

# Proving a data flow analysis algorithm for computing dominators

Nan Jiang

September 13, 2023

## Abstract

This entry formalises a fast iterative algorithm for computing dominators [1]. It gives a specification of computing dominators on a control flow graph where each node refers to its reverse post order number. A semilattice of reversed-ordered list which represents dominators is built and a Kildall's algorithm on the semilattice is defined for computing dominators. Finally the soundness and completeness of the algorithm are proved w.r.t. the specification.

## Contents

<b>1</b>	<b>The specification of computing dominators</b>	<b>1</b>
<b>2</b>	<b>More auxiliary lemmas for Lists Sorted wrt <math>&lt;</math></b>	<b>10</b>
<b>3</b>	<b>Operations on sorted lists</b>	<b>11</b>
<b>4</b>	<b>A semilattice of reversed-ordered list</b>	<b>12</b>
<b>5</b>	<b>A kildall's algorithm for computing dominators</b>	<b>14</b>
<b>6</b>	<b>Properties of the kildall's algorithm on the semilattice</b>	<b>16</b>
<b>7</b>	<b>Soundness and completeness</b>	<b>19</b>

## 1 The specification of computing dominators

```
theory Cfg
imports Main
begin
```

The specification of computing dominators is defined. For fast data flow analysis presented by CHK [1], a directed graph with explicit node list and

sets of initial nodes is defined. Each node refers to its rPO (reverse PostOrder) number w.r.t a DFST, and related properties as assumptions are handled using a locale.

**type-synonym**  $'a$  *digraph* =  $('a \times 'a)$  *set*

**record**  $'a$  *graph-rec* =

$g\text{-}V :: 'a$  *list*  
 $g\text{-}V0 :: 'a$  *set*  
 $g\text{-}E :: 'a$  *digraph*

$tail :: 'a \times 'a \Rightarrow 'a$   
 $head :: 'a \times 'a \Rightarrow 'a$

**definition** *wf-cfg* ::  $'a$  *graph-rec*  $\Rightarrow$  *bool* **where**

$wf\text{-}cfg\ G \equiv g\text{-}V0\ G \subseteq set( g\text{-}V\ G)$

**type-synonym** *node* = *nat*

**locale** *cfg-doms* =

— Nodes are rPO numbers

**fixes**  $G :: nat$  *graph-rec* (**structure**)

— General properties

**assumes** *wf-cfg*:  $wf\text{-}cfg\ G$

**assumes** *tail[simp]*:  $e = (u,v) \Longrightarrow tail\ G\ e = u$

**assumes** *head[simp]*:  $e = (u,v) \Longrightarrow head\ G\ e = v$

**assumes** *tail-in-verts[simp]*:  $e \in g\text{-}E\ G \Longrightarrow tail\ G\ e \in set( g\text{-}V\ G)$

**assumes** *head-in-verts[simp]*:  $e \in g\text{-}E\ G \Longrightarrow head\ G\ e \in set( g\text{-}V\ G)$

— Properties of a cfg where nodes are rPO numbers

**assumes** *entry0*:  $g\text{-}V0\ G = \{0\}$

**assumes** *dfst*:  $\forall v \in set( g\text{-}V\ G) - \{0\}. \exists prev. (prev, v) \in g\text{-}E\ G \wedge prev < v$

**assumes** *reachable*:  $\forall v \in set( g\text{-}V\ G). v \in (g\text{-}E\ G)^* \text{ `` } \{0\}$

**assumes** *verts*:  $g\text{-}V\ G = [0 ..< (length( g\text{-}V\ G))]$

— It is required that the entry node has an immediate successor which is not itself; Otherwise, no need to compute dominators It is required for proving the lemma: "wf\_dom start (unstables r step start)".

**assumes** *succ-of-entry0*:  $\exists s. (0,s) \in g\text{-}E\ G \wedge s \neq 0$

**begin**

**inductive** *path-entry* ::  $nat$  *digraph*  $\Rightarrow$   $nat$  *list*  $\Rightarrow$   $nat$   $\Rightarrow$  *bool* **for**  $E$  **where**

*path-entry0*:  $path\text{-}entry\ E\ []\ 0$

| *path-entry-prepend*:  $[(u,v) \in E; path\text{-}entry\ E\ l\ u] \Longrightarrow path\text{-}entry\ E\ (u\#\ l)\ v$

**lemma** *path-entry0-empty-conv*:  $path\text{-}entry\ E\ []\ v \longleftrightarrow v = 0$

*<proof>*

**inductive-cases** *path-entry-uncons*:  $\text{path-entry } E (u\#l) w$   
**inductive-simps** *path-entry-cons-conv*:  $\text{path-entry } E (u\#l) w$

**lemma** *single-path-entry*:  $\text{path-entry } E [p] w \implies p = 0$   
 ⟨*proof*⟩

**lemma** *path-entry-append*:  
 $\llbracket \text{path-entry } E l v; (v,w) \in E \rrbracket \implies \text{path-entry } E (v\#l) w$   
 ⟨*proof*⟩

**lemma** *entry-rtranc1-is-path*:  
**assumes**  $(0,v) \in E^*$   
**obtains**  $p$  **where**  $\text{path-entry } E p v$   
 ⟨*proof*⟩

**lemma** *path-entry-is-tranc1*:  
**assumes**  $\text{path-entry } E l v$   
**and**  $l \neq []$   
**shows**  $(0,v) \in E^+$   
 ⟨*proof*⟩

