Minimal Static Single Assignment Form

Max Wagner         Denis Lohner

January 19, 2017

Abstract

This formalization is an extension to [3]. In their work, the authors have shown that Braun et al.'s static single assignment (SSA) construction algorithm [1] produces minimal SSA form for input programs with a reducible control flow graph (CFG). However Braun et al. also proposed an extension to their algorithm that they claim produces minimal SSA form even for irreducible CFGs. In this formalization we support that claim by giving a mechanized proof.

As the extension of Braun et al.'s algorithm aims for removing so-called redundant strongly connected components (sccs) of \( \phi \) functions, we show that this suffices to guarantee minimality according to Cytron et al. [2].

Contents

1 Minimality under Irreducible Control Flow 1
   1.1 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 \( \langle v \rangle \) outside \( P \). A redundant SCC is a redundant set that is strongly connected according to the \textit{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 $g P \equiv \{ (\phi, \phi'), \phiArg g \phi \phi' \} \cap P \times P$
```

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

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

```
definition redundant-scc $g P \text{ scc} \equiv \text{redundant-set g scc} \land \text{is-scc (induced-phi-graph g P) scc}$
```

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

```
definition condensation-nodes $g P \equiv \text{scc-of (induced-phi-graph g P) ' P}$
```

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

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 ?phiEdges={(a,b). \phiArg g a b}
  have finite ?phiEdges
  proof
    let ?phiDomRan=(\text{dom (phi g)} \times (\text{ran (phi g)}))
    from phi-finite
    have finite ?phiDomRan by (simp add: imageE finiteE map-dom-ran-finite)
    have ?phiEdges \subseteq ?phiDomRan
      apply (rule subst[of \forall a \in ?phiEdges. a \in ?phiDomRan])
      apply (simp-all add: subsetE[symmetric] phiArg-def)
```

by (auto simp: ran-def)
with (finite ?phiDomRan)
show finite ?phiEdges by (rule Finite-Set.rev-finite-subset)
qed
hence \( \bigwedge f. \text{finite } (f \cdot (\text{phiEdges} \cap (P \times P))) \) by auto
thus finite (condensation-edges g P) unfolding condensation-edges-def induced-phi-graph-def by auto
qed

auxiliary lemmas for acyclicity

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

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

  All that’s left is to combine these paths.
with assms(2) \(x-y\)-props(2)
have \((\varphi_a, x) \in \text{induced-phi-graph g P}^*\) by (meson is-scc-connected scc-of-is-scc)
moreover with assms(3) \(x-y\)-props(3) \(\forall y \in b\):
  have \((y, \varphi_b) \in \text{induced-phi-graph g P}^*\) by (meson is-scc-connected scc-of-is-scc)
ultimately
show \((\varphi_a, \varphi_b) \in \text{induced-phi-graph g P}^*\) using \(x-y\)-props(1) by auto
qed

lemma path-in-condensation-impl-path:
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:trancl-induct)
  fix \(y, z\) \(\varphi_b\)
  assume \((y, z) \in \text{condensation-edges g P}\)
hence \( \text{is-scc} \ (\text{induced-phi-graph } g \ P) \ \text{y unfolding} \ \text{condensation-edges-def} \) by auto
hence \( \exists \varphi_y. \varphi_y \in y \) using \( \text{scc-non-empty'} \) by auto
then obtain \( \varphi_y \) where \( \varphi_y \text{-in-y: } \varphi_y \in y \) by auto

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

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

lemma \( \text{condensation-acyclic: } \text{acyclic} \ (\text{condensation-edges } g \ P) \)
proof (rule acyclicI, rule allI, rule ccontr, simp)
fix \( x \)
Assume there is a cycle in the condensation graph.
assume \( \text{cyclic: } (x, x) \in (\text{condensation-edges } g \ P)^+ \)
have \( \land \varphi_c, \varphi_a \in a \implies \varphi_a \in y \implies (\varphi_a, \varphi_b) \in (\text{induced-phi-graph } g \ P)^+ \)
by auto
Then there must be a second SCC \( b \) on this path.
from \( \text{this cyclic} \)
obtain \( b \) where \( b \text{-on-path: } (x, b) \in (\text{condensation-edges } g \ P)^+ \) \( (b, x) \in (\text{condensation-edges } g \ P)^+ \)
by (meson converse-tranclE)

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

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

have \( \exists \varphi_x. \varphi_x \in x \ \exists \varphi_b. \varphi_b \in b \) using \( \text{nodes-are-scc } \text{scc-non-empty'} \text{ ex-in-conv} \)
by auto
then obtain \( \varphi_x \varphi_b \) where \( \varphi_x \text{-elem: } \varphi_x \in x \ \varphi_b \in b \) by metis
with \( \text{nodes-are-scc}(1) \) \( b \text{-on-path } \text{path-in-condensation-impl-path} \)
\( \text{condensation-edge-impl-path} \)
\( \varphi_x \text{b-elem}(2) \)
have \( \varphi_b \in x \)
This however means \( x \) and \( b \) must be the same SCC, which is a contradiction to the nonreflexivity of condensation-edges.

