

# Elementary Graph Traversal Algorithms

Mohammad Abdulaziz  
King's College London

February 6, 2026

## Abstract

In this entry, we verify the correctness of:

- Depth-first search
- Breadth-first search

The entry's main aim is pedagogical. It has the formal material presented in one chapter of the book *Functional Data Structures and Algorithms: A Proof Assistant Approach*. The verification is based on a simple set-based representation of directed graphs. The main feature is that a graph's set of vertices is implicitly represented by its edges. The main focus of the development is to develop a representation suitable for mathematical reasoning and for executable algorithms. The entry also shows the verification of executable algorithms that need background mathematical libraries with basic Isabelle tools. Two main features are its automation setup aimed at verifying executable graph algorithms in a human-readable way and to verify the algorithms w.r.t. the same graph representation, making all of them compatible. The setup we use here includes using the function package to model loops and reason about their termination, using records to model program states, and using *locales* to implement parametric as well as step-wise refinement based verification. The material presented here is a part of an ongoing development of formal results on graphs and graph algorithms in Isabelle/HOL<sup>1</sup>.

## Contents

<b>1</b>	<b>A Basic Representation of Diraphs</b>	<b>2</b>
<b>2</b>	<b>A Theory on Vertex Walks in a Digraph</b>	<b>5</b>
<b>3</b>	<b>Vwalks to paths (as opposed to arc walks (<i>awalk-to-aph</i> before))</b>	<b>16</b>

---

<sup>1</sup><https://github.com/mabdula/Isabelle-Graph-Library>

<b>4</b>	<b>Distances</b>	<b>21</b>
4.1	Distance from a vertex . . . . .	21
4.2	Shortest Paths . . . . .	22
4.3	Distance from a set of vertices . . . . .	24
<b>5</b>	<b>A Digraph Representation for Efficient Executable Functions</b>	<b>27</b>
5.1	Abstraction lemmas . . . . .	27
5.2	Abstraction lemmas . . . . .	30
<b>6</b>	<b>Depth-First Search</b>	<b>31</b>
6.1	The program state . . . . .	31
6.2	Setup for automation . . . . .	31
6.3	A <i>locale</i> for fixing data structures and their implemenations .	32
6.4	Defining the Algorithm . . . . .	32
6.5	Setup for Reasoning About the Algorithm . . . . .	33
6.6	Loop Invariants . . . . .	35
6.7	Proofs that the Invariants Hold . . . . .	40
6.8	Termination . . . . .	42
6.9	Final Correctness Theorems . . . . .	43
<b>7</b>	<b>Breadth-First Search</b>	<b>44</b>
7.1	The Program State . . . . .	44
7.2	Setup for Automation . . . . .	44
7.3	A <i>locale</i> for fixing data structures and their implemenations .	44
7.4	The Algorithm Definition . . . . .	45
7.5	Setup for Reasoning About the Algorithm . . . . .	45
7.6	The Loop Invariants . . . . .	47
7.7	Termination Measures and Relation . . . . .	48
7.8	Proofs that the Invariants Hold . . . . .	52
7.9	Termination . . . . .	57
7.10	Final Correctness Theorems . . . . .	58

**theory** *More-Lists*

**imports** *Main*

**begin**

**lemma** *list-2-preds-aux*:

$$\begin{aligned} & \llbracket x' \in \text{set } xs; P1\ x'; \bigwedge xs1\ x\ xs2. \llbracket xs = xs1\ @\ [x]\ @\ xs2; P1\ x \rrbracket \implies \\ & \quad \exists\ ys1\ y\ ys2. x\ \# \ xs2 = ys1\ @\ [y]\ @\ ys2 \wedge P2\ y \rrbracket \implies \\ & \quad \exists\ xs1\ x\ xs2. xs = xs1\ @\ [x]\ @\ xs2 \wedge P2\ x \wedge (\forall y \in \text{set } xs2. \neg P1\ y) \end{aligned}$$

*<proof>*

**lemma** *list-2-preds*:

$$\begin{aligned} & \llbracket x \in \text{set } xs; P1\ x; \bigwedge xs1\ x\ xs2. \llbracket xs = xs1\ @\ [x]\ @\ xs2; P1\ x \rrbracket \implies \exists\ ys1\ y\ ys2. \\ & x\ \# \ xs2 = ys1\ @\ [y]\ @\ ys2 \wedge P2\ y \rrbracket \implies \\ & \quad \exists\ xs1\ x\ xs2. xs = xs1\ @\ [x]\ @\ xs2 \wedge P2\ x \wedge (\forall y \in \text{set } xs2. \neg P1\ y \wedge \neg P2\ y) \end{aligned}$$

*<proof>*

**lemma** *list-inter-mem-iff*:  $set\ xs \cap s \neq \{\}$   $\longleftrightarrow (\exists\ xs1\ x\ xs2. xs = xs1 @ [x] @ xs2 \wedge x \in s)$

*<proof>*

**end**

**theory** *Pair-Graph*

**imports**

*Main*

*Graph-Theory.Rtrancl-On*

**begin**

## 1 A Basic Representation of Diraphs

**type-synonym** *'v dgraph* = (*'v* × *'v*) *set*

**definition** *dVs*::(*'v* × *'v*) *set*  $\Rightarrow$  *'v set* **where**

$dVs\ G = \bigcup \{\{v1, v2\} \mid v1\ v2. (v1, v2) \in G\}$

**lemma** *induct-pcpl*:

$\llbracket P\ []; \bigwedge x. P\ [x]; \bigwedge x\ y\ zs. P\ zs \Longrightarrow P\ (x\ \#\ y\ \#\ zs) \rrbracket \Longrightarrow P\ xs$

*<proof>*

**lemma** *induct-pcpl-2*:

$\llbracket P\ []; \bigwedge x. P\ [x]; \bigwedge x\ y. P\ [x, y]; \bigwedge x\ y\ z. P\ [x, y, z]; \bigwedge w\ x\ y\ z\ zs. P\ zs \Longrightarrow P\ (w\ \#\ x\ \#\ y\ \#\ z\ \#\ zs) \rrbracket \Longrightarrow P\ xs$

*<proof>*

**lemma** *dVs-empty[simp]*:  $dVs\ \{\} = \{\}$

*<proof>*

**lemma** *dVs-empty-iff[simp]*:  $dVs\ E = \{\} \longleftrightarrow E = \{\}$

*<proof>*

**lemma** *dVsI[intro]*:

**assumes**  $(a, v) \in dG$  **shows**  $a \in dVs\ dG$  **and**  $v \in dVs\ dG$

*<proof>*

**lemma** *dVsI'*:

**assumes**  $e \in dG$  **shows**  $fst\ e \in dVs\ dG$  **and**  $snd\ e \in dVs\ dG$

*<proof>*

**lemma** *dVs-union-distr[simp]*:  $dVs\ (G \cup H) = dVs\ G \cup dVs\ H$

*<proof>*

**lemma** *dVs-union-big-distr*:  $dVs\ (\bigcup G) = \bigcup (dVs\ ` G)$

*<proof>*

**lemma** *dVs-eq*:  $dVs\ E = fst\ ` E \cup snd\ ` E$

*<proof>*

**lemma** *finite-vertices-iff*:  $finite (dVs E) \longleftrightarrow finite E$  (**is** ?L  $\longleftrightarrow$  ?R)  
*<proof>*