**lemma** *tail-is-vert*:  $(u,v) \in g\text{-}E G \implies u \in \text{set } (g\text{-}V G)$   
 ⟨*proof*⟩

**lemma** *head-is-vert*:  $(u,v) \in g\text{-}E G \implies v \in \text{set } (g\text{-}V G)$   
 ⟨*proof*⟩

**lemma** *tail-is-vert2*:  $(u,v) \in (g\text{-}E G)^+ \implies u \in \text{set } (g\text{-}V G)$   
 ⟨*proof*⟩

**lemma** *head-is-vert2*:  $(u,v) \in (g\text{-}E G)^+ \implies v \in \text{set } (g\text{-}V G)$   
 ⟨*proof*⟩

**lemma** *verts-set*:  $\text{set } (g\text{-}V G) = \{0 \dots \text{length } (g\text{-}V G)\}$   
 ⟨*proof*⟩

**lemma** *fin-verts*:  $\text{finite } (\text{set } (g\text{-}V G))$   
 ⟨*proof*⟩

**lemma** *zero-in-verts*:  $0 \in \text{set } (g\text{-}V G)$   
 ⟨*proof*⟩

**lemma** *verts-not-empty*:  $g\text{-}V G \neq []$   
 ⟨*proof*⟩

**lemma** *len-verts-gt0*:  $\text{length } (g\text{-}V G) > 0$   
 ⟨*proof*⟩

**lemma** *len-verts-gt1*:  $\text{length } (g-V \ G) > 1$   
*<proof>*

**lemma** *verts-ge-Suc0* :  $\neg [0..<\text{length } (g-V \ G)] = [0]$   
*<proof>*

**lemma** *distinct-verts1*:  $\text{distinct } [0..<\text{length } (g-V \ G)]$   
*<proof>*

**lemma** *distinct-verts2*:  $\text{distinct } (g-V \ G)$   
*<proof>*

**lemma** *single-entry*:  $\text{is-singleton } (g-V0 \ G)$   
*<proof>*

**lemma** *entry-is-0*:  $\text{the-elem } (g-V0 \ G) = 0$   
*<proof>*

**lemma** *wf-digraph*:  $\text{cfg-doms } G$  *<proof>*

**lemma** *path-entry-prepend-conv*:  $\text{path-entry } (g-E \ G) \ p \ n \implies p \neq [] \implies \exists v.$   
 $\text{path-entry } (g-E \ G) \ (\text{tl } p) \ v \wedge (v, n) \in (g-E \ G)$   
*<proof>*

**lemma** *path-verts*:  $\text{path-entry } (g-E \ G) \ p \ n \implies n \in \text{set } (g-V \ G)$   
*<proof>*

**lemma** *path-in-verts*:  
**assumes**  $\text{path-entry } (g-E \ G) \ l \ v$   
**shows**  $\text{set } l \subseteq \text{set } (g-V \ G)$   
*<proof>*

**lemma** *any-node-exits-path*:  
**assumes**  $v \in \text{set } (g-V \ G)$   
**shows**  $\exists p. \text{path-entry } (g-E \ G) \ p \ v$   
*<proof>*

**lemma** *entry0-path*:  $\text{path-entry } (g-E \ G) \ [] \ 0$   
*<proof>*

**definition** *reachable* ::  $\text{node} \Rightarrow \text{bool}$   
**where**  $\text{reachable } v \equiv v \in (g-E \ G)^* \ \{\!\!\{0\}\!\!\}$

**lemma** *path-entry-reachable*:  
**assumes**  $\text{path-entry } (g-E \ G) \ p \ n$   
**shows**  $\text{reachable } n$   
*<proof>*

**lemma** *nin-nodes-reachable*:  $n \notin \text{set } (g-V \ G) \implies \neg \text{reachable } n$

$\langle proof \rangle$

**lemma** *reachable-path-entry*:  $reachable\ n \implies \exists p. path\_entry\ (g-E\ G)\ p\ n$   
 $\langle proof \rangle$

**lemma** *path-entry-unconc*:  
 **assumes**  $path\_entry\ (g-E\ G)\ (la@lb)\ w$   
 **obtains**  $v$  **where**  $path\_entry\ (g-E\ G)\ lb\ v$   
 $\langle proof \rangle$

**lemma** *path-entry-append-conv*:  
  $path\_entry\ (g-E\ G)\ (v\#l)\ w \longleftrightarrow (path\_entry\ (g-E\ G)\ l\ v \wedge (v,w) \in (g-E\ G))$   
 $\langle proof \rangle$

**lemma** *takeWhileNot-path-entry*:  
 **assumes**  $path\_entry\ E\ p\ x$   
 **and**  $v \in set\ p$   
 **and**  $takeWhile\ ((\neq)\ v)\ (rev\ p) = c$   
 **shows**  $path\_entry\ E\ (rev\ c)\ v$   
 $\langle proof \rangle$

**lemma** *path-entry-last*:  $path\_entry\ (g-E\ G)\ p\ n \implies p \neq [] \implies last\ p = 0$   
 $\langle proof \rangle$

**lemma** *path-entry-loop*:  
 **assumes**  $n-path: path\_entry\ (g-E\ G)\ p\ n$   
 **and**  $n: n \in set\ p$   
 **shows**  $\exists p'. path\_entry\ (g-E\ G)\ p'\ n \wedge n \notin set\ p'$   
 $\langle proof \rangle$

**lemma** *path-entry-hd-edge*:  
 **assumes**  $path\_entry\ (g-E\ G)\ pa\ p$   
 **and**  $pa \neq []$   
 **shows**  $(hd\ pa, p) \in (g-E\ G)$   
 $\langle proof \rangle$

**lemma** *path-entry-edge*:  
 **assumes**  $pa \neq []$   
 **and**  $path\_entry\ (g-E\ G)\ pa\ p$   
 **shows**  $\exists u \in set\ pa. (path\_entry\ (g-E\ G)\ (rev\ (takeWhile\ ((\neq)\ u)\ (rev\ pa)))\ u) \wedge$   
  $(u,p) \in (g-E\ G)$   
 $\langle proof \rangle$

**definition** *is-tail* ::  $node \Rightarrow node \times node \Rightarrow bool$   
 **where**  $is-tail\ v\ e = (v = tail\ G\ e)$