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

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

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

lemma scc-in-P:
assumes \( \text{scc} \in \text{condensation-nodes} \ g P \)
shows \( \text{scc} \subseteq \ P \)
proof –
have \( \text{scc} \subseteq \ P \) if \( \text{y-props: scc = scc-of (induced-phi-graph} \ g P) \) \( n \in \ P \) for \( n \)
proof –
from y-props
show \( \text{scc} \subseteq \ P \)
proof (clarsimp simp:y-props(1); case-tac \( n = x \))
fix \( \text{x} \)
assume different: \( n \neq x \)
assume \( x \in \text{scc-of (induced-phi-graph} \ g P) \) \( n \)
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 \(\text{step}: (z, x) \in (\text{induced-phi-graph } g P)\) by (meson tranclE)
from \(\text{step}\)
show \(x \in P\)
unfolding induced-phi-graph-def by auto
qed simp
qed 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 \(scc \in \text{condensation-nodes } g P\) \(x \in \text{scc}\)
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.

lemma 1:
assumes redundant-set g P
shows \(\exists \text{scc } \subseteq P.\ \text{redundant-scc } g P\ \text{scc}\)
proof –
from assms Ex-condensation-leaf[of \(P\) \(g\)]
obtain \(\text{leaf}\) where \(\text{leaf-props}: \text{leaf} \in (\text{condensation-nodes } g P)\ \forall \text{scc}.\ (\text{leaf}, \text{scc}) \notin \text{condensation-edges } g P\)
unfolding redundant-set-def by auto
hence is-scc (induced-phi-graph g P) leaf unfolding condensation-nodes-def by auto
moreover
hence leaf \(\neq \{\}\) by (rule scc-non-empty')
moreover
have leaf \(\subseteq \text{dom } (\phi g)\)
apply (subst subset-eq, rule ballI)
using redundant-scc-phis leaf-props(1) assms(1) by auto
moreover
from assms
obtain \(\text{pred}\) where \(\text{pred-props}: \text{pred} \in \text{allVars } g\ \forall \varphi \in P.\ \forall \varphi'.\ \phi \text{Arg } g \varphi \varphi' \rightarrow \varphi' \in P \cup \{\text{pred}\}\)
unfolding redundant-set-def by auto

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

fix $\varphi \varphi'$

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

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

proof cases

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

moreover

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

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

ultimately

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

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

scc-of-is-scc)

with leaf-props(2)

show ?thesis by auto

next

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

with that leaf-props pred-props

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

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

with assms

show ?thesis by auto

qed auto — the other cases are trivial

}

with pred-props(1)

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

ultimately

have redundant-scc $g \ P$ leaf unfolding redundant-scc-def redundant-set-def by auto

thus ?thesis using leaf-props(1) scc-in-$P$ by meson

qed

1.2 Proof of Minimality

We inductively define the reachable-set of a $\phi$ function as all $\phi$ functions reachable from a given node via an unbroken chain of $\phi$ argument edges to unnecessary $\phi$ functions.
inductive-set reachable :: 'g ⇒ 'val ⇒ 'val set
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 ccontr)
  assume infinite (trueArgs g ϕ)
  hence a: infinite {ϕ'. ∃ ϕ'' ∈ reachable g ϕ. phiArg g ϕ'' ϕ'} by auto
  have phiArg-set: {ϕ'. ∃ ϕ. phiArg g ϕ ϕ'} = ∪ (set ' {b. ∃ a. phi g a = Some b})
  unfolding phiArg-def by auto

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

  have infinite {ϕ'. ∃ ϕ. phiArg g ϕ ϕ'}
    by (rule infinite-super[of {ϕ'. ∃ ϕ'' ∈ reachable g ϕ. phiArg g ϕ'' ϕ'}]) (auto simp: a)
  with phiArg-set
  have infinite (ran (phi g)) unfolding ran-def phiArg-def by clarsimp

  Which cannot be.