**abbreviation** *reachable1* ::  $('v \times 'v) set \Rightarrow 'v \Rightarrow 'v \Rightarrow bool$  ( $- \rightarrow^+ 1 - [100,100]$  40) **where**  
 $reachable1 E u v \equiv (u,v) \in E^+$

**definition** *reachable* ::  $('v \times 'v) set \Rightarrow 'v \Rightarrow 'v \Rightarrow bool$  ( $- \rightarrow^* 1 - [100,100]$  40) **where**  
 $reachable E u v = ((u,v) \in rtrancl-on (dVs E) E)$

**lemma** *reachableE[elim?]*:  
**assumes**  $(u,v) \in E$   
**obtains**  $e$  **where**  $e \in E$   $e = (u, v)$   
*<proof>*

**lemma** *reachable-refl[intro!, Pure.intro!, simp]*:  $v \in dVs E \Longrightarrow v \rightarrow^* E v$   
*<proof>*

**lemma** *reachable-trans[trans,intro]*:  
**assumes**  $u \rightarrow^* E v$   $v \rightarrow^* E w$  **shows**  $u \rightarrow^* E w$   
*<proof>*

**lemma** *reachable-edge[dest,intro]*:  $(u,v) \in E \Longrightarrow u \rightarrow^* E v$   
*<proof>*

**lemma** *reachable-induct[consumes 1, case-names base step]*:  
**assumes** *major*:  $u \rightarrow^* E v$   
**and cases**:  $\llbracket u \in dVs E \rrbracket \Longrightarrow P u$   
 $\bigwedge x y. \llbracket u \rightarrow^* E x; (x,y) \in E; P x \rrbracket \Longrightarrow P y$   
**shows**  $P v$   
*<proof>*

**lemma** *converse-reachable-induct[consumes 1, case-names base step, induct pred: reachable]*:  
**assumes** *major*:  $u \rightarrow^* E v$   
**and cases**:  $v \in dVs E \Longrightarrow P v$   
 $\bigwedge x y. \llbracket (x,y) \in E; y \rightarrow^* E v; P y \rrbracket \Longrightarrow P x$   
**shows**  $P u$   
*<proof>*

**lemma** *reachable-in-dVs*:  
**assumes**  $u \rightarrow^* E v$   
**shows**  $u \in dVs E$   $v \in dVs E$   
*<proof>*

**lemma** *reachable1-in-dVs*:

**assumes**  $u \rightarrow^+ E v$   
**shows**  $u \in dVs E v \in dVs E$   
 $\langle proof \rangle$

**lemma** *reachable1-reachable*[*intro*]:  
 $v \rightarrow^+ E w \implies v \rightarrow^* E w$   
 $\langle proof \rangle$

**lemmas** *reachable1-reachableE*[*elim*] = *reachable1-reachable*[*elim-format*]

**lemma** *reachable-neq-reachable1*[*intro*]:  
**assumes** *reach*:  $v \rightarrow^* E w$   
**and** *neq*:  $v \neq w$   
**shows**  $v \rightarrow^+ E w$   
 $\langle proof \rangle$

**lemmas** *reachable-neq-reachable1E*[*elim*] = *reachable-neq-reachable1*[*elim-format*]

**lemma** *arc-implies-dominates*:  $e \in E \implies (fst e, snd e) \in E$   $\langle proof \rangle$

**definition** *neighbourhood*:: $(v \times v)$  set  $\implies v \implies v$  set **where**  
*neighbourhood*  $G u = \{v. (u,v) \in G\}$

**lemma**  
*neighbourhoodI*[*intro*]:  $v \in (neighbourhood G u) \implies (u,v) \in G$  **and**  
*neighbourhoodD*[*dest*]:  $(u,v) \in G \implies v \in (neighbourhood G u)$   
 $\langle proof \rangle$

**definition** *sources*  $G = \{u \mid \exists v. (u,v) \in G\}$

**definition** *sinks*  $G = \{v \mid \exists u. (u,v) \in G\}$

**lemma** *dVs-subset*:  $G \subseteq G' \implies dVs G \subseteq dVs G'$   
 $\langle proof \rangle$

**lemma** *dVs-insert*[*elim*]:  
 $v \in dVs (insert (x,y) G) \implies \llbracket v = x \implies P; v = y \implies P; v \in dVs G \implies P \rrbracket \implies P$   
 $\langle proof \rangle$

**lemma** *in-neighbourhood-dVs*[*simp, intro*]:  
 $v \in neighbourhood G u \implies v \in dVs G$   
 $\langle proof \rangle$

**lemma** *subset-neighbourhood-dVs*[*simp, intro*]:  
 $neighbourhood G u \subseteq dVs G$   
 $\langle proof \rangle$

**lemma** *in-dVsE*:  $v \in dVs\ G \implies \llbracket (\bigwedge u. (u, v) \in G \implies P); (\bigwedge u. (v, u) \in G \implies P) \rrbracket \implies P$

$v \notin dVs\ G \implies \llbracket (\bigwedge u. (u, v) \notin G); (\bigwedge u. (v, u) \notin G) \rrbracket \implies P \implies P$   
 ⟨proof⟩

**lemma** *neighbourhood-union[simp]*:  $neighbourhood\ (G \cup G')\ u = neighbourhood\ G\ u \cup neighbourhood\ G'\ u$

⟨proof⟩

**lemma** *vs-are-gen*:  $dVs\ (set\ E-impl) = set\ (map\ prod.fst\ E-impl) \cup set\ (map\ prod.snd\ E-impl)$

⟨proof⟩

**lemma** *dVs-swap*:  $dVs\ (prod.swap\ 'E) = dVs\ E$

⟨proof⟩

**end**

**theory** *Vwalk*

**imports** *Pair-Graph*

**begin**

## 2 A Theory on Vertex Walks in a Digraph

**context** *fixes*  $G :: 'v\ dgraph$  **begin**

**inductive** *vwalk* **where**

*vwalk0*:  $vwalk\ []\ |$

*vwalk1*:  $v \in dVs\ G \implies vwalk\ [v]\ |$

*vwalk2*:  $(u, v) \in G \implies vwalk\ (v\#\ vs) \implies vwalk\ (u\#\ v\#\ vs)$

**end**

**declare** *vwalk0[simp]*

**inductive-simps** *vwalk-1[simp]*:  $vwalk\ E\ [v]$

**inductive-simps** *vwalk-2[simp]*:  $vwalk\ E\ (u\ \#\ v\ \#\ vs)$

**definition** *vwalk-bet*:: $( 'v \times 'v )\ set \Rightarrow 'v \Rightarrow 'v\ list \Rightarrow 'v \Rightarrow bool$  **where**

*vwalk-bet*  $G\ u\ p\ v = ( vwalk\ G\ p \wedge p \neq [] \wedge hd\ p = u \wedge last\ p = v )$