**definition** *is-head* ::  $node \Rightarrow node \times node \Rightarrow bool$   
 **where**  $is-head\ v\ e = (v = head\ G\ e)$

**definition** *succs*:: *node*  $\Rightarrow$  *node set*  
**where** *succs* *v* = (*g-E* *G*) “ {*v*}

**lemma** *succ-in-verts*:  $s \in \text{succs } n \implies \{s, n\} \subseteq \text{set } (g-V \ G)$   
*<proof>*

**lemma** *succ0-not-nil*: *succs* 0  $\neq$  {}  
*<proof>*

**definition** *prevs*:: *node*  $\Rightarrow$  *node set* **where**  
*prevs* *v* = (*converse* (*g-E* *G*)) “ {*v*}

**lemma**  $v \in \text{succs } u \longleftrightarrow u \in \text{prevs } v$   
*<proof>*

**lemma** *succ-edge*:  $\forall v \in \text{succs } n. (n, v) \in g-E \ G$   
*<proof>*

**lemma** *prev-edge*:  $u \in \text{set } (g-V \ G) \implies \forall v \in \text{prevs } u. (v, u) \in g-E \ G$   
*<proof>*

**lemma** *succ-in-G*:  $\forall v \in \text{succs } n. v \in \text{set } (g-V \ G)$   
*<proof>*

**lemma** *succ-is-subset-of-verts*:  $\forall v \in \text{set } (g-V \ G). \text{succs } v \subseteq \text{set}(g-V \ G)$   
*<proof>*

**lemma** *fin-succs*:  $\forall v \in \text{set } (g-V \ G). \text{finite } (\text{succs } v)$   
*<proof>*

**lemma** *fin-succs'*:  $v < \text{length } (g-V \ G) \implies \text{finite } (\text{succs } v)$   
*<proof>*

**lemma** *succ-range*:  $\forall v \in \text{succs } n. v < \text{length } (g-V \ G)$   
*<proof>*

**lemma** *path-entry-gt*:

**assumes**  $\forall p. \text{path-entry } E \ p \ n \longrightarrow d \in \text{set } p$   
**and**  $\forall p. \text{path-entry } E \ p \ n' \longrightarrow n \in \text{set } p$   
**shows**  $\forall p. \text{path-entry } E \ p \ n' \longrightarrow d \in \text{set } p$   
*<proof>*

**definition** *dominate* :: *nat*  $\Rightarrow$  *nat*  $\Rightarrow$  *bool*

**where** *dominate* *n1* *n2*  $\equiv$   
 $\forall pa. \text{path-entry } (g-E \ G) \ pa \ n2 \longrightarrow$   
 $(n1 \in \text{set } pa \vee n1 = n2)$

**definition** *strict-dominate*:: *nat*  $\Rightarrow$  *nat*  $\Rightarrow$  *bool* **where**  
*strict-dominate* *n1* *n2*  $\equiv$

$\forall pa. \text{path-entry } (g-E \ G) \ pa \ n2 \longrightarrow$   
 $(n1 \in \text{set } pa \wedge n1 \neq n2)$

**lemma** *any-dominate-unreachable*:  $\neg \text{reachable } n \Longrightarrow \text{dominate } d \ n$   
*<proof>*

**lemma** *any-sdominate-unreachable*:  $\neg \text{reachable } n \Longrightarrow \text{strict-dominate } d \ n$   
*<proof>*

**lemma** *dom-reachable*:  $\text{reachable } n \Longrightarrow \text{dominate } d \ n \Longrightarrow \text{reachable } d$   
*<proof>*

**lemma** *dominate-refl*:  $\text{dominate } n \ n$   
*<proof>*

**lemma** *entry0-dominates-all*:  $\forall p \in \text{set } (g-V \ G). \ \text{dominate } 0 \ p$   
*<proof>*

**lemma** *strict-dominate i j*:  $j \in \text{set } (g-V \ G) \Longrightarrow i \neq j$   
*<proof>*

**definition** *non-strict-dominate*::  $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool}$  **where**  
 $\text{non-strict-dominate } n1 \ n2 \equiv \exists pa. \text{path-entry } (g-E \ G) \ pa \ n2 \wedge (n1 \notin \text{set } pa)$

**lemma** *any-sdominate-0*:  $n \in \text{set } (g-V \ G) \Longrightarrow \text{non-strict-dominate } n \ 0$   
*<proof>*

**lemma** *non-sdominate-succ*:  $(i, j) \in g-E \ G \Longrightarrow k \neq i \Longrightarrow \text{non-strict-dominate } k \ i$   
 $i \Longrightarrow \text{non-strict-dominate } k \ j$   
*<proof>*

**lemma** *any-node-non-sdom0*:  $\text{non-strict-dominate } k \ 0$   
*<proof>*

**lemma** *nonstrict-eq*:  $\text{non-strict-dominate } i \ j \Longrightarrow \neg \text{strict-dominate } i \ j$   
*<proof>*

**lemma** *dominate-trans*:  
**assumes**  $\text{dominate } n1 \ n2$   
**and**  $\text{dominate } n2 \ n3$   
**shows**  $\text{dominate } n1 \ n3$   
*<proof>*

**lemma** *len-takeWhile-lt*:  $x \in \text{set } xs \Longrightarrow \text{length } (\text{takeWhile } ((\neq) \ x) \ xs) < \text{length } xs$   
*<proof>*

**lemma** *len-takeWhile-comp*:  
**assumes**  $n1 \in \text{set } xs$

**and**  $n2 \in \text{set } xs$   
**and**  $n1 \neq n2$   
**shows**  $\text{length } (\text{takeWhile } ((\neq) n1) xs) \neq \text{length } (\text{takeWhile } ((\neq) n2) xs)$   
 $\langle \text{proof} \rangle$

