Minimal Static Single Assignment Form

Max Wagner  Denis Lohner

August 16, 2018

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.

\textbf{theory irreducible}

\textbf{imports Formal-SSA.Minimality}

\textbf{begin}

\textbf{context CFG-SSA-Transformed}

\textbf{begin}

\subsection*{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.

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

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

\textbf{definition redundant-set} $g \ P \equiv \{\} \land \ P \subseteq \text{dom} \ (phi \ g) \land (\exists \ v' \in \text{allVars} \ g. \ \forall \varphi \in \ P. \ \forall \varphi'. \ \phiArg \ g \ {\varphi, \varphi'} \rightarrow \varphi' \in \ P \cup \{v'\})$

\textbf{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.

\textbf{definition condensation-nodes} $g \ P \equiv \text{sec-of} \ (\text{induced-phi-graph} \ g \ P) \ \cdot \ P$

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

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

\textbf{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.

\textbf{proof} –

\textbf{let} $\phiEdges = \{(a, b). \ \phiArg \ g \ a \ b\}$

\textbf{have} finite $\phiEdges$

\textbf{proof} –

\textbf{let} $\phiDomRan = (\text{dom} \ (phi \ g) \times \bigcup \ (\text{set} \ \cdot \ (\text{ran} \ (phi \ g))))$

\textbf{from} phi-finite

\textbf{have} finite $\phiDomRan$ \textbf{by} (simp add: imageE phi-finite map-dom-mn-finite)

\textbf{have} $\phiEdges \subseteq \phiDomRan$

\textbf{apply} (rule subst[of $\forall \ a \in \phiEdges. \ a \in \phiDomRan$])

\textbf{apply} (simp-all add: subset-eq_subsetE phiArg-def)
by (auto simp: mn-def)
with (finite ?phiDomRan)
show finite ?phiEdges by (rule Finite-Set.rev-finite-subset)
qed

hence \( \bigwedge f \cdot \text{finite } (f \cdot (?phiEdges \cap (P \times P))) \) by auto

thus finite (condensation-edges g P) unfolding condensation-edges-def induced-phi-graph-def
by auto
qed

auxiliary lemmas for acyclicity

lemma condensation-nodes-edges: (condensation-edges g P) \subseteq (condensation-nodes g P \times condensation-nodes g P)

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

lemma condensation-edge-impl-path:
assumes \((a, b) \in (\text{condensation-edges } g P)\)
assumes \(\varphi_a \in a\)
assumes \(\varphi_b \in b\)
shows \((\varphi_a, \varphi_b) \in (\text{induced-phi-graph } g P)^+\)

unfolding condensation-edges-def

proof --
  from assms(1)
  obtain \(x\ y\) where \(x-y\)-props:
    \((x, y) \in (\text{induced-phi-graph } g P)\)
    \(a = \text{sec-of } (\text{induced-phi-graph } g P) x\)
    \(b = \text{sec-of } (\text{induced-phi-graph } g P) y\)

unfolding condensation-edges-def by auto

hence \(x \in a\ \ y \in b\) by auto

All that's left is to combine these paths.

with assms(2) \(x-y\)-props(2)
have \((\varphi_a, x) \in (\text{induced-phi-graph } g P)^+\) by (meson is-sec-connected sec-of-is-sec)

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-sec-connected sec-of-is-sec)

ultimately
show \((\varphi_a, \varphi_b) \in (\text{induced-phi-graph } g P)^+\) using \(x-y\)-props(1) by auto
qed

lemma path-in-condensation-impl-path:
assumes \((a, b) \in (\text{condensation-edges } g P)^+\)
assumes \(\varphi_a \in a\)
assumes \(\varphi_b \in b\)
shows \((\varphi_a, \varphi_b) \in (\text{induced-phi-graph } g P)^+\)

using assms

proof (induction arbitrary: \(\varphi_b\) rule:trans-induct)
  fix \(y\ z\ \varphi_b\)
  assume \((y, z) \in \text{condensation-edges } g P\)
hence is-sc c (induced-phi-graph g P) y unfolding condensation-edges-def by auto
hence \( \exists \phi_y. \phi_y \in y \) using sec-non-empty’ by auto
then obtain \( \phi_y \) where \( \phi_y \)-in-y: \( \phi_y \in y \) by auto