**lemma** *vwalk-then-edge*:  $vwalk-bet\ dG\ u\ p\ v \implies u \neq v \implies (\exists v''. (u, v'') \in dG)$

⟨proof⟩

**lemma** *vwalk-then-in-dVs*:  $vwalk\ dG\ p \implies v \in set\ p \implies v \in dVs\ dG$

⟨proof⟩

**lemma** *vwalk-cons*:  $vwalk\ dG\ (v1\ \#\ v2\ \#\ p) \implies (v1, v2) \in dG$

⟨proof⟩

**lemma** *hd-of-vwalk-bet*:

*vwalk-bet*  $dG\ v\ p\ v' \implies \exists p'. p = v\ \#\ p'$

*<proof>*

**lemma** *hd-of-vwalk-bet'*:

$vwalk\text{-bet } dG\ v\ p\ v' \implies v = hd\ p$

*<proof>*

**lemma** *hd-of-vwalk-bet''*:  $vwalk\text{-bet } dG\ u\ p\ v \implies u \in set\ p$

*<proof>*

**lemma** *last-of-vwalk-bet*:

$vwalk\text{-bet } dG\ v\ p\ v' \implies v' = last\ p$

*<proof>*

**lemma** *last-of-vwalk-bet'*:

$vwalk\text{-bet } dG\ v\ p\ v' \implies \exists p'. p = p' @ [v']$

*<proof>*

**lemma** *append-vwalk-pref*:  $vwalk\ dG\ (p1 @ p2) \implies vwalk\ dG\ p1$

*<proof>*

**lemma** *append-vwalk-suff*:  $vwalk\ dG\ (p1 @ p2) \implies vwalk\ dG\ p2$

*<proof>*

**lemma** *append-vwalk*:  $vwalk\ dG\ p1 \implies vwalk\ dG\ p2 \implies (p1 \neq [] \implies p2 \neq [] \implies (last\ p1, hd\ p2) \in dG) \implies vwalk\ dG\ (p1 @ p2)$

*<proof>*

**lemma** *split-vwalk*:

$vwalk\text{-bet } dG\ v1\ (p1 @ v2 \# p2)\ v3 \implies vwalk\text{-bet } dG\ v2\ (v2 \# p2)\ v3$

*<proof>*

**lemma** *vwalk-bet-cons*:

$vwalk\text{-bet } dG\ v\ (v \# p)\ u \implies p \neq [] \implies vwalk\text{-bet } dG\ (hd\ p)\ p\ u$

*<proof>*

**lemma** *vwalk-bet-cons-2*:

$vwalk\text{-bet } dG\ v\ p\ v' \implies p \neq [] \implies vwalk\text{-bet } dG\ v\ (v \# tl\ p)\ v'$

*<proof>*

**lemma** *vwalk-bet-snoc*:

$vwalk\text{-bet } dG\ v'\ (p @ [v])\ v'' \implies v = v''$

*<proof>*

**lemma** *vwalk-bet-vwalk*:

$p \neq [] \implies vwalk\text{-bet } dG\ (hd\ p)\ p\ (last\ p) \longleftrightarrow vwalk\ dG\ p$

*<proof>*

**lemma** *vwalk-snoc-edge'*:  $vwalk\ dG\ (p @ [v]) \implies (v, v') \in dG \implies vwalk\ dG\ ((p @ [v]) @ [v'])$

*<proof>*

**lemma** *vwalk-snoc-edge*:  $vwalk\ dG\ (p\ @\ [v]) \implies (v, v') \in dG \implies vwalk\ dG\ (p\ @\ [v, v'])$   
 ⟨proof⟩

**lemma** *vwalk-snoc-edge-2*:  $vwalk\ dG\ (p\ @\ [v, v']) \implies (v, v') \in dG$   
 ⟨proof⟩

**lemma** *vwalk-append-edge*:  $vwalk\ dG\ (p1\ @\ p2) \implies p1 \neq [] \implies p2 \neq [] \implies (last\ p1, hd\ p2) \in dG$   
 ⟨proof⟩

**lemma** *vwalk-transitive-rel*:  
**assumes**  $(\bigwedge x\ y\ z. R\ x\ y \implies R\ y\ z \implies R\ x\ z) (\bigwedge v1\ v2. (v1, v2) \in dG \implies R\ v1\ v2)$   
**shows**  $vwalk\ dG\ (v\ \#\ vs) \implies v' \in set\ vs \implies R\ v\ v'$   
 ⟨proof⟩

**lemma** *vwalk-transitive-rel'*:  
**assumes**  $(\bigwedge x\ y\ z. R\ x\ y \implies R\ y\ z \implies R\ x\ z) (\bigwedge v1\ v2. (v1, v2) \in dG \implies R\ v1\ v2)$   
**shows**  $vwalk\ dG\ (vs\ @\ [v]) \implies v' \in set\ vs \implies R\ v'\ v$   
 ⟨proof⟩

**fun** *edges-of-vwalk* **where**  
*edges-of-vwalk* [] = [] |  
*edges-of-vwalk* [v] = [] |  
*edges-of-vwalk* (v#v'#l) = (v, v') # *edges-of-vwalk* (v'#l)

**lemma** *vwalk-ball-edges*:  $vwalk\ E\ p \implies b \in set\ (edges-of-vwalk\ p) \implies b \in E$   
 ⟨proof⟩

**lemma** *edges-of-vwalk-index*:  
 $Suc\ i < length\ p \implies edges-of-vwalk\ p\ !\ i = (p\ !\ i, p\ !\ Suc\ i)$   
 ⟨proof⟩

**lemma** *edges-of-vwalk-length*:  $length\ (edges-of-vwalk\ p) = length\ p - 1$   
 ⟨proof⟩

With the given assumptions we can only obtain an outgoing edge from  $v$ .

**lemma** *edges-of-vwalk-for-inner*:  
**assumes**  $p\ !\ i = v\ Suc\ i < length\ p$   
**obtains**  $w$  **where**  $edges-of-vwalk\ p\ !\ i = (v, w)$   
 ⟨proof⟩

For an incoming edge we need an additional assumption ( $0 < i$ ).

**lemma** *edges-of-vwalk-for-inner'*:

**assumes**  $p ! (Suc\ i) = v\ Suc\ i < length\ p$   
**obtains**  $u\ where\ (u, v) = edges-of-vwalk\ p ! i$   
 $\langle proof \rangle$

**lemma** *hd-edges-neq-last*:  
 $\llbracket (last\ p, hd\ p) \notin set\ (edges-of-vwalk\ p); hd\ p \neq last\ p; p \neq [] \rrbracket \implies$   
 $hd\ (edges-of-vwalk\ (last\ p \# p)) \neq last\ (edges-of-vwalk\ (last\ p \# p))$   
 $\langle proof \rangle$

**lemma** *v-in-edge-in-vwalk*:  
**assumes**  $(u, v) \in set\ (edges-of-vwalk\ p)$   
**shows**  $u \in set\ p\ v \in set\ p$   
 $\langle proof \rangle$

**lemma** *distinct-edges-of-vwalk*:  
 $distinct\ p \implies distinct\ (edges-of-vwalk\ p)$   
 $\langle proof \rangle$

**lemma** *distinct-edges-of-vwalk-cons*:  
 $distinct\ (edges-of-vwalk\ (v \# p)) \implies distinct\ (edges-of-vwalk\ p)$   
 $\langle proof \rangle$

**lemma** *tl-vwalk-is-vwalk*:  $vwalk\ E\ p \implies vwalk\ E\ (tl\ p)$   
 $\langle proof \rangle$

**lemma** *vwalk-concat*:  
**assumes**  $vwalk\ E\ p\ vwalk\ E\ q\ q \neq []\ p \neq [] \implies last\ p = hd\ q$   
**shows**  $vwalk\ E\ (p @ tl\ q)$   
 $\langle proof \rangle$

