto

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

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

}\lem condensation-acyclic: 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: \((x, x) \notin \text{condensation-edges } g P)\) unfolding condensation-edges-def by auto

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

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

  hence \(x \in \text{condensation-nodes } g P\) \(b \in \text{condensation-nodes } g P\) using con-
densation-nodes-edges by auto
  hence nodes-are-scc: is-scc \((\text{induced-phi-graph } g P)\) \(x\) is-scc \((\text{induced-phi-graph } g P)\) \(b\)
  using scc-of-is-scc 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-scc scc-non-empty' ex-in-conv
  by auto
  then obtain \(\varphi_x\ \varphi_b\) where \(\varphi_x\)-elem: \(\varphi_x \in x\) \(\varphi_b \in b\) by metis
  with \(\text{nodes-are-scc(1)}\) \(b\)-on-path path-in-condensation-impl-path condensation-edge-impl-path
  \(\varphi_x\)-elem(2)
  have \(\varphi_b \in x\)
by \hspace{1em} (\text{rule is-scc-closed})

This however means $x$ and $b$ must be the same SCC, which is a contradiction to the nonreflexivity of condensation-edges.

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

qed

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

\begin{lemma}
\text{Ex-condensation-leaf:}
\end{lemma}

\begin{assumes}
P \neq \{\}
\end{assumes}

\begin{shows}
\exists \text{leaf}. \text{ leaf } \in (\text{condensation-nodes } g \ P) \land (\forall \text{scc}.(\text{leaf}, \text{scc}) \notin \text{condensation-edges } g \ P)
\end{shows}

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

\begin{lemma}
\text{scc-in-P:}
\end{lemma}

\begin{assumes}
\text{scc } \in \text{condensation-nodes } g \ P
\end{assumes}

\begin{shows}
\text{scc } \subseteq \ P
\end{shows}

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

with different

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

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

from step

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

qed simp

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

apply –

apply (rule imageE[of scc scc-of (induced-phi-graph g P)])

using condensation-nodes-def x-node by blast+

thus \(?thesis\) by clarify

qed

lemma redundant-scc-phis:

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

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

using assms by (meson domIff redundant-set-def scc-in-P subsetCE)

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

Lemma 1. Every redundant set contains a redundant SCC.
Any argument of a $\phi$ function in the leaf SCC which is not in the leaf SCC itself must be the unique argument of $P$

$$\text{fix } \varphi \varphi'$$

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

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

$$\text{proof cases}$$

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

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

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

$$\text{ultimately have } (\text{leaf, scc-of (induced-phi-graph g P) } \varphi') \in \text{condensation-edges g P unfolding condensation-edges-def}\text{ using leaf-props(1) that } \langle \text{is-scc (induced-phi-graph g P) leaf}\rangle$$

$$\text{apply –}$$

$$\text{apply clarsimp}$$

$$\text{apply (rule conjI)}$$

$$\text{prefer 2}$$

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

$$\text{unfolding condensation-nodes-def}\text{ by (metis (no-types, lifting) is-scc-unique node-in-scc-of-node pair-imageI scc-of-is-scc)}$$

$$\text{with leaf-props(2)}$$

$$\text{show ?thesis by auto}$$

$$\text{next case neither} — \text{In which case P itself wasn’t redundant, a contradiction}$$

$$\text{with that leaf-props pred-props}\text{ have } \neg \text{redundant-set g P unfolding redundant-set-def}\text{ by (meson rev-subsetD scc-in-P)}$$

$$\text{with assms}\text{ show ?thesis by auto}$$

$$\text{qed auto} — \text{the other cases are trivial}$$

$$\text{1.2 Proof of Minimality}$$

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]: true Args 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-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 reachableintros(1) by auto
next
  from assms(2)
show reachable g ϕ ⊆ dom (phi g) by (metis domIff reachable.cases subsetI unnecessaryPhi-def)

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

lemma phiArg-trancl-same-var:
assumes (phiArg g)++ ϕ 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@[ ]→defNode g φr
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 ⊢ n−ns@[ ]→defNode g φr
  set (butlast (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’ _predφr 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]