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

Any unnecessary ϕ with less than 2 true arguments induces with reachable g ϕ a redundant set itself.

lemma few-preds-redundant:
assumes card (trueArgs g ϕ) < 2 unnecessaryPhi g ϕ
shows redundant-set g (reachable g ϕ)
unfolding redundant-set-def
proof (intro conjI)
  from assms
  show reachable g ϕ ≠ {} using empty-iff reachable.intros(1) by auto
next
  from assms(2)
show reachable g 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 force
  hence pred ∈ allVars g by (auto intro: Int-Coll 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)
  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′′
    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

def 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 φ r
obtains ns’
where g ⊢ n−ns@ns’→defNode g φ r
and set (butlast (ns@ns’)) ∩ set ms = {}
proof (cases r = φ r)
  case (True)
If the node to extend the path to is already the endpoint, the lemma is trivial.

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

    from assms(10)
    have φ r ∈ allVars g unfolding phiArg-def
      by (metis allDefs-in-allVars phi-def phi-phiDefs phis-in-αn)
    with assms(10)
    obtain rs’ pred φ r where rs’-props: g ⊢ defNode g r→rs’→pred φ r old.EntryPath
      g rs’ r ∈ phiUses g pred φ r pred φ r ∈ set (old.predecessors g (defNode g φ r))
      by (rule phiArg-path-ex’)

    def rs≡rs@[defNode g φ r]
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 φ r ≠ defNode g r
  apply
    apply (rule phiArg-distinct-nodes)
      apply (auto intro:phiArg-in-allVars)[2]
unfolding $\phi$Arg-def by (metis allDefs-in-allVars phiDefs-in-allDefs phi-phiDefs phi-phiDefs phi-in-cn)

from $rs'\text{-props}$
have $rs\text{-props}$: $g \vdash \text{defNode } g \rightarrow \text{defNode } g \varphi_r$  length $rs > 1$  defNode $g \forall r 
set (tl rs)$
  apply (subgoal-tac defNode $g \rightarrow \text{hd } rs'$)
  prefer 2 using $rs'\text{-props}(1)$
  apply (rule old.path2-hd)
  using old.path2-snoc old.path2-def $rs'\text{-props}(1)$  rs-def $rs'\text{-loopfree} (\text{defNode } g \varphi_r \neq \text{defNode } g r) \text{ by auto}$

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

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

We proceed by case distinction:

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

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

thus thesis
proof (cases)
case \( ri-is-valid \)

\( ri \) is a valid path extension.

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

moreover
have \( \text{set (butlast (ns@(tl ri))) } \cap \text{ set } ms = \{ \} \)
proof (rule contr)
assume \( \text{contr: set (butlast (ns @ tl ri)) } \cap \text{ set } ms \neq \{ \} \)
from this
obtain \( x \) where \( x \) props: \( x \in \text{ set (butlast (ns @ tl ri)) } \) \( x \in \text{ set } ms \) by auto
with \( \text{assms(9) have } x \notin \text{ set } ns \) by auto
with \( x \) props \( \langle g \vdash n - ns@tl ri \rightarrow \text{defNode } g \varphi_r \rangle \langle \text{defNode } g \varphi_r \neq \text{defNode } g \rangle \)
\( r \) it uses \( \text{assms(7)} \)
have \( x \in \text{ set (butlast (tl ri)) } \)
by (metis Un-iff append-Nil2 butlast-append old.path2-last set-append)
with \( x \) props \( \langle \text{defNode } g \varphi_r \rangle \langle \text{defNode } g \varphi_r \neq \text{defNode } g \rangle \)
show \( False \) by (metis FormalSSA-Misc.in-set-tlD List.butlast-tl)
qed
ultimately
show thesis by (rule that)
next
case \( m-i-same \)