**lemma** *len-takeWhile-comp1:*

**assumes**  $n1 \in \text{set } xs$   
**and**  $n2 \in \text{set } xs$   
**and**  $n1 \neq n2$   
**shows**  $\text{length } (\text{takeWhile } ((\neq) n1) (\text{rev } (x \# xs))) \neq \text{length } (\text{takeWhile } ((\neq) n2) (\text{rev } (x \# xs)))$   
 $\langle \text{proof} \rangle$

**lemma** *len-takeWhile-comp2:*

**assumes**  $n1 \in \text{set } xs$   
**and**  $n2 \notin \text{set } xs$   
**shows**  $\text{length } (\text{takeWhile } ((\neq) n1) (\text{rev } (x \# xs))) \neq \text{length } (\text{takeWhile } ((\neq) n2) (\text{rev } (x \# xs)))$   
 $\langle \text{proof} \rangle$

**lemma** *len-compare1:*

**assumes**  $n1 = x$  **and**  $n2 \neq x$   
**shows**  $\text{length } (\text{takeWhile } ((\neq) n1) (\text{rev } (x \# xs))) \neq \text{length } (\text{takeWhile } ((\neq) n2) (\text{rev } (x \# xs)))$   
 $\langle \text{proof} \rangle$

**lemma** *len-compare2:*

**assumes**  $n1 \in \text{set } xs$   
**and**  $n1 \neq n2$   
**shows**  $\text{length } (\text{takeWhile } ((\neq) n1) (\text{rev } (x \# xs))) \neq \text{length } (\text{takeWhile } ((\neq) n2) (\text{rev } (x \# xs)))$   
 $\langle \text{proof} \rangle$

**lemma** *len-takeWhile-set:*

**assumes**  $\text{length } (\text{takeWhile } ((\neq) n1) xs) > \text{length } (\text{takeWhile } ((\neq) n2) xs)$   
**and**  $n1 \neq n2$   
**and**  $n1 \in \text{set } xs$   
**and**  $n2 \in \text{set } xs$   
**shows**  $\text{set } (\text{takeWhile } ((\neq) n2) xs) \subseteq \text{set } (\text{takeWhile } ((\neq) n1) xs)$   
 $\langle \text{proof} \rangle$

**lemma** *reachable-dom-acyclic:*

**assumes** *reachable*  $n2$   
**and** *dominate*  $n1 n2$   
**and** *dominate*  $n2 n1$   
**shows**  $n1 = n2$   
 $\langle \text{proof} \rangle$

**lemma** *sdom-dom:* *strict-dominate*  $n1 n2 \implies \text{dominate } n1 n2$



*<proof>*

**lemma** *dominate-sdominate*:  $\text{dominate } n1 \ n2 \implies n1 \neq n2 \implies \text{strict-dominate } n1 \ n2$   
*<proof>*

**lemma** *sdom-neg*:  
 **assumes** *reachable*  $n2$   
 **and** *strict-dominate*  $n1 \ n2$   
 **shows**  $n1 \neq n2$   
*<proof>*

**lemma** *reachable-dom-acyclic2*:  
 **assumes** *reachable*  $n2$   
 **and** *strict-dominate*  $n1 \ n2$   
 **shows**  $\neg \text{dominate } n2 \ n1$   
*<proof>*

**lemma** *not-dom-eq-not-sdom*:  $\neg \text{dominate } n1 \ n2 \implies \neg \text{strict-dominate } n1 \ n2$   
*<proof>*

**lemma** *reachable-sdom-acyclic*:  
 **assumes** *reachable*  $n2$   
 **and** *strict-dominate*  $n1 \ n2$   
 **shows**  $\neg \text{strict-dominate } n2 \ n1$   
*<proof>*

**lemma** *strict-dominate-trans1*:  
 **assumes** *strict-dominate*  $n1 \ n2$   
 **and** *dominate*  $n2 \ n3$   
 **shows** *strict-dominate*  $n1 \ n3$   
*<proof>*

**lemma** *strict-dominate-trans2*:  
 **assumes** *dominate*  $n1 \ n2$   
 **and** *strict-dominate*  $n2 \ n3$   
 **shows** *strict-dominate*  $n1 \ n3$   
*<proof>*

**lemma** *strict-dominate-trans*:  
 **assumes** *strict-dominate*  $n1 \ n2$   
 **and** *strict-dominate*  $n2 \ n3$   
 **shows** *strict-dominate*  $n1 \ n3$   
*<proof>*

**lemma** *sdominate-dominate-succs*:  
 **assumes** *i-sdom-j*: *strict-dominate*  $i \ j$   
 **and** *j-in-succ-k*:  $j \in \text{succs } k$   
 **shows** *dominate*  $i \ k$

*<proof>*

**end**

**end**

## 2 More auxiliary lemmas for Lists Sorted wrt $<$

**theory** *Sorted-Less2*

**imports** *Main HOL-Data-Structures.Cmp HOL-Data-Structures.Sorted-Less*

**begin**

**lemma** *Cons-sorted-less*:  $\text{sorted } (\text{rev } xs) \implies \forall x \in \text{set } xs. x < p \implies \text{sorted } (\text{rev } (p \# xs))$

*<proof>*

**lemma** *Cons-sorted-less-nth*:  $\forall x < \text{length } xs. xs ! x < p \implies \text{sorted } (\text{rev } xs) \implies \text{sorted } (\text{rev } (p \# xs))$