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

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

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)) ∩ set ms
    by auto
    with rs-props(2)
    consider (in-n) x ∈ set ns | (in-rs) x ∈ set (butlast (tl rs)) by (metis Un-iff butlast-appd 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→ ri→ i ∈ set ms ∀ n ∈ set (butlast ri). n ∉ set ms prefix ri rs
  apply –
  apply (rule old.path2-split-first-prop[of g defNode g r rs defNode g φ_r, where P=λm. m ∈ set ms])
  apply blast
  apply (metis disjoint-iff-not-equal in-set-butlastD)
  by blast
  with assms(8) old.path2-prefix-ex
obtain $ms'$ where $ms'$-props: $g \vdash m \rightarrow ms' \rightarrow i$ prefix $ms'$ $ms \notin \text{set (butlast } ms')$ 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 $(ri-is-valid) i = \text{defNode } g \varphi_r$ | $(m-i\text{-same}) i \neq \text{defNode } g \varphi_r, m = i$ | $(m-i\text{-differ}) i \neq \text{defNode } g \varphi_r, m \neq i$ by auto

thus thesis

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

moreover have $\text{set (butlast } (ns \langle tl ri \rangle)) \cap \text{set } ms = \{\}$

proof (rule ccontr)
  assume contr: $\text{set (butlast } (ns @ tl ri)) \cap \text{set } ms \neq \{\}$ from this
  obtain $x$ where $x$-props: $x \in \text{set (butlast } (ns @ tl ri)) \cap \text{set } ms$ by auto
  with $\text{assms}(9)$ have $x \notin \text{set } ns$ by auto
  with $x$-props $\langle g \vdash n - ns \rightarrow \text{defNode } g \varphi_r \rangle \langle \text{defNode } g \varphi_r \neq \text{defNode } g\rangle$

  have $x \in \text{set (butlast } (tl ri))$
  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 $m$-i\text{-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 $r \in \text{allDefs } g \langle \text{defNode } g r \rangle$ by (simp add: $\text{assms}(3)$)

moreover from $rs'$-props(3) have $r \in \text{allUses } g \text{pred}_{\varphi r}$ 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 \, r\rangle \text{assms}(7,9)

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

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 \text{assms}(6)\ obtain \text{def}_m \, \text{where} \text{def}_m \in \text{allDefs} \, g \, m \, \text{var} \, g \, \text{def}_m = V

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

ultimately

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

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

have False by simp

thus \text{thesis} by simp

next

case m-i-differ

If \(m \neq i\), \(i\) constitutes a proper path convergence point.

have \text{old.pathConverge} \, g \, m \, ms' \, n \, (ns @ tl \, ri) \, i

proof (rule old.pathConvergeI)

show \(1 < \text{length} \, ms'\) using m-i-differ props old.path2-nontriv by blast

next

show \(1 < \text{length} \, (ns @ tl \, ri)\)

using ri-props old.path2-nontriv assms\((9)\) by (metis assms\((7)\) disjoint-elem old.path2-app old.path2-hd-in-ns)

next

show \(\text{set} \, (\text{butlast} \, ms') \cap \text{set} \, (\text{butlast} \, (ns @ tl \, ri)) = \{\}\)

proof (rule ccontr)

assume \(\text{set} \, (\text{butlast} \, ms') \cap \text{set} \, (\text{butlast} \, (ns @ tl \, ri)) \neq \{\}\)

then obtain \(i'\) where \(i'\)-props: \(i' \in \text{set} \, (\text{butlast} \, ms')\) \(i' \in \text{set} \, (\text{butlast} \, (ns @ tl \, ri))\) by auto

with \(ms'\)-props\((2)\)

have \(i'\)-not-in-ms: \(i' \in \text{set} \, (\text{butlast} \, ms)\) by (metis in-set-butlast-appendI prefixE)

with assms\((9)\)

show False

proof (cases \(i' \neq \text{set} \, ns\))

case True

with \(i'\)-props\((2)\)

have \(i' \in \text{set} \, (\text{butlast} \, (tl \, ri))\)

by (metis Un-iff butlast-append in-set-butlastD set-append)

hence \(i' \in \text{set} \, (\text{butlast} \, ri)\) 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-differ 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' = tmp@i by auto more```

```from old.path2-last[OF ri-props(1)] last-snoc[of - i] obtain tmp where rs'-rest-prop: rs' = tmp@i by auto more```