**lemma** *edges-of-vwalk-append-2*:  
 $p' \neq [] \implies edges-of-vwalk\ (p @ p') = edges-of-vwalk\ (p @ [hd\ p']) @ edges-of-vwalk\ p'$   
 $\langle proof \rangle$

**lemma** *edges-of-vwalk-append*:  $\exists ep. edges-of-vwalk\ (p @ p') = ep @ edges-of-vwalk\ p'$   
 $\langle proof \rangle$

**lemma** *append-butlast-last-cancel*:  $p \neq [] \implies butlast\ p @ last\ p \# p' = p @ p'$   
 $\langle proof \rangle$

**lemma** *edges-of-vwalk-append-3*:  
**assumes**  $p \neq []$   
**shows**  $edges-of-vwalk\ (p @ p') = edges-of-vwalk\ p @ edges-of-vwalk\ (last\ p \# p')$   
 $\langle proof \rangle$

**lemma** *vwalk-vertex-has-edge*:

**assumes**  $\text{length } p \geq 2 \ v \in \text{set } p$   
**obtains**  $e \ u$  **where**  $e \in \text{set } (\text{edges-of-vwalk } p) \ e = (u, v) \vee e = (v, u)$   
 $\langle \text{proof} \rangle$

**lemma** *v-in-edge-in-vwalk-inj*:  
**assumes**  $e \in \text{set } (\text{edges-of-vwalk } p)$   
**obtains**  $u \ v$  **where**  $e = (u, v)$   
 $\langle \text{proof} \rangle$

**lemma** *v-in-edge-in-vwalk-gen*:  
**assumes**  $e \in \text{set } (\text{edges-of-vwalk } p) \ e = (u, v)$   
**shows**  $u \in \text{set } p \ v \in \text{set } p$   
 $\langle \text{proof} \rangle$

**lemma** *edges-of-vwalk-dVs*:  $dVs (\text{set } (\text{edges-of-vwalk } p)) \subseteq \text{set } p$   
 $\langle \text{proof} \rangle$

**lemma** *last-v-snd-last-e*:  
**assumes**  $\text{length } p \geq 2$   
**shows**  $\text{last } p = \text{snd } (\text{last } (\text{edges-of-vwalk } p))$  — is this the best formulation for this?  
 $\langle \text{proof} \rangle$

**lemma** *hd-v-fst-hd-e*:  
**assumes**  $\text{length } p \geq 2$   
**shows**  $\text{hd } p = \text{fst } (\text{hd } (\text{edges-of-vwalk } p))$   
 $\langle \text{proof} \rangle$

**lemma** *last-in-edge*:  
 $p \neq [] \implies \exists u. (u, \text{last } p) \in \text{set } (\text{edges-of-vwalk } (v \# p)) \wedge u \in \text{set } (v \# p)$   
 $\langle \text{proof} \rangle$

**lemma** *edges-of-vwalk-append-subset*:  
**shows**  $\text{set } (\text{edges-of-vwalk } p') \subseteq \text{set } (\text{edges-of-vwalk } (p @ p'))$   
 $\langle \text{proof} \rangle$

**lemma** *nonempty-vwalk-vwalk-bet[intro?]*:  
**assumes**  $\text{vwalk } E \ p \ p \neq [] \ \text{hd } p = u \ \text{last } p = v$   
**shows**  $\text{vwalk-bet } E \ u \ p \ v$   
 $\langle \text{proof} \rangle$

**lemma** *vwalk-bet-nonempty*:  
**assumes**  $\text{vwalk-bet } E \ u \ p \ v$   
**shows**  $[\text{simp}]$ :  $p \neq []$   
 $\langle \text{proof} \rangle$

**lemma** *vwalk-bet-nonempty-vwalk[elim]*:  
**assumes**  $\text{vwalk-bet } E \ u \ p \ v$   
**shows**  $\text{vwalk } E \ p \ p \neq [] \ \text{hd } p = u \ \text{last } p = v$

*<proof>*

**lemma** *vwalk-bet-reflexive*[*intro*]:

**assumes**  $w \in dVs\ E$

**shows** *vwalk-bet*  $E\ w\ [w]\ w$

*<proof>*

**lemma** *singleton-hd-last*:  $q \neq [] \implies tl\ q = [] \implies hd\ q = last\ q$

*<proof>*

**lemma** *singleton-hd-last'*:  $q \neq [] \implies butlast\ q = [] \implies hd\ q = last\ q$

*<proof>*

**lemma** *vwalk-bet-transitive*:

**assumes** *vwalk-bet*  $E\ u\ p\ v$  *vwalk-bet*  $E\ v\ q\ w$

**shows** *vwalk-bet*  $E\ u\ (p\ @\ tl\ q)\ w$

*<proof>*

**lemma** *vwalk-bet-in-dVs*:

**assumes** *vwalk-bet*  $E\ a\ p\ b$

**shows**  $set\ p \subseteq dVs\ E$

*<proof>*

**lemma** *vwalk-bet-endpoints*:

**assumes** *vwalk-bet*  $E\ u\ p\ v$

**shows** [*simp*]:  $u \in dVs\ E$

**and** [*simp*]:  $v \in dVs\ E$

*<proof>*

**lemma** *vwalk-bet-pref*:

**assumes** *vwalk-bet*  $E\ u\ (pr\ @\ [x]\ @\ su)\ v$

**shows** *vwalk-bet*  $E\ u\ (pr\ @\ [x])\ x$

*<proof>*

**lemma** *vwalk-bet-suff*:

**assumes** *vwalk-bet*  $E\ u\ (pr\ @\ [x]\ @\ su)\ v$

**shows** *vwalk-bet*  $E\ x\ (x\ \#\ su)\ v$

*<proof>*

**lemma** *edges-are-vwalk-bet*:

**assumes**  $(v, w) \in E$

**shows** *vwalk-bet*  $E\ v\ [v, w]\ w$

*<proof>*

**lemma** *induct-vwalk-bet*[*case-names* *path1 path2*, *consumes* *1*, *induct set*: *vwalk-bet*]:

**assumes** *vwalk-bet*  $E\ a\ p\ b$

**assumes** *Path1*:  $\bigwedge v. v \in dVs\ E \implies P\ E\ [v]\ v\ v$

**assumes** *Path2*:  $\bigwedge v\ v'\ vs\ b. (v, v') \in E \implies \text{vwalk-bet}\ E\ v'\ (v'\ \#\ vs)\ b \implies P\ E\ (v'\ \#\ vs)\ v'\ b \implies P\ E\ (v\ \#\ v'\ \#\ vs)\ v\ b$

**shows**  $P E p a b$   
 $\langle proof \rangle$

**lemma** *vwalk-append*:  
**assumes**  $vwalk E xs vwalk E ys last xs = hd ys$   
**shows**  $vwalk E (xs @ tl ys)$   
 $\langle proof \rangle$

**lemma** *vwalk-append2*:  
**assumes**  $vwalk E (xs @ [x]) vwalk E (x \# ys)$   
**shows**  $vwalk E (xs @ x \# ys)$   
 $\langle proof \rangle$