*<proof>*

**lemma** *distinct-sorted-rev*:  $\text{sorted } (\text{rev } xs) \implies \text{distinct } xs$

*<proof>*

**lemma** *sorted-le2lt*:

**assumes** *List.sorted xs*

**and** *distinct xs*

**shows** *sorted xs*

*<proof>*

**lemma** *sorted-less-sorted-list-of-set*:  $\text{sorted } (\text{sorted-list-of-set } S)$

*<proof>*

**lemma** *distinct-sorted*:  $\text{sorted } xs \implies \text{distinct } xs$

*<proof>*

**lemma** *sorted-less-set-unique*:

**assumes** *sorted xs*

**and** *sorted ys*

**and**  $\text{set } xs = \text{set } ys$

**shows**  $xs = ys$

*<proof>*

**lemma** *sorted-less-rev-set-unique*:

**assumes** *sorted (rev xs)*

**and** *sorted (rev ys)*

**and**  $\text{set } xs = \text{set } ys$

**shows**  $xs = ys$

*<proof>*

**lemma** *sorted-less-set-eq*:  
**assumes** *sorted xs*  
**shows**  $xs = \text{sorted-list-of-set } (\text{set } xs)$   
 $\langle \text{proof} \rangle$

**lemma** *sorted-less-rev-set-eq*:  
**assumes** *sorted (rev xs)*  
**shows**  $\text{sorted-list-of-set } (\text{set } xs) = \text{rev } xs$   
 $\langle \text{proof} \rangle$

**lemma** *sorted-insert-remove1*:  $\text{sorted } w \implies (\text{insert } a (\text{remove1 } a w)) = \text{sorted-list-of-set } (\text{insert } a (\text{set } w))$   
 $\langle \text{proof} \rangle$

**end**

### 3 Operations on sorted lists

**theory** *Sorted-List-Operations2*  
**imports** *Sorted-Less2*  
**begin**

The definition and the `inter_sorted_correct` lemma in this theory are the same as those in `Collections` [2]. except the former is for a descending list while the latter is for an ascending one.

**fun** *inter-sorted-rev* ::  $'a::\{\text{linorder}\}$  list  $\Rightarrow 'a$  list  $\Rightarrow 'a$  list **where**  
*inter-sorted-rev* [] *l2* = []  
| *inter-sorted-rev* *l1* [] = []  
| *inter-sorted-rev* (*x1* # *l1*) (*x2* # *l2*) =  
  (if (*x1* > *x2*) then (*inter-sorted-rev* *l1* (*x2* # *l2*)) else  
  (if (*x1* = *x2*) then *x1* # (*inter-sorted-rev* *l1* *l2*) else *inter-sorted-rev* (*x1* # *l1*)  
*l2*))

**lemma** *inter-sorted-correct* :  
**assumes** *l1-OK*: *sorted (rev l1)*  
**assumes** *l2-OK*: *sorted (rev l2)*  
**shows**  $\text{sorted } (\text{rev } (\text{inter-sorted-rev } l1 \ l2)) \wedge \text{set } (\text{inter-sorted-rev } l1 \ l2) = \text{set } l1 \cap \text{set } l2$   
 $\langle \text{proof} \rangle$

**lemma** *inter-sorted-rev-refl*:  $\text{inter-sorted-rev } xs \ xs = xs$   
 $\langle \text{proof} \rangle$

**lemma** *inter-sorted-correct-col*:  
**assumes** *sorted (rev xs)*  
**and** *sorted (rev ys)*  
**shows**  $(\text{inter-sorted-rev } xs \ ys) = \text{rev } (\text{sorted-list-of-set } (\text{set } xs \cap \text{set } ys))$   
 $\langle \text{proof} \rangle$

**lemma** *cons-set-eq*:  $set (x \# xs) \cap set xs = set xs$   
 ⟨*proof*⟩

**lemma** *inter-sorted-cons*:  $sorted (rev (x \# xs)) \implies inter\text{-sorted}\text{-rev } (x \# xs) xs = xs$   
 ⟨*proof*⟩

**end**

## 4 A semilattice of reversed-ordered list

**theory** *Dom-Semi-List*

**imports** *Main Jinja.Semilat Sorted-List-Operations2 Sorted-Less2 Cfg*  
**begin**

**type-synonym** *node* = *nat*

**context** *cfg-doms*  
**begin**

**definition** *nodes* :: *nat list*  
**where**  $nodes \equiv (g\text{-}V\ G)$

**definition** *nodes-le* :: *node list*  $\Rightarrow$  *node list*  $\Rightarrow$  *bool* **where**  
 $nodes\text{-}le\ xs\ ys \equiv (sorted (rev\ ys) \wedge sorted (rev\ xs) \wedge (set\ ys) \subseteq (set\ xs)) \vee xs = ys$

**definition** *nodes-sup* :: *node list*  $\Rightarrow$  *node list*  $\Rightarrow$  *node list* **where**  
 $nodes\text{-}sup = (\lambda x\ y. inter\text{-sorted}\text{-rev } x\ y)$

**definition** *nodes-semi* :: *node list sl* **where**  
 $nodes\text{-}semi \equiv ((rev \circ sorted\text{-list-of-set}) \text{ ` } (Pow (set (nodes))), nodes\text{-}le, nodes\text{-}sup)$

**lemma** *subset-nodes-inpow*:  
**assumes**  $sorted (rev\ xs)$   
**and**  $set\ xs \subseteq set\ nodes$   
**shows**  $xs \in (rev \circ sorted\text{-list-of-set}) \text{ ` } (Pow (set\ nodes))$   
 ⟨*proof*⟩

**lemma** *nil-in-A*:  $[] \in (rev \circ sorted\text{-list-of-set}) \text{ ` } (Pow (set\ nodes))$   
 ⟨*proof*⟩

**lemma** *single-n-in-A*:  $p < length\ nodes \implies [p] \in (rev \circ sorted\text{-list-of-set}) \text{ ` } (Pow (set\ nodes))$   
 ⟨*proof*⟩

**lemma** *inpow-subset-nodes*:  
**assumes**  $xs \in (rev \circ sorted\text{-list-of-set}) \text{ ` } (Pow (set\ nodes))$