assume \( \phi_b \)-elem: \( \phi_b \in z \)
assume \( \bigwedge \phi_b. \phi_a \in a \implies \phi_b \in y \implies (\phi_a, \phi_b) \in (induced-phi-graph g P)^+ \)
with assms(2) \( \phi_y \)-in-y
have \( \phi_a \)-to-\( \phi_y \): \( (\phi_a, \phi_y) \in (induced-phi-graph g P)^+ \) using condensation-edge-impl-path by auto

from \( \phi_b \)-elem \( \phi_y \)-in-y \((y, z) \in condensation-edges g P\)
have newfl: \( (x, x) \notin (condensation-edges g P)^+ \) 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: \( (x, b) \in (condensation-edges g P) \) \( (b, x) \in (condensation-edges g P)^+ \)
by (meson converse-trancE)

hence \( x \in (condensation-nodes g P) b \in (condensation-nodes g P) \) using condensation-nodes-edges by auto
hence nodes-are-sec: is-sec (induced-phi-graph g P) x is-sec (induced-phi-graph g P) b
using sec-of-is-sec 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 \( \exists \phi_x. \phi_x \in x \ \exists \phi_b. \phi_b \in b \) using nodes-are-sec sec-non-empty’ ex-in-conv by auto
then obtain \( \phi_x \) \( \phi_b \) where \( \phi_x \)-elem: \( \phi_x \in x \ \phi_b \in b \) by metis
with nodes-are-sec(1) \( b \)-on-path path-in-condensation-impl-path condensation-edge-impl-path \( \phi_b \)-elem(2)
have \( \phi_b \in x \)
This however means $x$ and $b$ must be the same SCC, which is a contradiction to the nonreflexivity of $condensation\text{-}edges$.

by $−$ (rule $is\text{-}sec\text{-}closed$)

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

With $nonrefl$ show $False$ by simp

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

lemma $Ex\text{-}condensation\text{-}leaf$:

assumes $P ≠ \{\}$
shows $∃\ leaf.\ leaf ∈ (condensation\text{-}nodes g P) ∧ (∀\ sec.\ (leaf, sec) \notin \ condensation\text{-}edges g P)$

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

lemma $sec\text{-}in\text{-}P$:

assumes $sec ∈ condensation\text{-}nodes g P$
shows $sec ⊆ P$

proof $−$
have $sec ⊆ P$ if $y\text{-}props: sec = sec\text{-}of \ (induced\text{-}phi\text{-}graph g P)$ $n \in P$ for $n$

proof $−$
from $y\text{-}props$
show $sec ⊆ P$

proof (clar simp simp:y\text{-}props(1); case-tac $n = x$)
fix $x$
assume different: $n ≠ x$
assume $x ∈ sec\text{-}of \ (induced\text{-}phi\text{-}graph g P)$ $n$
hence \((n, x) \in (\text{induced-phi-graph } g P)^+\) by (metis is-scc-connected scc-of-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 rtranclE)

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 scc \(\subseteq\) condensation-nodes g P \(x \in\) scc

shows phi g \(x \neq\) None

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

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

Lemma 1. Every redundant set contains a redundant SCC.

lemma 1:

assumes redundant-set g P

shows \(\exists\) scc \(\subseteq\) P. redundant-scc g P scc

proof

from assms Ex-condensation-leaf[of P g]

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

unfolding redundant-set-def by auto

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

moreover

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

moreover

have leaf \(\subseteq\) dom (phi g)

apply (subst subset-eq, rule ballI)

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

moreover

from assms

obtain pred where pred-props: pred \(\in\) allVars g \(\forall\) \(\varphi\) \(\in\) P. \(\forall\) \(\varphi'\). phiAny g \(\varphi\) \(\varphi'\) \(\rightarrow\) \(\varphi'\) \(\in\) P \(\cup\) \{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** $\varphi'$. 

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

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

**proof** 

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

**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 sex-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** $\neg \text{redundant-set } g P$ **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 φ' and unnecessaryPhi g φ'

using assms

by (induction φ' rule: reachable.induct) auto

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

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

**lemma** preds-finite: finite (trueArgs g φ)

**proof** (rule contr)

assumes 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 (mn (phi g)) unfolding ran-def phiArg-def by clarsimp

Which cannot be.