**lemma** *vwalk-appendD-last*:  
 $vwalk E (xs @ [x, y]) \implies vwalk E (xs @ [x])$   
 $\langle proof \rangle$

**lemma** *vwalk-ConsD*:  
 $vwalk E (x \# xs) \implies vwalk E xs$   
 $\langle proof \rangle$

**lemmas**  $vwalkD = vwalk-ConsD \text{ append-vwalk-pref append-vwalk-suff}$

**lemma** *vwalk-alt-induct*[*consumes 1, case-names Single Snoc*]:  
**assumes**  
 $vwalk E p P [] (\bigwedge x. P [x])$   
 $\bigwedge y x xs. (y, x) \in E \implies vwalk E (xs @ [y]) \implies P (xs @ [y]) \implies P (xs @ [y,$   
 $x])$   
**shows**  $P p$   
 $\langle proof \rangle$

**lemma** *vwalk-append-single*:  
**assumes**  $vwalk E p (last p, x) \in E$   
**shows**  $vwalk E (p @ [x])$   
 $\langle proof \rangle$

**lemmas**  $vwalk-decomp = \text{append-vwalk-pref append-vwalk-suff vwalk-append-edge}$

**lemma** *vwalk-rotate*:  
**assumes**  $vwalk E (x \# xs @ y \# ys @ [x])$   
**shows**  $vwalk E (y \# ys @ x \# xs @ [y])$   
 $\langle proof \rangle$

**lemma** *vwalk-bet-nonempty*'[*simp*]:  $\neg vwalk-bet E u [] v$   
 $\langle proof \rangle$

**lemma** *vwalk-ConsE*:  
**assumes**  $vwalk E (a \# p) p \neq []$   
**obtains**  $e$  **where**  $e \in E e = (a, hd p) vwalk E p$

$\langle proof \rangle$

**lemma** *vwalk-reachable*:

$p \neq [] \implies vwalk\ E\ p \implies hd\ p \rightarrow^*_E\ last\ p$   
 $\langle proof \rangle$

**lemma** *vwalk-reachable'*:

$vwalk\ E\ p \implies p \neq [] \implies hd\ p = u \implies last\ p = v \implies u \rightarrow^*_E\ v$   
 $\langle proof \rangle$

**lemma** *vwalkI*:  $(x, hd\ p) \in E \implies vwalk\ E\ p \implies vwalk\ E\ (x\#\!p)$

$\langle proof \rangle$

**lemma** *reachable-vwalk*:

**assumes**  $u \rightarrow^*_E\ v$   
**shows**  $\exists p. hd\ p = u \wedge last\ p = v \wedge vwalk\ E\ p \wedge p \neq []$   
 $\langle proof \rangle$

**lemma** *reachable-vwalk-iff*:

$u \rightarrow^*_E\ v \iff (\exists p. hd\ p = u \wedge last\ p = v \wedge vwalk\ E\ p \wedge p \neq [])$   
 $\langle proof \rangle$

**lemma** *reachable-vwalk-bet-iff*:

$u \rightarrow^*_E\ v \iff (\exists p. vwalk\ bet\ E\ u\ p\ v)$   
 $\langle proof \rangle$

**lemma** *reachable-vwalk-betD*:

$vwalk\ bet\ E\ u\ p\ v \implies u \rightarrow^*_E\ v$   
 $\langle proof \rangle$

**lemma** *vwalk-reachable1*:

$vwalk\ E\ (u\ \#\!p\ @\ [v]) \implies u \rightarrow^+_E\ v$   
 $\langle proof \rangle$

**lemma** *reachable1-vwalk*:

**assumes**  $u \rightarrow^+_E\ v$   
**shows**  $\exists p. vwalk\ E\ (u\ \#\!p\ @\ [v])$   
 $\langle proof \rangle$

**lemma** *reachable1-vwalk-iff*:

$u \rightarrow^+_E\ v \iff (\exists p. vwalk\ E\ (u\ \#\!p\ @\ [v]))$   
 $\langle proof \rangle$

**lemma** *reachable-vwalk-iff2*:

$u \rightarrow^*_E\ v \iff (u = v \wedge u \in dVs\ E \vee (\exists p. vwalk\ E\ (u\ \#\!p\ @\ [v])))$   
 $\langle proof \rangle$

**lemma** *vwalk-remove-cycleE*:

**assumes**  $vwalk\ E\ (u\ \#\!p\ @\ [v])$

**obtains**  $p'$  **where**  $vwalk\ E\ (u\ \# \ p' \ @ \ [v])$   
*distinct*  $p' \ u \notin\ set\ p' \ v \notin\ set\ p' \ set\ p' \subseteq\ set\ p$   
*<proof>*

**abbreviation**  $closed\text{-}vwalk\text{-}bet :: ('v \times 'v)\ set \Rightarrow 'v\ list \Rightarrow 'v \Rightarrow bool$  **where**  
 $closed\text{-}vwalk\text{-}bet\ E\ c\ v \equiv vwalk\text{-}bet\ E\ v\ c\ v \wedge Suc\ 0 < length\ c$

**lemma** *edge-iff-vwalk-bet*:  $(u, v) \in E = vwalk\text{-}bet\ E\ u\ [u, v]\ v$   
*<proof>*

**lemma** *vwalk-bet-in-vertices*:  $vwalk\text{-}bet\ E\ u\ p\ v \Longrightarrow w \in set\ p \Longrightarrow w \in dVs\ E$   
*<proof>*

**lemma** *vwalk-bet-hd-neq-last-implies-edges-nonempty*:  
**assumes**  $vwalk\text{-}bet\ E\ u\ p\ v$   
**assumes**  $u \neq v$   
**shows**  $E \neq \{\}$   
*<proof>*

**lemma** *vwalk-bet-edges-in-edges*:  $vwalk\text{-}bet\ E\ u\ p\ v \Longrightarrow set\ (edges\text{-}of\text{-}vwalk\ p) \subseteq E$   
*<proof>*

**lemma** *vwalk-bet-prefix-is-vwalk-bet*:  
**assumes**  $p \neq []$   
**assumes**  $vwalk\text{-}bet\ E\ u\ (p\ @\ q)\ v$   
**shows**  $vwalk\text{-}bet\ E\ u\ p\ (last\ p)$   
*<proof>*

**lemma** *vwalk-bet-suffix-is-vwalk-bet*:  
**assumes**  $q \neq []$   
**assumes**  $vwalk\text{-}bet\ E\ u\ (p\ @\ q)\ v$   
**shows**  $vwalk\text{-}bet\ E\ (hd\ q)\ q\ v$   
*<proof>*

**lemma** *vwalk-bet-append-append-is-vwalk-bet*:  
**assumes**  $vwalk\text{-}bet\ E\ u\ p\ v$   
**assumes**  $vwalk\text{-}bet\ E\ v\ q\ w$   
**assumes**  $vwalk\text{-}bet\ E\ w\ r\ x$   
**shows**  $vwalk\text{-}bet\ E\ u\ (p\ @\ tl\ q\ @\ tl\ r)\ x$   
*<proof>*

**lemma**  
**assumes**  $p \neq []$   
**shows**  $edges\text{-}of\text{-}vwalk\ (p\ @\ q) = edges\text{-}of\text{-}vwalk\ p\ @\ edges\text{-}of\text{-}vwalk\ ([last\ p]\ @\ q)$   
*<proof>*