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

note \( rs'\)-props(1) \( rs'\)-loopfree
moreover have \( r \in \text{allDefs} g \) \( \text{defNode } g \varphi_r \) by (simp add: \( \text{assms(3)} \))
moreover from \( rs'\)-props(3) have \( r \in \text{allUses} g \varphi_{\text{pred } r} \) unfolding \( \text{allUses-def} \) by simp

moreover
from rs-props(1) m-i-same rs-def ri-props(1,2,4) :defNode g φr ≠ defNode g r) assms(7,9)
have m ∈ set (tl rs')
by (metis disjoint-elem hd-append in-hd-or-tl-conv in-prefix list.sel(1) old.path2-hd old.path2-last old.path2-last-in-ns prefix-snoc)

moreover
from assms(6) obtain def_m where def_m ∈ allDefs g m var g def_m = V
unfolding oldDefs-def using defs-in-allDefs by blast
ultimately
have var g def_m ≠ var g r
by (rule conventional, simp-all)
with ⟨var g def_m = V⟩ assms(1)
have False by simp
thus ?thesis by simp

next
case m-i-differ
If m ≠ i, i constitutes a proper path convergence point.

have old.pathsConverge g m ms' n (ns @ tl ri) i
proof (rule old.pathsConvergeI)
show 1 < length ms' using m-i-differ ms'-props old.path2-nontriv by blast
next
show 1 < length (ns @ tl ri)
using ri-props old.path2-nontriv assms(9) by (metis assms(7) disjoint-elem old.path2-app old.path2-hd-in-ns)
next
show set (butlast ms') ∩ set (butlast (ns @ tl ri)) = {}
proof (rule ccontr)
assume set (butlast ms') ∩ set (butlast (ns @ tl ri)) ≠ {}
then obtain i' where i'-props: i' ∈ set (butlast ms') i' ∈ set (butlast (ns @ tl ri))
by auto
with ms'-props(2)
have i'-not-in-ms: i' ∈ set (butlast ms) by (metis in-set-butlast-appendI)
hence i' ∈ set (butlast ri)
with i'-not-in-ms ri-props(3)
show False by (auto dest: in-set-butlastD)
qed (meson disjoint-elem in-set-butlastD)
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-ri'prefix: prefix ri rs' by fastforce
then obtain rs'-rest where rs'-rest-prop: rs' = ri@rs'-rest using prefixE
by auto
from old.path2-last[OF ri-props(1)] last-snoc[of - i] obtain tmp where

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 auto

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 \textit{allDefs-disjoint} property.

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

unfolding \textit{allDefs-def phiDefs-def}

apply clarsimp

apply (erule-tac c=r in equalityCE)

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

thus \(?thesis by simp

qed

lemma \textit{reachable-same-var}:

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

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

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

lemma \textit{phi-node-no-defs}:

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

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

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

unfolding \textit{unnecessaryPhi-def} by auto

lemma \textit{defNode-differ-aux}:

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

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

proof (rule ccontr)

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 \textit{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 (erule reachable-props)

unfolding \textit{reachable-def by (meson IntD1 mem-Collect-eq rtranclpD)}
have \( \varphi_s \in \text{allVars} \) 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 \( \varphi_s \in \text{allVars} \) by (rule phiArg-in-allVars)
next
  case eq: True
  with assms (2)
  show \( \varphi_s \in \text{allVars} \) by (subst eq [symmetric])
qed

from eq \( \varphi_s \in \text{allVars} \) assms (3, 4)
have \( \text{var } g \varphi_s \neq \text{var } g s \) 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.

**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 \( \varphi \) function which does not sit at the convergence point of multiple liveness intervals.
  then obtain \( \varphi \) where \( \varphi \)-props: unnecessaryPhi g \( \varphi \in \text{allVars} \) \( \varphi \in \text{reachable } g \varphi \)
  using cytronMinimal-def unnecessaryPhi-def reachable-def unnecessaryPhi-def reachable.intros by auto
  We consider the reachable-set of \( \varphi \). If \( \varphi \) has less than two true arguments, we know it to be a redundant set, a contradiction. Otherwise, we know there to be at least two paths from different definitions leading into the reachable-set of \( \varphi \).
  consider (nontrivial) card (trueArgs g \( \varphi \)) \( \geq 2 \) | (trivial) card (trueArgs g \( \varphi \)) \( < 2 \)
  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 \( \varphi \)-props (1)
  have redundant-set g (reachable g \( \varphi \)) 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** $r \ s \ \phi_r, \ \phi_s$ **where** assign-nodes-props:

- $r \neq s \ \varphi_r, \ \varphi_s \in \text{reachable} \ g \ \varphi_1, \ \varphi_3 \in \text{reachable} \ g \ \varphi$
- $\neg \text{unnecessaryPhi} \ g \ r \ \neg \text{unnecessaryPhi} \ g \ s$
- $r \in \{n. (\text{phiArg} \ g)^{**} \ \varphi \ n\} \ s \in \{n. (\text{phiArg} \ g)^{**} \ \varphi \ n\}$
- $\text{phiArg} \ g \ \varphi_r \ \text{phiArg} \ g \ \varphi_s$
- 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-rtrancl)