```note (var g r = V) [simp] from rs'-props(3) have r ∈ allUses g pred$_\phi_r$ 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$_\phi_r$) have $\forall . . x ∈ set rs'$-rest $\Rightarrow$ r $\notin$ allDefs g x by (metis defNode-eq list.distinct(1) list.sel(3) list.set-cases old.path2-cases old.path2-in-on)```

```moreover from assms(7,9) (g $\vdash$ i $-$ i#$rs'$-rest $\rightarrow$ pred$_\phi_r$) ri-props(2) have r $\notin$ defNode g i by (metis defNode-eq defNode-in-allDefs disjoint-elem old.path2-hd-in-cnv old.path2-last-in-nss) 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. phis g (i, r) = Some y by (rule convergence-prop [where g=g and n=i and v=r] simplified) moreover from (g $\vdash$ n-ns$\rightarrow$defNode g r) have defNode g r $\in$ set ns by auto```

14
with \( \langle \text{set } ns \cap \text{set } ms = \{\} \rangle \langle i \in \text{set } ms \rangle \) have \( i \neq \text{defNode } g \ r \) by auto
moreover

from \( ms'.\text{-props(1)} \) 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 \( \text{False} \)
  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 \( \text{equalityCE} \))
  using \( \text{phi-def phis-phi} \) by auto
thus \( \text{thesis} \) by simp
qed
qed
qed

lemma \( \text{reachable-same-var} \):
  assumes \( \varphi' \in \text{reachable } g \varphi \)
  shows \( \text{var } g \varphi = \text{var } g \varphi' \)
  using assms by (metis Nitpick.rtranclp_unfold phiArg_trancl_same_var reachable_prop(1))

lemma \( \varphi\text{-node-no-defs} \):
  assumes \( \text{unnecessaryPhi } g \varphi \varphi \in \text{allVars } g \text{ var } g \varphi \in \text{oldDefs } g \ n \)
  shows \( \text{defNode } g \varphi \neq n \)
  using assms simpleDefs_phiDefs_var_disjoint defNode(1) notNone_eq phi_phiDefs
  unfolding unnecessaryPhi_def by auto

lemma \( \text{defNode-differ-aux} \):
  assumes \( \varphi_s \in \text{reachable } g \varphi \varphi \in \text{allVars } g \ s \in \text{allVars } g \varphi_s \neq s \text{ var } g \varphi = \text{var } g \varphi_s \)
  shows \( \text{defNode } g \varphi_s \neq \text{defNode } g \ s \) unfolding 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 \( \text{vars-eq: var } g \varphi = \text{var } g \varphi_s \)
      apply -
      apply (cases \( \varphi = \varphi_s \))
      apply simp
      apply (rule phiArg_trancl_same_var)
      apply (drule reachable_prop)
    unfolding reachable_def by (meson IntD1 mem_Collect_eq rtranclpD)
have \( \phi \in \text{allVars} \): \( \phi \in \text{allVars} g \) 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 \( \phi_s \in \text{allVars} g \) \( \phi_s \) \( \phi \) \( \phi \in \text{allVars} g \) \( \phi \in \text{reachable} g \) 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 \( \phi \)) \( \geq 2 \mid \) (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 \( \text{reachable} g \) \( \phi \) by (rule few-preds-redundant)

with no-redundant-set
show False by simp
next
  case nontrivial

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 \( \neg \text{cytronMinimal} g \)
    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 \( \phi \)-props: unnecessaryPhi g \( \phi \) \( \phi \in \text{allVars} g \) \( \phi \in \text{reachable} g \) \( \phi \)
    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 \( \phi \)) \( \geq 2 \mid \) (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 \( \text{reachable} g \) \( \phi \) 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 $r\ s\ \varphi_\cdot\ \varphi_s$ where assign-nodes-props:

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

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
have $\varphi\cdot\text{-r-s-uneq}: \varphi \neq r\ \varphi \neq s$ using $\varphi\cdot\text{-props}$ by auto
moreover from assign-nodes-props this
have $r\cdot\text{-in-tranclp}: (\phiArg\ g)^*\ \varphi\ r\ (\phiArg\ g)^*\ \varphi\ s$
by (meson mem-Collect-eq rtranclpD) (meson assign-nodes-props(7) $\varphi\cdot\text{-r-s-uneq}(2)$ mem-Collect-eq rtranclpD)
from this obtain $V$ where $V\cdot\text{-props}: \text{var}\ g\ r = V\ \text{var}\ g\ s = V\ \text{var}\ g\ \varphi = V$ by (metis phiArg-trancl-same-var)
moreover from $r\cdot\text{-in-tranclp}$
have $r\cdot\text{-allVars}: r \in \text{allVars}\ g\ s \in \text{allVars}\ g\ \text{by (metis phiArg-in-allVars tranclp.cases)+}$
moreover from $V\cdot\text{-props}\ \text{defNode-var-disjoint}\ r\cdot\text{-allVars}\ assign-nodes-props(1)$
have $r\cdot\text{-defNode-distinct}: \text{defNode}\ g\ r \neq \text{defNode}\ g\ s$ by auto
ultimately obtain $n\ ns\ m\ ms$ where $r\cdot\text{-path-props}: V \in \text{oldDefs}\ g\ n\ g \vdash n\cdot\text{-ns} \rightarrow \text{defNode}\ g\ r\ V \in \text{oldDefs}\ g\ m\ g \vdash m\cdot\text{-ms} \rightarrow \text{defNode}\ g\ s$