**fun** *is-vwalk-bet-vertex-decomp* ::  $('v \times 'v)\ set \Rightarrow 'v\ list \Rightarrow 'v \Rightarrow 'v\ list \times 'v\ list \Rightarrow bool$  **where**  
 $is\text{-}vwalk\text{-}bet\text{-}vertex\text{-}decomp\ E\ p\ v\ (q, r) \longleftrightarrow p = q\ @\ tl\ r \wedge (\exists u\ w. vwalk\text{-}bet\ E$

$u q v \wedge \text{vwalk-bet } E v r w)$

**definition**  $\text{vwalk-bet-vertex-decomp} :: ('v \times 'v) \text{ set} \Rightarrow 'v \text{ list} \Rightarrow 'v \Rightarrow 'v \text{ list} \times 'v \text{ list}$  **where**

$\text{vwalk-bet-vertex-decomp } E p v = (\text{SOME } qr. \text{is-vwalk-bet-vertex-decomp } E p v qr)$

**lemma**  $\text{vwalk-bet-vertex-decomp}E$ :

**assumes**  $p\text{-vwalk}$ :  $\text{vwalk-bet } E u p v$

**assumes**  $p\text{-decomp}$ :  $p = xs @ y \# ys$

**obtains**  $q r$  **where**

$p = q @ tl r$

$q = xs @ [y]$

$r = y \# ys$

$\text{vwalk-bet } E u q y$

$\text{vwalk-bet } E y r v$

$\langle \text{proof} \rangle$

**lemma**  $\text{vwalk-bet-vertex-decomp-is-vwalk-bet-vertex-decomp}$ :

**assumes**  $p\text{-vwalk}$ :  $\text{vwalk-bet } E u p w$

**assumes**  $v\text{-in-}p$ :  $v \in \text{set } p$

**shows**  $\text{is-vwalk-bet-vertex-decomp } E p v (\text{vwalk-bet-vertex-decomp } E p v)$

$\langle \text{proof} \rangle$

**lemma**  $\text{vwalk-bet-vertex-decomp}E\text{-}2$ :

**assumes**  $p\text{-vwalk}$ :  $\text{vwalk-bet } E u p w$

**assumes**  $v\text{-in-}p$ :  $v \in \text{set } p$

**assumes**  $qr\text{-def}$ :  $\text{vwalk-bet-vertex-decomp } E p v = (q, r)$

**obtains**

$p = q @ tl r$

$\text{vwalk-bet } E u q v$

$\text{vwalk-bet } E v r w$

$\langle \text{proof} \rangle$

**definition**  $\text{vtrail} :: ('v \times 'v) \text{ set} \Rightarrow 'v \Rightarrow 'v \text{ list} \Rightarrow 'v \Rightarrow \text{bool}$  **where**

$\text{vtrail } E u p v = (\text{vwalk-bet } E u p v \wedge \text{distinct } (\text{edges-of-vwalk } p))$

**abbreviation**  $\text{closed-vtrail} :: ('v \times 'v) \text{ set} \Rightarrow 'v \text{ list} \Rightarrow 'v \Rightarrow \text{bool}$  **where**

$\text{closed-vtrail } E c v \equiv \text{vtrail } E v c v \wedge \text{Suc } 0 < \text{length } c$

**lemma**  $\text{closed-vtrail-implies-Cons}$ :

**assumes**  $\text{closed-vtrail } E c v$

**shows**  $c = v \# tl c$

$\langle \text{proof} \rangle$

**lemma**  $tl\text{-non-empty-conv}$ :

**shows**  $tl l \neq [] \longleftrightarrow \text{Suc } 0 < \text{length } l$

*<proof>*

**lemma** *closed-vtrail-implies-tl-nonempty:*

**assumes** *closed-vtrail*  $E\ c\ v$

**shows**  $tl\ c \neq []$

*<proof>*

**lemma** *vtrail-in-vertices:*

*vtrail*  $E\ u\ p\ v \implies w \in set\ p \implies w \in dVs\ E$

*<proof>*

**lemma**

**assumes** *vtrail*  $E\ u\ p\ v$

**shows** *vtrail-hd-in-vertices:*  $u \in dVs\ E$

**and** *vtrail-last-in-vertices:*  $v \in dVs\ E$

*<proof>*

**lemma** *list-set-tl:*  $x \in set\ (tl\ xs) \implies x \in set\ xs$

*<proof>*

**lemma** *closed-vtrail-hd-tl-in-vertices:*

**assumes** *closed-vtrail*  $E\ c\ v$

**shows**  $hd\ (tl\ c) \in dVs\ E$

*<proof>*

**lemma** *vtrail-prefix-is-vtrail:*

**notes** *vtrail-def* [*simp*]

**assumes**  $p \neq []$

**assumes** *vtrail*  $E\ u\ (p\ @\ q)\ v$

**shows** *vtrail*  $E\ u\ p\ (last\ p)$

*<proof>*

**lemma** *vtrail-suffix-is-vtrail:*

**notes** *vtrail-def* [*simp*]

**assumes**  $q \neq []$

**assumes** *vtrail*  $E\ u\ (p\ @\ q)\ v$

**shows** *vtrail*  $E\ (hd\ q)\ q\ v$

*<proof>*

**definition** *distinct-vwalk-bet* ::  $( 'v \times 'v )\ set \Rightarrow 'v \Rightarrow 'v\ list \Rightarrow 'v \Rightarrow bool$  **where**  
 $distinct-vwalk-bet\ E\ u\ p\ v = ( vwalk-bet\ E\ u\ p\ v \wedge distinct\ p )$

**lemma** *distinct-vwalk-bet-length-le-card-vertices:*

**assumes** *distinct-vwalk-bet*  $E\ u\ p\ v$

**assumes** *finite*  $E$

**shows**  $length\ p \leq card\ (dVs\ E)$

*<proof>*

**lemma** *distinct-vwalk-bet-triples-finite:*

**assumes** *finite E*  
**shows** *finite*  $\{(p, u, v). \text{distinct-vwalk-bet } E \ u \ p \ v\}$   
 $\langle \text{proof} \rangle$

**lemma** *distinct-vwalk-bets-finite*:  
*finite E*  $\implies$  *finite*  $\{p. \text{distinct-vwalk-bet } E \ u \ p \ v\}$   
 $\langle \text{proof} \rangle$

### 3 Vwalks to paths (as opposed to arc walks (*awalk-to-apat* before))

**fun** *is-closed-decomp* ::  $('v \times 'v) \text{ set} \Rightarrow 'v \text{ list} \Rightarrow 'v \text{ list} \times 'v \text{ list} \times 'v \text{ list} \Rightarrow \text{bool}$   
**where**

*is-closed-decomp*  $E \ p \ (q, r, s) \longleftrightarrow$   
 $p = q \ @ \ tl \ r \ @ \ tl \ s \ \wedge$   
 $(\exists u \ v \ w. \text{vwalk-bet } E \ u \ q \ v \ \wedge \ \text{closed-vwalk-bet } E \ r \ v \ \wedge \ \text{vwalk-bet } E \ v \ s \ w) \ \wedge$   
*distinct*  $q$

**definition** *closed-vwalk-bet-decomp* ::  $('v \times 'v) \text{ set} \Rightarrow 'v \text{ list} \Rightarrow 'v \text{ list} \times 'v \text{ list} \times 'v \text{ list}$  **where**