**shows**  $set\ xs \subseteq set\ nodes$   
*<proof>*

**lemma** *inter-in-pow-nodes:*

**assumes**  $xs \in (rev \circ sorted\ list\ of\ set) \text{ ' } (Pow\ (set\ nodes))$   
**shows**  $(rev \circ sorted\ list\ of\ set)(set\ xs \cap set\ ys) \in (rev \circ (sorted\ list\ of\ set)) \text{ ' } (Pow\ (set\ nodes))$   
*<proof>*

**lemma** *nodes-le-order: order nodes-le ((rev o sorted-list-of-set) ' (Pow (set nodes)))*  
*<proof>*

**lemma** *nodes-semi-auxi:*

*let*  $A = (rev \circ sorted\ list\ of\ set) \text{ ' } (Pow\ (set\ (nodes)))$ ;  
 $r = nodes\ le$ ;  
 $f = (\lambda x\ y. (inter\ sorted\ rev\ x\ y))$   
*in*  $semilat(A, r, f)$   
*<proof>*

**lemma** *nodes-semi-is-semilat: semilat (nodes-semi)*  
*<proof>*

**lemma** *sorted-rev-subset-len-lt:*

**assumes**  $sorted\ (rev\ a)$   
**and**  $sorted\ (rev\ b)$   
**and**  $set\ a \subset set\ b$   
**shows**  $length\ a < length\ b$   
*<proof>*

**lemma** *wf-nodes-le-auxi: wf {(y, x). (sorted (rev y) ^ sorted (rev x) ^ set y C set x) ^ x ≠ y}*  
*<proof>*

**lemma** *wf-nodes-le-auxi2:*

*wf*  $\{(y, x). sorted\ (rev\ y) \wedge sorted\ (rev\ x) \wedge set\ y \subset set\ x \wedge rev\ x \neq rev\ y\}$   
*<proof>*

**lemma** *wf-nodes-le: wf {(y, x). nodes-le x y ^ x ≠ y}*  
*<proof>*

**lemma** *acc-nodes-le: acc nodes-le*  
*<proof>*

**lemma** *acc-nodes-le2: acc (fst (snd nodes-semi))*  
*<proof>*

**lemma** *nodes-le-refl [iff] : nodes-le s s*  
*<proof>*

end

end

## 5 A kildall's algorithm for computing dominators

**theory** *Dom-Kildall*

**imports** *Dom-Semi-List HOL-Library.While-Combinator Jinja.SemilatAlg*

**begin**

A kildall's algorithm for computing dominators. It uses the ideas and the framework of kildall's algorithm implemented in Jinja [3], and modifications are needed to make it work for a fast algorithm for computing dominators

**type-synonym** *state-dom* = *nat list*

**primrec** *propa* ::

*'s binop*  $\Rightarrow$  (*nat*  $\times$  *'s*) *list*  $\Rightarrow$  *'s list*  $\Rightarrow$  *nat list*  $\Rightarrow$  *'s list* \* *nat list*

**where**

*propa f []*  $\tau s$  *wl* = ( $\tau s, wl$ )  
| *propa f (q'# qs)*  $\tau s$  *wl* = (*let* (*q, \tau*) = *q'*;  
                                  *u* = ( $\tau \sqcup_f \tau s!q$ );  
                                  *wl'* = (*if* *u* =  $\tau s!q$  *then* *wl*  
                                      *else* (*insort* *q* (*remove1* *q* *wl*)))  
*in propa f qs* ( $\tau s[q := u]$ ) *wl'*)

**definition** *iter* ::

*'s binop*  $\Rightarrow$  *'s step-type*  $\Rightarrow$  *'s list*  $\Rightarrow$  *nat list*  $\Rightarrow$  *'s list*  $\times$  *nat list*

**where**

*iter f step*  $\tau s$  *w* =  
  *while* ( $\lambda(\tau s, w). w \neq []$ )  
    ( $\lambda(\tau s, w). \text{let } p = \text{hd } w$   
      *in propa f* (*step* *p* ( $\tau s!p$ ))  $\tau s$  (*tl w*))  
  ( $\tau s, w$ )

**definition** *unstables* :: *state-dom ord*  $\Rightarrow$  *state-dom step-type*  $\Rightarrow$  *state-dom list*  $\Rightarrow$  *nat list*

**where**

*unstables r step*  $\tau s$  = *sorted-list-of-set* {*p. p* < *size*  $\tau s \wedge \neg$  *stable r step*  $\tau s$  *p*}

**definition** *kildall* :: *state-dom ord*  $\Rightarrow$  *state-dom binop*  $\Rightarrow$  *state-dom step-type*  $\Rightarrow$  *state-dom list*  $\Rightarrow$  *state-dom list* **where**

*kildall r f step*  $\tau s$  = *fst*(*iter f step*  $\tau s$  (*unstables r step*  $\tau s$ ))

**lemma** *init-worklist-is-sorted*: *sorted* (*unstables r step*  $\tau s$ )

*<proof>*

**context** *cfg-doms*

**begin**

**definition** *transf* :: *node*  $\Rightarrow$  *state-dom*  $\Rightarrow$  *state-dom* **where**  
*transf n input*  $\equiv$  (*n* # *input*)

**definition** *exec* :: *node*  $\Rightarrow$  *state-dom*  $\Rightarrow$  (*node*  $\times$  *state-dom*) *list*  
**where** *exec n xs* = *map* ( $\lambda pc. (pc, (transf\ n\ xs))$ ) (*rev* (*sorted-list-of-set*(*succs*  
*n*)))

**lemma** *transf-res-is-rev*: *sorted* (*rev ns*)  $\implies$  *n* > *hd ns*  $\implies$  *sorted* (*rev* ((*transf n*  
(*ns*))))  
(*proof*)