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

qed

Any unnecessary 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 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: 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 phi-in-an)
  from empty assms(1)
  have no-preds: trueArgs g φ = {} by (subst card-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 \( \var{g} \var{\varphi} = \var{g} \var{n} \)

using assms

apply (induction rule: tranclp-induct)
  apply (rule phiArg-same-var[\text{symmetric}])
  apply simp
using phiArg-same-var by auto

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

**Lemma** phiArg-disjoint-paths-extend:

assumes \( \var{g} \var{r} = \var{V} \) and \( \var{g} \var{s} = \var{V} \) and \( \var{r} \in \text{allVars} \var{g} \) and \( \var{s} \in \text{allVars} \var{g} \) and \( \var{V} \in \text{oldDefs} \var{g} \var{n} \) and \( \var{V} \in \text{oldDefs} \var{g} \var{m} \)
and \( \var{g} \var{n} \rightarrow \text{defNode} \var{g} \var{r} \) and \( \var{g} \var{m} \rightarrow \text{defNode} \var{g} \var{s} \)
and \( \text{set} \var{n} \cap \text{set} \var{m} = \{\} \)
and \( \var{g} \var{\varphi} \var{r} \) obtains \( \var{n}' \) where \( \var{g} \var{n} \rightarrow \var{n}@[\text{defNode} \var{g} \var{r}] \)
and \( \text{set} (\text{butlast} (\var{n}@[\var{n}'])) \cap \text{set} \var{m} = \{\} \)
proof (cases \( \var{r} = \var{\varphi} \var{r} \))
  case True
  If the node to extend the path to is already the endpoint, the lemma is trivial.

  with assms(7,8,9) in-set-butlastD
  have \( \var{g} \var{n} \rightarrow \var{n}@[] \rightarrow \text{defNode} \var{g} \var{r} \) \( \text{set} (\text{butlast} (\var{n}@[]))) \cap \text{set} \var{m} = \{\} \)
    by simp-all fastforce
  with that show \(?thesis\).
  next
  case False

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

  from assms(10)
  have \( \var{\varphi} \var{r} \in \text{allVars} \var{g} \) unfolding phiArg-def
    by (metis allDefs-in-allVars phiDefs-in-allDefs phi-def phi-phiDefs phiDef phi-s-in-\alpha)
  with assms(10)
  obtain \( \var{rs}' \var{pred} \var{r} \) where \( \var{rs}' \var{props}: \var{g} \var{n} \rightarrow \text{defNode} \var{g} \var{r} \var{n} \rightarrow \text{defNode} \var{g} \var{r} \rightarrow \var{rs}' \rightarrow \var{pred} \var{\varphi} \var{r} \var{old} \var{Entry}\var{Path} \var{g} \var{rs}' \var{r} \in \text{phiUses} \var{g} \var{pred} \var{\varphi} \var{r} \var{pred} \var{\varphi} \var{r} \in \text{set} (\text{old.predecessors} \var{g} (\text{defNode} \var{g} \var{\varphi} \var{r}))) \)
    by (rule phiArg-path-ex')

define \( \var{rs} \) where \( \var{rs} = \var{rs}'@0[@\text{defNode} \var{g} \var{\varphi} \var{r}] \)
from \( \var{rs}' \var{props}(2,1) \) \( \var{old} \var{Entry}\var{Path}-\var{distinct} \var{old} \var{path}-2-\var{hd} \)
have \( \var{rs}' \var{loopfree}: \var{defNode} \var{g} \var{r} \notin \text{set} (\var{tl} \var{rs}') \) by (simp add: Misc.distinct-hd-tl)

from False assms have \( \var{defNode} \var{g} \var{\varphi} \var{r} \neq \var{defNode} \var{g} \var{r} \)
  apply
  apply (rule phiArg-distinct-nodes)
  apply (auto intro:phiArg-in-allVars)[2]
unfolding phiAr-g-def by (metis allDefs-in-allVars phiDefs-in-allDefs phi-def phi-phiDefs phi-in-CN)

from rs'-props have rs-props: g ⊢ defNode g r → 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-smoc old.path2-def rs'-props(1) rs-def rs'-loopfree (defNode g φ_r ≠ defNode g r) by auto