*closed-vwalk-bet-decomp*  $E \ p = (\text{SOME } \text{qrs}. \text{is-closed-decomp } E \ p \ \text{qrs})$

**lemma** *closed-vwalk-bet-decompE*:  
**assumes** *p-vwalk*: *vwalk-bet*  $E \ u \ p \ v$   
**assumes** *p-decomp*:  $p = xs \ @ \ y \ \# \ ys \ @ \ y \ \# \ zs$   
**obtains**  $q \ r \ s$  **where**  
 $p = q \ @ \ tl \ r \ @ \ tl \ s$   
 $q = xs \ @ \ [y]$   
 $r = y \ \# \ ys \ @ \ [y]$   
 $s = y \ \# \ zs$   
*vwalk-bet*  $E \ u \ q \ y$   
*vwalk-bet*  $E \ y \ r \ y$   
*vwalk-bet*  $E \ y \ s \ v$   
 $\langle \text{proof} \rangle$

**lemma** *closed-vwalk-bet-decomp-is-closed-decomp*:  
**assumes** *p-vwalk*: *vwalk-bet*  $E \ u \ p \ v$   
**assumes** *p-not-distinct*:  $\neg \text{distinct } p$   
**shows** *is-closed-decomp*  $E \ p$  (*closed-vwalk-bet-decomp*  $E \ p$ )  
 $\langle \text{proof} \rangle$

**lemma** *closed-vwalk-bet-decompE-2*:  
**assumes** *p-vwalk*: *vwalk-bet*  $E \ u \ p \ v$   
**assumes** *p-not-distinct*:  $\neg \text{distinct } p$   
**assumes** *qrs-def*: *closed-vwalk-bet-decomp*  $E \ p = (q, r, s)$   
**obtains**  
 $p = q \ @ \ tl \ r \ @ \ tl \ s$   
 $\exists w. \text{vwalk-bet } E \ u \ q \ w \ \wedge \ \text{closed-vwalk-bet } E \ r \ w \ \wedge \ \text{vwalk-bet } E \ w \ s \ v$

*distinct q*  
⟨*proof*⟩

**function** *vwalk-bet-to-distinct* :: ('v × 'v) set ⇒ 'v list ⇒ 'v list **where**  
  *vwalk-bet-to-distinct E p* =  
    (*if* (∃ u v. *vwalk-bet E u p v*) ∧ ¬ *distinct p*  
      *then let* (q, r, s) = *closed-vwalk-bet-decomp E p in vwalk-bet-to-distinct E* (q  
  @ *tl s*)  
      *else p*)  
  ⟨*proof*⟩

**termination** *vwalk-bet-to-distinct*  
⟨*proof*⟩

**lemma** *vwalk-bet-to-distinct-induct* [*consumes 1, case-names path decomp*]:  
  **assumes** *vwalk-bet E u p v*  
  **assumes** *distinct*: ∧*p. [[ vwalk-bet E u p v; distinct p ] ⇒ P p*  
  **assumes**  
    *decomp*: ∧*p q r s. [[ vwalk-bet E u p v; ¬ distinct p;*  
      *closed-vwalk-bet-decomp E p = (q, r, s); P (q @ tl s) ] ⇒ P p*  
  **shows** *P p*  
  ⟨*proof*⟩

**lemma** *vwalk-bet-to-distinct-is-distinct-vwalk-bet*:  
  **assumes** *vwalk-bet E u p v*  
  **shows** *distinct-vwalk-bet E u (vwalk-bet-to-distinct E p) v*  
  ⟨*proof*⟩

**lemma** *vwalk-betE[elim?]*:  
  **assumes** *vwalk-bet E u p v*  
  **assumes** *singleton*: [[ *v* ∈ *dVs E*; *u = v* ] ⇒ *P*  
  **assumes**  
    *step*: ∧*p' x. [[ p = u#x#p'; (u,x) ∈ E; vwalk-bet E x (x#p') v ] ⇒ P*  
  **shows** *P*  
  ⟨*proof*⟩

**lemma** *vwalk-subset*:  
  [[*vwalk G p*; *G ⊆ G'*] ⇒ *vwalk G' p*  
  ⟨*proof*⟩

**lemma** *vwalk-bet-subset*:  
  [[*vwalk-bet G u p v*; *G ⊆ G'*] ⇒ *vwalk-bet G' u p v*  
  ⟨*proof*⟩

**lemma** *reachable-subset[intro]*:  
  [[*u →\*<sub>G</sub> v*; *G ⊆ G'*] ⇒ *u →\*<sub>G'</sub> v*  
  ⟨*proof*⟩

**lemma** *reachable-Union-1[intro]*:

$$\begin{aligned} \llbracket u \rightarrow^* G v \rrbracket &\Longrightarrow u \rightarrow^* G \cup G' v \\ \llbracket u \rightarrow^* G v \rrbracket &\Longrightarrow u \rightarrow^* G' \cup G v \\ \langle \text{proof} \rangle \end{aligned}$$

**lemma** *reachableE[elim?]*:  
**assumes** *reachable E u v*  
**assumes** *singleton:  $\llbracket v \in dVs E; u = v \rrbracket \Longrightarrow P$*   
**assumes**  
*step:  $\bigwedge x. \llbracket (u,x) \in E; \text{reachable } E x v \rrbracket \Longrightarrow P$*   
**shows** *P*  
 $\langle \text{proof} \rangle$

**lemma** *vwalk-insertE[case-names nil sing1 sing2 in-e in-E]*:  
 $\llbracket \text{vwalk } (insert\ e\ E) p; (p = [] \Longrightarrow P); (\bigwedge u\ v. p = [v] \Longrightarrow e = (u,v) \Longrightarrow P); (\bigwedge u\ v. p = [v] \Longrightarrow e = (v,u) \Longrightarrow P); (\bigwedge v. p = [v] \Longrightarrow v \in dVs\ E \Longrightarrow P); (\bigwedge p'\ v1\ v2. \llbracket \text{vwalk } \{e\} [v1, v2]; \text{vwalk } (insert\ e\ E) (v2 \# p'); p = v1 \# v2 \# p' \rrbracket \Longrightarrow P); (\bigwedge p'\ v1\ v2. \llbracket \text{vwalk } E [v1, v2]; \text{vwalk } (insert\ e\ E) (v2 \# p'); p = v1 \# v2 \# p' \rrbracket \Longrightarrow P) \rrbracket \Longrightarrow P$   
 $\langle \text{proof} \rangle$

A lemma which allows for case splitting over paths when doing induction on graph edges.