**abbreviation** *start*  $\equiv$  [] # (*replicate* (*length* (*g-V G*) - 1) ( (*rev*[0..*length*(*g-V*  
*G*]))))

**definition** *dom-kildall* :: *state-dom list*  $\Rightarrow$  *state-dom list*  
**where** *dom-kildall* = *kildall* (*fst* (*snd nodes-semi*)) (*snd* (*snd nodes-semi*)) *exec*

**definition** *dom*:: *nat*  $\Rightarrow$  *nat*  $\Rightarrow$  *bool* **where**  
*dom i j*  $\equiv$  (*let res* = (*dom-kildall start*) !*j in i*  $\in$  (*set res*)  $\vee$  *i* = *j*)

**lemma** *dom-refl*: *dom i i*  
(*proof*)

**definition** *strict-dom* :: *nat*  $\Rightarrow$  *nat*  $\Rightarrow$  *bool* **where**  
*strict-dom i j*  $\equiv$  (*let res* = (*dom-kildall start*) !*j in i*  $\in$  *set res*)

**lemma** *strict-domI1*: (*dom-kildall* ( [] # (*replicate* (*length* (*g-V G*) - 1) ( (*rev*[0..*length*(*g-V*  
*G*])))))) !*j* = *res*  $\implies$  *i*  $\in$  *set res*  $\implies$  *strict-dom i j*  
(*proof*)

**lemma** *strict-domD*:  
*strict-dom i j*  $\implies$   
*dom-kildall* ( ( [] # (*replicate* (*length* (*g-V G*) - 1) ( (*rev*[0..*length*(*g-V G*])))))) !*j*  
= *a*  $\implies$   
*i*  $\in$  *set a*  
(*proof*)

**lemma** *sdom-dom*: *strict-dom i j*  $\implies$  *dom i j*  
(*proof*)

**lemma** *not-sdom-not-dom*:  $\neg$ *strict-dom i j*  $\implies$  *i*  $\neq$  *j*  $\implies$   $\neg$ *dom i j*  
(*proof*)

**lemma** *dom-sdom*: *dom i j*  $\implies$  *i*  $\neq$  *j*  $\implies$  *strict-dom i j*

*<proof>*

**end**

**end**

## 6 Properties of the kildall's algorithm on the semi-lattice

**theory** *Dom-Kildall-Property*

**imports** *Dom-Kildall Jinja.Listn Jinja.Kildall-1*

**begin**

**lemma** *sorted-list-len-lt*:  $x \subset y \implies \text{finite } y \implies \text{length } (\text{sorted-list-of-set } x) < \text{length } (\text{sorted-list-of-set } y)$

*<proof>*

**lemma** *wf-sorted-list*:

$\text{wf } ((\lambda(x,y). (\text{sorted-list-of-set } x, \text{sorted-list-of-set } y)) \text{ ' } \text{finite-psubset})$

*<proof>*

**lemma** *sorted-list-psub*:

$\text{sorted } w \longrightarrow$

$w \neq [] \longrightarrow$

$(\text{sorted-list-of-set } (\text{set } (\text{tl } w)), w) \in (\lambda(x, y). (\text{sorted-list-of-set } x, \text{sorted-list-of-set } y)) \text{ ' } \{(A, B). A \subset B \wedge \text{finite } B\}$

*<proof>*

**locale** *dom-sl* = *cfg-doms* +

**fixes** *A* and *r* and *f* and *step* and *start* and *n*

**defines**  $A \equiv ((\text{rev} \circ \text{sorted-list-of-set}) \text{ ' } (\text{Pow } (\text{set } (\text{nodes}))))$

**defines**  $r \equiv \text{nodes-le}$

**defines**  $f \equiv \text{nodes-sup}$

**defines**  $n \equiv (\text{size } \text{nodes})$

**defines**  $\text{step} \equiv \text{exec}$

**defines**  $\text{start} \equiv ([] \# (\text{replicate } (\text{length } (g\text{-}V\ G) - 1) (\text{rev}[0..\text{<n>}])))::\text{state-dom}$   
*list*

**begin**

**lemma** *is-semi*:  $\text{semilat}(A,r,f)$

*<proof>*

**lemma** *Cons-less-Conss* [*simp*]:

$x \# xs \sqsubseteq_r y \# ys = (x \sqsubseteq_r y \wedge xs \sqsubseteq_r ys) \vee x = y \wedge xs \sqsubseteq_r ys$

*<proof>*

**lemma** *acc-le-listI* [*intro!*]:



$acc\ r \implies acc\ (Listn.le\ r)$   
*<proof>*

**lemma** *wf-listn*:  $wf\ \{(y,x). x \sqsubset_{Listn.le\ r}\ y\}$   
*<proof>*

**lemma** *wf-listn'*:  $wf\ \{(y,x). x \sqsubset_r\ y\}$   
*<proof>*

**lemma** *wf-listn-termination-rel*:  
 $wf\ (\{(y,x). x \sqsubset_{Listn.le\ r}\ y\} \lt *lex* \gt (\lambda(x,y). (sorted-list-of-set\ x, sorted-list-of-set\ y)))$  'finite-psubset'  
*<proof>*

**lemma** *inA-is-sorted*:  $xs \in A \implies sorted\ (rev\ xs)$   
*<proof>*

**lemma** *list-nA-lt-refl*:  $xs \in nlists\ n\ A \longrightarrow xs \sqsubset_r\ xs$   
*<proof>*

**lemma** *nil-inA*:  $[] \in A$   
*<proof>*

**lemma** *upt-n-in-pow-nodes*:  $\{0..<n\} \in Pow\ (set\ nodes)$   
*<proof>*

**lemma** *rev-all-inA*:  $rev\ [0..<n] \in A$   
*<proof>*

**lemma** *len-start-is-n*:  $length\ start = n$   
*<proof>*

**lemma** *len-start-is-len-verts*:  $length\ start = length\ (g-V\ G)$   
*<proof>*

**lemma** *start-len-gt-0*:  $length\ start > 0$   
*<proof>*

**lemma** *start-subset-A*:  $set\ start \subseteq A$   
*<proof>*

**lemma** *start-in-A* :  $start \in (nlists\ n\ A)$   
*<proof>*

**lemma** *sorted-start-nth*:  $i < n \implies sorted\ (rev\ (start!i))$   
*<proof>*

**lemma** *start-nth0-empty*:  $start!0 = []$   
*<proof>*

**lemma** *start-nth-lt0-all*:  $\forall p \in \{1..< \text{length } \text{start}\}. \text{start!}p = (\text{rev } [0..<n])$   
 <proof>