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

  show thesis
  proof (rule that)
    show set (butlast (ns @ tl rs)) ∩ set ms = {}
    proof (rule contr, simp only: ex-in-conv[symmetric])
      assume ∃ 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.buttlast-tld 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-buttlastD)
    by blast
  with assms(8) old.path2-prefix-ex
obtain $ms'$ where $ms'$-props: $g \vdash m \rightarrow ms' \rightarrow i$ prefix $ms'$ $ms \not\in \text{set (butlast ms')}$ 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 $\phi$ argument to the $\phi$ itself leads to a contradiction. However, we still need to distinguish the cases of whether $m = i$

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

thus thesis

proof (cases)
  case $ri-is-valid$
  $ri$ is a valid path extension.

  with assms(7) $ri$-props(1)
  have $g \vdash n - ns \circ (tl ri) \rightarrow \text{defNode } g \varphi_r$ by auto

moreover
  have $\text{set (butlast (ns \circ (tl ri)))} \cap \text{set ms} = \{\}$
  proof (rule contr)
    assume $\text{contr: set (butlast (ns \circ (tl ri)))} \cap \text{set ms} \neq \{\}$
    from this
    obtain $x$ where $x$-props: $x \in \text{set (butlast (ns \circ (tl ri)))} x \in \text{set ms}$ by auto
    with assms(9) have $x \notin \text{set ns}$ by auto
    with $x$-props ($g \vdash n - ns \circ (tl ri) \rightarrow \text{defNode } g \varphi_r \circ \text{defNode } g \varphi_r \neq \text{defNode } g$
    $r$) assms(7)
    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 $m$-same

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

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

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

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

moreover

12
from rs-props(1) m-i-same rs-def ri-prps(1,2,4) \( \text{defNode } g \varphi_r \neq \text{defNode } g r \) 
asms(7,9)

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

by (metis disjoint-elem hd-append in-hd-or-tl-conv in-prefix list.set(1) old.path2-hd old.path2-last old.path2-last-in-ns prefix-snoc)

moreover

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

unfolding oldDefs-def using \( \text{def}_m\text{-props } \text{asinz}_\text{ms} \) by blast

ultimately

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

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

have False by simp

thus \?thesis by simp

next

\textbf{case } m-i-difer

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

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

proof (rule old.pathConvergeI)

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

next

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

using ri-prps old.path2-nontriv asms(9) by (metis asms(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 \text{condr})

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

then obtain \( i' \) where \( i'\text{-props: } i' \in \text{set } (\text{butlast } ms') \)

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

\textbf{case } True

with \( i'\text{-props}(2) \)

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

proof (metis \text{Un-if} butlast-append in-set-butlastD set-append)

hence \( i' \in \text{set } (\text{butlast } ri) \) by (simp \text{add:in-set-ID } \text{List.butlast-ii})

with \( i'\text{-not-in-ms } \text{ri-prps}(3) \)

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

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

qed

qed (auto intro: \text{assms}(7) ri-prps(1) old.path2-app ms'\text{-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 necessaryPhi)

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 ri = tmp@[i]
apply (subgoal-tac ri ≠ [])
prefer 2
using ri-props(1) apply (simp add: old.path2-not-Nil)
apply (rule-tac that)
using append-butlast-last-id[symmetric] by auto
with rs'-rest-prop have rs'-rest-def: rs' = tmp@[i#rs'-rest] by auto
with rs'-props(1) have $g \vdash i - i#rs'-rest \Rightarrow \text{pred}_{\varphi_r}$
by (simp add: old.path2-split)
moreover
note (var $g \, r \, = \, V$) [simp]
from rs'-props(3) have $r \in \text{allUses}_g \, \text{pred}_{\varphi_r}$ unfolding allUses-def by simp

moreover
from 'defNode $g \, r \, \notin \, \text{set} \, (\text{tl} \, rs')$: rs'-rest-def have defNode $g \, r \, \notin \, \text{set} \, rs'$ by auto
with $(g \vdash i - i#rs'-rest \Rightarrow \text{pred}_{\varphi_r})$
have $\forall x. \, x \in \text{set} \, rs' \Rightarrow r \notin \, \text{allDefs} \, g \, x$
by (metis defNode-eq list.distinct(1) list.sel(3) list.set-cases old.path2-cases old.path2-in-cn)

moreover
from assms(7,8) $(g \vdash i - i#rs'-rest \Rightarrow \text{pred}_{\varphi_r})$ 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-ns)
ultimately