set $\text{ns} \cap \text{set ms} = \{\}$ by (auto intro: unnecessaryPhis-disjoint-paths[of g r s])

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

These paths can be extended into paths reaching $\phi$ functions in our set.
from $V\cdot\text{-props}\ r\cdot\text{-allVars}\ r\cdot\text{-path-props}\ assign-nodes-props$
**obtain** \( rs \) where \( rs \)-props: \( g \vdash n - ns @ rs \rightarrow \text{defNode } g \varphi_r \) set (butlast (ns @ rs)) \[
\cap \text{ set } ms = \{\}
\]
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** \( \text{defNode } g \varphi_r \notin \text{ set } ms \)

**proof** (rule ccontr)

**assume** \( \varphi_r \)-in-ms: \( \neg \text{defNode } g \varphi_r \notin \text{ set } ms \)

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

**obtain** \( ms' \) where \( ms'\)-props: \( g \vdash m - ms' \rightarrow \text{defNode } g \varphi_r \) prefix \( ms' \) ms by

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

**have** \( \text{old} . \text{pathsConverge } g \ n \ (ns @ rs) \ m \ ms' \ (\text{defNode } g \varphi_r) \)

**proof** (rule old.path2-ConvergeI)

**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: \( c \in \text{ set } (\text{butlast } (\text{ns} @ \text{rs})) \) \( c \in \text{ set } (\text{butlast } ms') \) by auto

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

**have** \( c \in \text{ set } ms \) by (simp add: in-prefix in-set-butlastD)

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

**show** False by auto

**qed**

next

**have** \( m-n-\varphi_r\)-differ: \( n \neq \text{defNode } g \varphi_r \) \( m \neq \text{defNode } g \varphi_r \)

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

apply fastforce

using \( V\)-props(1) \( \varphi_r\)-in-ms \( \text{assign-nodes-props}(8) \) old.path2-in-\( \alpha \)n phiArg-def phiArg-same-var \( r\)-s-path-props(3,4) \( \text{simpleDefs-phiDefs-var-disjoint} \)

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** necessary\( \varphi' g \varphi \), unfolding necessary\( \varphi\)-def using \( \text{assign-nodes-props}(8) \)

phiArg-same-var by auto

with reachable-props(2)(OF \( \text{assign-nodes-props}(2) \))

**show** False unfolding unnecessary\( \varphi\)-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** \( \varphi_r-V \): \( \text{var } g \varphi_r = V \)

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

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

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

r-s-path-props(4) r-s-props(1) aax assign-nodes-props(9)

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

$\cap \ set (\text{butlast} \ (ns@rs)) = \{\}$

by (rule phiArg-disjoint-paths-extend) (metis disjoint-iff-not-equal in-set-butlastD)

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

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

have ind-props: $g \vdash m - p_m \rightarrow \text{defNode} g \varphi_s \ g \vdash n - p_n \rightarrow \text{defNode} g \varphi_r \ set$

(butlast $p_m) \cap \ set (\text{butlast} \ p_n) = \{\}$

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 $\varphi$ function in reachable $g \varphi$, yielding the path
convergence we seek.

have path-crossing-yields-convergence:

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

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

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

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

for $\varphi_r \varphi_s \ p_m \ p_n$

proof -

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

obtain $p_m^1 \ n_z \ p_m^2$ where $n_z$-props: $n_z \in \ set \ p_m \ p_m = p_m^1 \ @ \ n_z \neq \ p_m^2$

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

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

with that(3,4)

obtain $p_n'$ where $p_n'$-props: $g \vdash n - p_n' \rightarrow n_z \ g \vdash m - p_m \ @ (n_z) \rightarrow n_z$ prefix

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

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

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

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

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

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

have $m \neq \text{defNode} g \varphi_s$, $n \neq \text{defNode} g \varphi_s$ by blast+

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

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

thus $\exists \varphi_z \in \text{reachable } g \varphi$. $\exists \text{ns } m. \text{old.path2Converge } g \text{ } n \text{ } n s \text{ } m \text{ } m s \text{ (defNode } g \varphi_z)$