**lemma** *in-nodes-lt-n*:  $x \in \text{set } (g\text{-}V\ G) \implies x < n$   
 <proof>

**lemma** *start-nth0-unstable-axi*:  $\neg [0] \sqsubseteq_r (\text{rev } [0..<n])$   
 <proof>

**lemma** *start-nth0-unstable*:  $\neg \text{stable } r \text{ step } \text{start } 0$   
 <proof>

**lemma** *start-nth-unstable*:  
 assumes  $p \in \{1..< \text{length } (g\text{-}V\ G)\}$   
 and  $\text{succs } p \neq \{\}$   
 shows  $\neg \text{stable } r \text{ step } \text{start } p$   
 <proof>

**lemma** *start-unstable-cond*:  
 assumes  $\text{succs } p \neq \{\}$   
 and  $p < \text{length } (g\text{-}V\ G)$   
 shows  $\neg \text{stable } r \text{ step } \text{start } p$   
 <proof>

**lemma** *unstable-start*:  $\text{unstabes } r \text{ step } \text{start} = \text{sorted-list-of-set } (\{p. \text{succs } p \neq \{\} \wedge p < \text{length } \text{start}\})$   
 <proof>

**end**

**declare** *sorted-list-of-set-insert-remove*[simp del]

**context** *dom-sl*

**begin**

**lemma** (**in** *dom-sl*) *decomp-propa*:  $\bigwedge ss\ w.$

$(\forall (q,t) \in \text{set } qs. q < \text{size } ss \wedge t \in A) \implies$

$\text{sorted } w \implies$

$\text{set } ss \subseteq A \implies$

$\text{propa } f\ qs\ ss\ w = (\text{merges } f\ qs\ ss, (\text{sorted-list-of-set } (\{q. \exists t. (q,t) \in \text{set } qs \wedge t \sqcup_f$

$(ss!q) \neq ss!q\} \cup \text{set } w)))$  <proof><proof><proof><proof><proof><proof><proof><proof><proof><proof><proof><proof><proof><proof><proof><proof>

**lemma** (**in** *Semilat*) *list-update-le-listI* [rule-format]:

$\text{set } xs \subseteq A \longrightarrow \text{set } ys \subseteq A \longrightarrow xs \sqsubseteq_r ys \longrightarrow p < \text{size } xs \longrightarrow$

$x \sqsubseteq_r ys!p \longrightarrow x \in A \longrightarrow$

$xs[p := x \sqcup_f xs!p] \sqsubseteq_r ys$  <proof><proof><proof><proof><proof><proof><proof><proof><proof><proof><proof><proof><proof><proof><proof>

## 7 Soundness and completeness

```
theory Dom-Kildall-Correct  
imports Dom-Kildall-Property  
begin
```

```
context dom-sl  
begin
```

```
lemma entry-dominate-dom:  
  assumes  $i \in \text{set } (g-V \ G)$   
    and  $\text{dominate } i \ 0$   
  shows  $\text{dom } i \ 0$   
  <proof>
```

```
lemma path-entry-dom:  
  fixes  $pa \ i \ d$   
  assumes  $\text{path-entry } (g-E \ G) \ pa \ i$   
    and  $\text{dom } d \ i$   
  shows  $d \in \text{set } pa \vee d = i$   
  <proof>
```

```
lemma dom-sound:  $\text{dom } i \ j \implies \text{dominate } i \ j$   
<proof>
```

```
lemma sdom-sound:  $\text{strict-dom } i \ j \implies j \in \text{set } (g-V \ G) \implies \text{strict-dominate } i \ j$   
<proof>
```

```
lemma dom-complete-auxi:  $i < \text{length } \text{start} \implies (\text{dom-kildall } \text{start})!i = ss' \wedge k \notin$   
 $\text{set } ss' \implies \text{non-strict-dominate } k \ i$   
<proof>
```

```
lemma notsdom-notsdominate:  $\neg \text{strict-dom } i \ j \implies j < \text{length } \text{start} \implies \text{non-strict-dominate}$   
 $i \ j$   
<proof>
```

```
lemma notsdom-notsdominate':  $\neg \text{strict-dom } i \ j \implies j < \text{length } \text{start} \implies \neg$   
 $\text{strict-dominate } i \ j$   
<proof>
```

```
lemma dom-complete:  $\text{strict-dominate } i \ j \implies j < \text{size } \text{start} \implies \text{strict-dom } i \ j$   
<proof>
```

```
end
```

```
end
```

## References

- [1] K. D. Cooper, T. J. Harvey, and K. Kennedy. A simple, fast dominance algorithm. Technical report, Rice University, Houston, Jan. 2006. <https://scholarship.rice.edu/handle/1911/96345>.
- [2] P. Lammich. Operations on sorted lists. 2009. [https://www.isa-afp.org/browser\\_info/current/AFP/Collections/Sorted\\_List\\_Operations.html](https://www.isa-afp.org/browser_info/current/AFP/Collections/Sorted_List_Operations.html).
- [3] T. Nipkow and G. Klein. Operations on sorted lists. 2000. [https://www.isa-afp.org/browser\\_info/current/AFP/Jinja/Kildall.html](https://www.isa-afp.org/browser_info/current/AFP/Jinja/Kildall.html).