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

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

from $(g \vdash n \Rightarrow \text{defNode} \, g \, r)$ have defNode $g \, r \in \, \text{set} \, ns$ by auto

14
with \( \{ i \in \text{set } ms \} \) 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: \( \text{assms(3)} \))

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 \( \text{allDefs-def } \phi \text{Defs-def} \)

apply clarsimp

apply (erule_tac \( c=r \) in equalityCE)

using \( \phi \text{-def } \phi \text{is-phi by auto} \)

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

qed

lemma \( \text{reachable-same-var} : \)

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

shows \( \varphi = \varphi' \)

using \( \text{assms by } (\text{metis } \text{Nitpick} \text{.trancp-unfold } \phi \text{Arg-bundle-same-var } \text{reachable-prps(1)}) \)
Theorem 1. A graph which does not contain any redundant set is minimal according to Cytron et al.'s definition of minimality.

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

from eq \( \varphi_s \text{-in-allVars assms(3,4)} \)
have \( \varphi \) \( \varphi_s \neq \varphi g s \) 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.
\end{verbatim}
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:

- $r \neq s \varphi_r \in \text{reachable} g \varphi \varphi_s \in \text{reachable} g \varphi$
- $\neg \text{unecessaryPhi} g r \neg \text{unecessaryPhi} g s$
- $r \in \{n. \text{phiArg} g \} \varphi n \} s \in \{n. \text{phiArg} g \} \varphi n \}$
- $\text{phiArg} g\varphi r \text{phiArg} g \varphi s s$

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

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

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

moreover from assign-nodes-props this
have $r-s-in-trancp: (\text{phiArg} g) \varphi r (\text{phiArg} g) \varphi s$

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

from this
obtain $V$ where $V$-props: $\text{var g r = V var g s = V var g} \varphi = V$

by (metis phiArg-trancp-same-var)

moreover from $r-s-in-trancp$
have $r-s-allVars: \text r \in allVars g s \in allVars g$

by (metis phiArg-in-allVars tranclp-cases)+

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

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

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

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

proof (rule ccontr)
assume $n-m: n \neq m$

with $r-s-path$-props(2) old.path2-hd-in-ns
have $n \in set ns$ by blast

moreover from $n-m$ $r-s-path$-props(4) old.path2-hd-in-ns
have $n \in set ms$ by blast

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

qed

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

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

using phiArgs-disjoint-paths-extend by blast

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

have \( \text{defNode} \ g \ \varphi_r \ \notin \ \text{set} \ ms \)

proof (rule conbr)

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: \( g \vdash m - ms' \rightarrow \text{defNode} \ g \ \varphi_r \ \text{prefix} \ ms' \ ms \) by
\( -(\text{rule old.path2-prefix-ex [of} \ g \ m \ ms \ \text{defNode} \ g \ s \ \text{defNode} \ g \ \varphi_r], \ \text{auto}) \)

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

proof (rule old.pathsConvergeI)

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

proof (rule condr)

assume set (butlast (ns@rs)) \( \cap \ \text{set} (\text{butlast} ms') \neq \{\} \)

then obtain \( c\) where \( c\)-props: \( c \in \text{set} (\text{butlast} (ns@rs)) \ c \in \text{set} (\text{butlast} ms') \) 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 \( \text{False} \) by auto

qed

next

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

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 \( \text{assign-nodes-props(8)} \) \( \text{old.path2-in-con phiArgs-def phiArgs-same-var} \) \( \text{r}\)-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-\( \varphi_r\)-(difer \( rs\)-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 \( \text{assign-nodes-props(8)} \) phiArgs-same-var by auto

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

show \( \text{False} \) unfolding unnecessaryPhi-def by simp

qed

with \( rs\)-props

have aux: \( \text{set} \ ms \cap \text{set} (ns \ @ rs) = \{\} \)

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) \( \text{assign-nodes-props(8)} \) phiArgs-same-var by auto