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

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

- have $n-m$-distinct: $n \neq m$
- **proof** (rule ccontr)
- assume $n-m: \neg n \neq m$
- with $r-s-path$-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: \( g \vdash n \rightarrow \text{defNode} \ g \ \varphi_w \) \( \setminus \text{set} \ ms = \{\} \)

using phiArg-disjoint-paths-extend by blast

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

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

proof (rule ccontr)

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

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

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

\(-(\text{rule \ old.path2-pref \ ex} [of \ g \ m \ ms \ \text{defNode} \ g \ s \ \text{defNode} \ g \ \varphi_w], \text{auto})\)

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

proof (rule old.pathsConvergeI)

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

proof (rule ccontr)

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

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

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-\( \varphi_w\)-differ: \( n \neq \text{defNode} \ g \ \varphi_w, \ m \neq \text{defNode} \ g \ \varphi_w \)

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

apply fastforce

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

by auto

with \( ms'\)-props(1)

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

from \( m\)-n-\( \varphi_w\)-differ \( rs\)-props(1)

show \( 1 < \text{length} (ns @ rs) \) using \( \text{old.path2-nontriv} \) by blast

qed (auto intro: \( rs\)-props set-mono-prefix \( ms'\)-props)

with \( V\)-props \( r\)-s-path-props

have \( \text{necessaryPhi} \ g \ \varphi_w, \ \text{unfolding necessaryPhi-def} \) using assign-nodes-props(8) \( phiArg-same-var \) by auto

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

show False using unnecessaryPhi-def by simp

qed

with \( rs\)-props

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

by (metis disjoint-iff-not-equal not-in-butlast old.path2-last)

have \( \varphi_w\)-V: \( \var g \ \varphi_w = V \)

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

from V-props(2) $\varphi_r$-V r-s-allVars(2) $\varphi_r$-allVars r-s-path-props(3) r-s-path-props(1)
  r-s-path-props(4) rs-props(1) aux assign-nodes-props(9)
obtain ss where ss-props: $g \vdash m - ms@ss \rightarrow \text{defNode} g \varphi_s$ set (butlast (ms@ss))
  intersect, the first such intersection point must be a $\phi$ function in reachable $g \varphi$, yielding the path convergence we seek.

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

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

proof --
from that(6) split-list-first-propE
obtain $p_n.1 \ n_z \ p_m.2$ where $n_z$-props: $n_z \in \set p_n \ p_m = p_m.1 \ \& \ n_z \neq p_m.2$
  $\forall n \in \set p_n.1. n \notin \set p_n$
  by (auto intro: split-list-first-propE)

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

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-allVars(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 ns-m-props
consider (\(p_n\)-ends-in-\(p_m\)) \(n_z = \text{defNode} \ g \ \varphi_s\) | (\(p_m\)-ends-in-\(p_n\)) \(n_z = \text{defNode} \ g \ \varphi_r\)

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

qed

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

proof (cases)
  case \(p_n\)-ends-in-\(p_m\)
  have \(\text{old.pathsConverge} \ g \ n \ p_n' \ m \ p_m\) (defNode \ g \ \varphi_s)
  proof (rule old.pathsConverge1)
    from \(p_n\)-ends-in-\(p_m\) \(p_n'\)-props(1) show \(g \vdash n - p_n' \rightarrow \text{defNode} \ g \ \varphi_s\) by simp
  from \((n \neq \text{defNode} \ g \ \varphi_s) \ p_n\)-ends-in-\(p_m\) \(p_n'\)-props(1) old.path2-nontriv
  show \(I < \text{length} \ p_n'\) by auto
  from that(4) show \(g \vdash m - p_m \rightarrow \text{defNode} \ g \ \varphi_s\).
  with \((m \neq \text{defNode} \ g \ \varphi_s) \ old.path2-nontriv\) show \(I < \text{length} \ p_m\) by simp
  from that \(p_n'\)-props(3) show set (butlast \(p_n'\)) \cap set (butlast \(p_m\)) = \{\} by
  (meson butlast-prefix disjointI disjoint-elem in-prefix)