proof (cases)
  case $p_n\text{-ends-in-}p_m$
  have $\text{old.path2Converge } g \text{ } n \text{ } p_n' \text{ } m \text{ } p_m \text{ (defNode } g \varphi_s)$
  proof (rule old.path2ConvergeI)
  from $p_n\text{-ends-in-}p_m \text{ } p_n\text{-props}(1)$ show $g \vdash n - p_n' \rightarrow \text{defNode} g \varphi_s$, by simp
  from $\langle n \neq \text{defNode} g \varphi_s\rangle \text{ } p_n\text{-ends-in-}p_m \text{ } p_n\text{-props}(1) \text{ old.path2-nontriv}$
  show $l < \text{length } p_n' \text{ by auto}$
  from that(4) show $g \vdash m - p_m \rightarrow \text{defNode} g \varphi_s$.
  with $\langle m \neq \text{defNode} g \varphi_s\rangle \text{ old.path2-nontriv } \asthesis$, by simp
  from that $p_n$-props(3) show 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 $\asthesis$ by (auto intro:reachable.intros(2))
next
  case $p_m\text{-ends-in-}p_n$
  have $\text{old.path2Converge } g \text{ } n \text{ } p_n' \text{ } m \text{ } (p_m @ [n_z]) \text{ (defNode } g \varphi_r)$
  proof (rule old.path2ConvergeI)
  from $p_m\text{-ends-in-}p_n \text{ } p_n'\text{-props}(1,2)$ show $g \vdash n - p_n' \rightarrow \text{defNode} g \varphi_r \text{ } g \vdash m - p_m \rightarrow \text{defNode} g \varphi_r$, by simp-all
  with $n \neq \text{defNode} g \varphi_r \langle m \neq \text{defNode} g \varphi_r \rangle \text{ show } l < \text{length } p_n' \text{ } l < \text{length } (p_m @ [n_z])$
  using old.path2-nontriv[of $g$ $m$ $p_m$ $l$ $[n_z]$] old.path2-nontriv[of $g$ $n$] by
simp-all
from n-z-props p_n'-props(3) show set (butlast p_n') ∩ set (butlast (p_m I @ [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 φ, these paths must at some point converge within reachable g φ.

from assign-nodes-props(3,2) ind-props V-props(3) φ_r-V φ_r-allVars
have ∃φ_z ∈ reachable g φ. ∃ns ms. old.pathsConverge g n ns m ms (defNode g φ)
proof (induction arbitrary: p_m p_n rule: reachable.induct)
case refl
In the induction basis, we know that φ = φ_s, and a path to φ_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 φ_r = φ_s = φ, is trivial – φ suffices.

have old.pathsConverge g n p_n m p_m (defNode g φ)
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 domIff old.path2-hd-in-an old.path2-nontriv phi-phiDefs r-s-path-props(1) r-s-path-props(3)) +
    show g ⊢ n−p_n→defNode g φ g ⊢ m−p_m→defNode g φ set (butlast p_m)
      ∩ set (butlast p_n) = {}
      using refl by auto
    qed
  with ⟨φ ∈ reachable g φ⟩ show ?case by auto
next
case (step φ' φ_r)
In this case we have that φ = φ_s and need to acquire a path going to φ_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 ∩ set p_m = {}
  case paths-cross: False
  with step reachable.intros
  show ?thesis using path-crossing-yields-convergence[of φ_r φ 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.

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

step(2)

obtain ns where g ⊢ n - p_n@ns→ defNode g \varphi' set (butlast (p_n@ns)) ∩ set p_m = {} by (rule phiArg-disjoint-paths-extend)

from this(2) have set (butlast p_m) ∩ set (butlast (p_n @ ns)) = {}

using in-set-butlastD by fastforce

moreover

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

moreover

have \varphi' \in allVars g

by (metis \varphi'-props(2) phiArg-in-allVars reachable.cases step.hyps(1))

ultimately

show \exists \varphi \in reachable g. \exists ns ms. old.pathsConverge g n ns m ms (defNode g \varphi)

using set butlastD by blast

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 ∩ set p_n = {})

case True

have \varphi_s-V: var g \varphi_s = V using step(1,2,3,9) reachable-same-var by (simp add: phiArg-same-var)

from step(2) have \varphi_s-allVars: \varphi_s \in allVars g by (rule phiArg-in-allVars)

obtain p_m' where tmp: g ⊢ m - p_m@p_m'→ defNode g \varphi' set (butlast (p_m@p_m')) ∩ set (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-V \varphi_s-allVars step r-s-path-props(1,3) True disjoint-iff-not-equal in-set-butlastD+)

from step(5) this(1) step(7) this(2) step(9) step(10) step(11)

show ?thesis by (rule step.IH[of p_m@p_m' p_n])

next

case paths-cross: False

with step reachable.intro

show ?thesis using path-crossing-yields-convergence[of \varphi_r \varphi_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-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 $\neg(\exists P \text{ scc. redundant-sec } g \ P \text{ sec})$

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 $\varphi$ 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 $\neg$redundant $g$

proof –

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

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