18
have \( \varphi_r \in \text{allVars} \)
by \((\text{meson phiArg-def assign-nodes-props}(8) \text{ allDefs-in-allVars old.path2-tl-in-con phiDefs-in-allDefs phi-phiDefs rs-props})\)

from \( V\text{-props}(2) \varphi_r \cdot V \text{-allVars}(2) \varphi_r \cdot \text{allVars} \text{ r-s-path-props}(3) \text{ r-s-path-props}(1) \text{ r-s-path-props}(4) \text{ aux assign-nodes-props}(9)\)
obtain \( s \) where \( ss\text{-props}: g \vdash m - ms \otimes ss \rightarrow \text{defNode} g \varphi_s \) set \( \{ \text{butlast} (ms \otimes ss) \} \)
by \((\text{rule phiArg-disjoint-paths-extend}) \) (metis disjoint-iff-not-equal in-set-butlastD)
define \( p_m \) where \( p_m = ms \otimes ss \)
define \( p_n \) where \( p_n = ns \otimes rs \)

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

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

have \( \text{path-crossing-yields-convergence}: \)
\( \exists \varphi_z \in \text{reachable} g \varphi . \exists ns ms. \text{ old.pathConverge} g n ns m ms (\text{defNode} g \varphi_z) \)
if \( \varphi_r \in \text{reachable} g \varphi \) \( \text{and} \ \varphi_s \in \text{reachable} g \varphi \) \( g \vdash n - p_n \rightarrow \text{defNode} g \varphi_r \)
and \( g \vdash m - p_m \rightarrow \text{defNode} g \varphi_s \) \( \text{and} \) 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 \( \text{that}(6) \text{ split-list-first-propE} \)
obtain \( p_m I n_z p_m 2 ) \text{ where} \ n_z\text{-props}: n_z \in set p_n p_m = p_m I \oplus n_z \neq p_m 2 \forall n \in set p_m I \ n \notin set p_n \)
by \((\text{auto intro: split-list-first-propE})\)

with \( \text{that}(3,4) \)
obtain \( p_n' \) where \( p_n'\text{-props}: g \vdash n - p_n' \rightarrow n_z \) \( g \vdash m - p_m I @ [n_z] \rightarrow n_z \) \( \text{prefix} \)
\( p_n' p_n n_z \notin set \{ \text{butlast} p_n \} \)
by \((\text{meson old.path2-prefix-ex old.path2-split}(1))\)

from \( V\text{-props}(3) \text{ reachable-same-var}[OF that(1)] \text{ reachable-same-var}[OF that(2)]\)
have \( \text{phis-V: var} g \varphi_r = V \text{var} g \varphi_s = V \) by \( \text{simp-all} \)
from \( \text{reachable-props}(1) \) \( \text{that}(1,2) \) \( \varphi\text{-props}(2) \) \( \text{phiArg-in-allVars} \)
have \( \text{phis-allVars:} \varphi_r \in \text{allVars} g \varphi_s \in \text{allVars} g \) by \((\text{metis rtranclp.cases})\)

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

from assms \( n_z \)-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 (\( g \vdash n \) \( \neg p_n \Rightarrow \text{defNode } g \varphi_r \))

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

with that(2) show ?thesis.

next

case False

from \( n_z \)-props(2)

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

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

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

with that(1) show ?thesis.

qed

thus \( \exists \varphi_z \in \text{reachable } g \varphi \). \( \exists n s m s. \text{old.path} \cap \text{Converge } g n n s m m s (\text{defNode } g \varphi_z) \)

proof (cases)

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

have old.path \( \cap \text{Converge } g n p_n \text{'} m m (\text{defNode } g \varphi_s) \)

proof (rule old.path2-\cap \text{ConvergeI})

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

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

show \( I < \text{length } p_n \text{'} \) by auto

from that(4) show \( g \vdash m \) \( \neg p_m \Rightarrow \text{defNode } g \varphi_s \).

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

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

by (meson butlast-prefix disjoint1 disjoint-elem in-prefix)

qed

with that(1,2,3) show ?thesis by (auto intro:reachable.intro(2))

next

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

have old.path \( \cap \text{Converge } g n p_n \text{'} m (p_m \text{'} @ [n_z]) (\text{defNode } g \varphi_r) \)

proof (rule old.path2-\cap \text{ConvergeI})

from \( p_m \text{-ends-in-} p_n \) \( p_n \text{'} \text{-props}(1,2) \) show \( g \vdash n \) \( \neg p_n \text{'} \Rightarrow \text{defNode } g \varphi_r \) \( g \vdash m \) \( \neg p_m \text{'} \Rightarrow \text{defNode } g \varphi_r \) by simp-all