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

next
  case \(p_m\)-ends-in-\(p_n\)
  have \(\text{old.pathsConverge} \ g \ n \ p_n' \ m \ (p_m \ ![n_z] )\) (defNode \ g \ \varphi_r)
  proof (rule old.pathsConverge1)
    from \(p_m\)-ends-in-\(p_n\) \(p_n'\)-props(1,2) show \(g \vdash n - p_n' \rightarrow \text{defNode} \ g \ \varphi_r\) \(g \vdash m - p_m \ ![n_z] \rightarrow \text{defNode} \ g \ \varphi_r\) by simp-all

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

from assign-nodes-props(3,2) \( \) ind-props \( \) V-props(3) \( \) \( \varphi_r \)-V \( \) \( \varphi_r \)-allVars
have \( \exists \varphi_z \in \text{reachable } g \varphi \) \( \exists \) \( \) \( \) ns \( m \). \( \) \( \) old.pathsConverge \( g \) \( n \) \( n \) \( m \) \( m \) \( \) (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 refl.prems refl.\text{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 \( \) \( \) old.pathsConverge \( g \) \( n \) \( n \) \( m \) \( m \) \( \) (defNode \( g \) \( \varphi \))
proof \( \) (rule old.pathsConvergeI)

show \( 1 < \text{length } p_n \) \( 1 < \text{length } p_m \)
using refl V-vars simpleDefs-phiDefs-var-disjoint unfolding unnecessaryPhi-def by \( \) (metis domD domIff old.path2-hd-in-\( \alpha \) old.path2-nontriv phi-phiDefs r-s-path-vars(1) r-s-path-vars(3))

show \( g \vdash n \to \text{defNode } g \varphi \) \( g \vdash m \to \text{defNode } g \varphi \) \( \) set \( \) (butlast \( p_n \) )
using refl by auto
qed
with \( \varphi \in \text{reachable } g \varphi \) \) show ?case \( \) by \( \) auto

next
case (step \( \varphi' \) \( \varphi_r \))

In this case we have that \( \varphi = \varphi_z \) 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 \( p_n \) \( \) \( \) \( \) \( \) \( \) set \( p_m \) = \( \) \{\}\)

case paths-cross: False
with step reachable.intros

show ?thesis \( \) using path-crossing-yields-convergence[of \( \varphi_r \) \( \varphi \) \( p_n \) \( p_m \)] by \( \) (metis disjointI disjoint-elem)

next

21
case True

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

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

step(2)

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

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

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

moreover
have \( \varphi' \in \text{allVars } g \) by (metis \varphi'-props(2) phiArg-in-allVars reachable.cases step.hyps(1))

ultimately
show \( \exists \varphi_2 \in \text{reachable } g. \exists ns \text{ ms. old.pathsConverge } g n ns m ms (\text{defNode } g \varphi_2) \)

using step.prems(1) \( \varphi\)-props V-props \( g \vdash n \rightarrow p_n \@ p_m \rightarrow \text{defNode } g \varphi' \) by \(-(\text{rule step.IH; blast})\)

qed

next

next case (step \( \varphi' \varphi_s) \)

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

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

case True

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

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

obtain \( p_m \) \( \cdot \text{where}\) tmp: \( g \vdash m \rightarrow p_m \@ p_m' \rightarrow \text{defNode } g \varphi' \) set (butlast \((p_m \@ p_m')\)) \( \cap \) set \((\text{butlast } p_n)\) = {} 
by (rule phiArg-disjoint-paths-extend[of \( g \varphi_s V \varphi_r m n p_m p_n \varphi_2)]

(metis \varphi_s \cdot V \varphi_s \cdot \text{allVars step } r-s-path-props(1,3) \text{True disjoint-iff-not-equal in-set-butlastD})+

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

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

next

next case paths-cross: False

with step reachable.intros

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

qed
qed

then obtain $\varphi_z$ ns $ms$ where $\varphi_z \in$ 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-traclsame-var
rtranclpD)

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

unfolding necessaryPhi-def by blast

moreover with $\langle \varphi_z \in$ reachable $g$ $\varphi \rangle$ have unnecessaryPhi $g$ $\varphi_z$ by ¬(rule
reachable-props)

ultimately show False unfolding unnecessaryPhi-def by blast
qed
qed
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 $\neg(\exists P$ scc. redundant-scc $g$ $P$ scc)
shows cytronMinimal $g$
using assms 1 no-redundant-set-minimal by blast

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

corollary
assumes $\neg(\exists P$. redundant-set $g$ $P$)
shows $\neg$redundant $g$
proof –
have redundant $g$ $\Rightarrow \exists P$. redundant-set $g$ $P$
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