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

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

from \( \text{assign-nodes}\text{-props}(3,2) \text{ ind}\text{-props} \text{V\text{-props}(3) \ \varphi_r\text{-V \varphi_r\text{-allVars}}} \) have \( \exists \varphi_z \in \text{reachable} \ g \varphi. \exists \ ns \ ms. \ \text{old.pathConverge} \ g \ n \ ns \ m \ ms \ (\text{defNode} \ g \ \varphi_z) \)
proof (induction arbitrary: \( p_m \ p_n \) rule: reachable.induct)
case refl
In the induction basis, we know that \( \varphi = \varphi_z \) and a path to \( \varphi_r \) must be obtained — for this we need a second induction.

from \( \text{refl.prems refl.hyps show ?case}\) proof (induction arbitrary: \( p_m \ p_n \) rule: reachable.induct)
case refl
The first case, in which \( \varphi_r = \varphi_z = \varphi_\) is trivial — \( \varphi \) suffices.

have \( \text{old.pathConverge} \ g \ n \ p_n \ m \ p_m \ (\text{defNode} \ g \ \varphi) \)
proof (rule old.pathConverge1)
show \( 1 < \text{length} \ p_n, 1 < \text{length} \ p_m \)
using \( \text{refl} \text{V\text{-props} simpleDefs\text{-phiDefs-war-disjoint unfolding unnecessaryPhi-def by (metis domD domff old.path2-hd-in-\alpha old.path2-nontriv phi-phiDefs r-s-path\text{-props}(1) r-s-path\text{-props}(3))}+ \)
show \( g \vdash n\text{-p}\rightarrow \text{defNode} \ g \varphi \ g \vdash m\rightarrow \text{defNode} \ g \varphi \) \( \text{set} \ (\text{butlast} \ p_m) \)
\( \cap \text{set} \ (\text{butlast} \ p_n) = \{\} \)
using \( \text{refl by auto} \)
qed
with \( \varphi \in \text{reachable} \ g \varphi \) show ?case by auto
next
case (step \( \varphi' \ \varphi_r \))
In this case we have that \( \varphi = \varphi_z \) and need to acquire a path going to \( \varphi_r \), however with the aux. lemma we have, we still need that \( p_n \) and \( p_m \) are disjoint.

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

next
case True

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

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

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

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

next

\begin{enumerate}
\item case (step \varphi' \varphi_s)
\end{enumerate}

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

\begin{enumerate}
\item show \vDash \text{thesis}
\item proof (cases set p_m \cap set p_n = \{\})
\item case True
\item have \varphi_s \cdot V: \text{ var } g \varphi_s = V \text{ using step(1,2,3,9) } \text{ reachable-same-var by (simp add: phiArg-same-var)}
from \text{ step(2) } \text{ have } \varphi_s \cdot \text{allVars: } \varphi_s \in \text{allVars } g \text{ by (rule phiArg-in-allVars)}
\item obtain p_m' where \text{ tmp: } g \vdash m - p_{m}@p_{m}' \rightarrow \text{defNode } g \varphi' \text{ set } \text{(butlast } (p_{m}@p_{m}')) \cap set (\text{butlast } p_n) = \{\}
by (rule phiArg-disjoint-paths-extend[of g \varphi_s \text{ V } \varphi_r m n p_m p_n \varphi'])
\item (metis \varphi_s \cdot \text{V-props } V \cdot \text{allVars step } r-s-path-props(1,3) \text{ True disjoint-iff-not-equal in-set-butlastD)}+
\item from \text{ step(5) this(1) step(7) this(2) step(9) step(10) step(11) show } \vDash \text{thesis by (rule step.IH[of p_m@p_{m}' \cdot p_n])}
\end{enumerate}

next

\begin{enumerate}
\item case paths-cross: False
\item with \text{ step reachable.intros}
\item show \vDash \text{thesis using } \text{path-crossing-yields-convergence[of } \varphi_r \varphi_s p_n p_m\] by blast}
\item 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 $\varphi_z = V$ by (metis V-props(3) phiArg-trancl-same-var rtranclpD)

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

ultimately show False unfolding unnecessaryPhi-def by blast

qed

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

corollary no-redundant-SCC-minimal:

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

shows cytronMinimal $g$

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

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

corollary

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

shows $\neg \text{ redundant } g$

proof –

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

proof –

assume redundant $g$

then obtain $\varphi$ where phi $g$ $\varphi$ $\neq$ 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