**lemma** *vwalk-bet-insertE[case-names nil sing1 sing2 in-e in-E]*:  
 $\llbracket \text{vwalk-bet } (insert\ e\ E) v1\ p\ v2; (\llbracket v1 \in dVs (insert\ e\ E); v1 = v2; p = [] \rrbracket \Longrightarrow P); (\bigwedge u\ v. p = [u] \Longrightarrow e = (u,v) \Longrightarrow P); (\bigwedge u\ v. p = [v] \Longrightarrow e = (u,v) \Longrightarrow P); (\bigwedge v. p = [v] \Longrightarrow v = v1 \Longrightarrow v = v2 \Longrightarrow v \in dVs\ E \Longrightarrow P); (\bigwedge p'\ v3. \llbracket \text{vwalk-bet } \{e\} v1 [v1, v3] v3; \text{vwalk-bet } (insert\ e\ E) v3 (v3 \# p') v2; p = v1 \# v3 \# p' \rrbracket \Longrightarrow P); (\bigwedge p'\ v3. \llbracket \text{vwalk-bet } E v1 [v1, v3] v3; \text{vwalk-bet } (insert\ e\ E) v3 (v3 \# p') v2; p = v1 \# v3 \# p' \rrbracket \Longrightarrow P) \rrbracket \Longrightarrow P$   
 $\langle \text{proof} \rangle$

**find-theorems** *name: induct vwalk-bet*

**lemma** *vwalk-bet2[simp]*:  
 $\text{vwalk-bet } G\ u\ (u \# v \# vs)\ b \longleftrightarrow ((u,v) \in G \wedge \text{vwalk-bet } G\ v\ (v \# vs)\ b)$   
 $\langle \text{proof} \rangle$

**lemma** *butlast-vwalk-is-vwalk*:  $vwalk\ E\ p \implies vwalk\ E\ (butlast\ p)$   
 ⟨proof⟩

**lemma** *vwalk-concat-2*:

**assumes**  $vwalk\ E\ p\ vwalk\ E\ q\ q \neq []\ p \neq [] \implies last\ p = hd\ q$

**shows**  $vwalk\ E\ (butlast\ p\ @\ q)$

⟨proof⟩

**lemma** *vwalk-bet-transitive-2*:

**assumes**  $vwalk\ bet\ E\ u\ p\ v\ vwalk\ bet\ E\ v\ q\ w$

**shows**  $vwalk\ bet\ E\ u\ (butlast\ p\ @\ q)\ w$

⟨proof⟩

**lemma** *vwalk-not-vwalk*:

$\llbracket vwalk\ G\ p; \neg vwalk\ G'\ p \rrbracket \implies$

$(\exists (u,v) \in set\ (edges\ of\ vwalk\ p). (u,v) \in (G - G')) \vee$

$(\exists v \in set\ p. v \in (dVs\ G - dVs\ G'))$

⟨proof⟩

**lemma** *vwalk-not-vwalk-2*:

$\llbracket vwalk\ G\ p; \neg vwalk\ G'\ p; length\ p \geq 2 \rrbracket \implies$

$(\exists (u,v) \in set\ (edges\ of\ vwalk\ p). (u,v) \in (G - G'))$

⟨proof⟩

**lemma** *vwalk-not-vwalk-elim*:

$\llbracket vwalk\ G\ p; \neg vwalk\ G'\ p \rrbracket \implies$

$\llbracket \bigwedge u\ v. \llbracket (u,v) \in set\ (edges\ of\ vwalk\ p); (u,v) \in (G - G') \rrbracket \implies P; \bigwedge v. \llbracket v \in set\ p; v \in (dVs\ G - dVs\ G') \rrbracket \implies P \rrbracket \implies P$

⟨proof⟩

**lemma** *vwalk-not-vwalk-elim-2*:

$\llbracket vwalk\ G\ p; \neg vwalk\ G'\ p; length\ p \geq 2 \rrbracket \implies$

$\llbracket \bigwedge u\ v. \llbracket (u,v) \in set\ (edges\ of\ vwalk\ p); (u,v) \in (G - G') \rrbracket \implies P \rrbracket \implies P$

⟨proof⟩

**lemma** *vwalk-bet-not-vwalk-bet*:

$\llbracket vwalk\ bet\ G\ u\ p\ v; \neg vwalk\ bet\ G'\ u\ p\ v \rrbracket \implies$

$(\exists (u,v) \in set\ (edges\ of\ vwalk\ p). (u,v) \in (G - G')) \vee$

$(\exists v \in set\ p. v \in (dVs\ G - dVs\ G'))$

⟨proof⟩

**lemma** *vwalk-bet-not-vwalk-bet-elim*:

$\llbracket vwalk\ bet\ G\ u\ p\ v; \neg vwalk\ bet\ G'\ u\ p\ v \rrbracket \implies$

$\llbracket \bigwedge u\ v. \llbracket (u,v) \in set\ (edges\ of\ vwalk\ p); (u,v) \in (G - G') \rrbracket \implies P; \bigwedge v. \llbracket v \in set\ p; v \in (dVs\ G - dVs\ G') \rrbracket \implies P \rrbracket \implies P$

⟨proof⟩

$\langle proof \rangle$

**lemma** *vwalk-bet-not-vwalk-bet-elim-2*:

$\llbracket vwalk\text{-bet } G \ u \ p \ v; \neg vwalk\text{-bet } G' \ u \ p \ v; \text{length } p \geq 2 \rrbracket \implies$   
 $\llbracket \bigwedge u \ v. \llbracket (u,v) \in \text{set } (\text{edges-of-vwalk } p); (u,v) \in (G - G') \rrbracket \implies P \rrbracket \implies P$   
 $\langle proof \rangle$

**lemma** *vwalk-bet-props*:

$vwalk\text{-bet } G \ u \ p \ v \implies (\llbracket vwalk \ G \ p; p \neq []; hd \ p = u; last \ p = v \rrbracket \implies P) \implies P$   
 $\langle proof \rangle$

**lemma** *no-outgoing-last*:

$\llbracket vwalk \ G \ p; \bigwedge v. (u,v) \notin G; u \in \text{set } p \rrbracket \implies last \ p = u$   
 $\langle proof \rangle$

**lemma** *not-vwalk-bet-empty*:  $\neg Vwalk.vwalk\text{-bet } \{ \} \ u \ p \ v$

$\langle proof \rangle$

**lemma** *edges-in-vwalk-split*:

$(u, v) \in \text{set } (\text{edges-of-vwalk } p) \implies \exists \ p1 \ p2. p = p1 \ @[u,v]@p2$   
 $\langle proof \rangle$

**end**

**theory** *Enat-Misc*

**imports** *Main HOL-Library.Extended-Nat*

**begin**

**declare** *one-enat-def*

**declare** *zero-enat-def*

**lemma** *eval-enat-numeral*:

$\text{numeral } Num.One = eSuc \ 0$   
 $\text{numeral } (Num.Bit0 \ n) = eSuc \ (\text{numeral } (Num.BitM \ n))$   
 $\text{numeral } (Num.Bit1 \ n) = eSuc \ (\text{numeral } (Num.Bit0 \ n))$   
 $\langle proof \rangle$

**declare** *eSuc-enat[symmetric, simp]*

**end**

**theory** *Dist*

**imports** *Enat-Misc Vwalk*

**begin**

## 4 Distances

### 4.1 Distance from a vertex

**definition** *distance*::('v × 'v) set ⇒ 'v ⇒ 'v ⇒ enat **where**

*distance* G u v = ( INF p. if Vwalk.vwalk-bet G u p v then length p - 1 else ∞)

**lemma** *vwalk-bet-dist*:

Vwalk.vwalk-bet G u p v ⇒ *distance* G u v ≤ length p - 1

⟨proof⟩

**lemma** *reachable-dist*:

reachable G u v ⇒ *distance* G u v < ∞

⟨proof⟩

**lemma** *unreachable-dist*:

¬reachable G u v ⇒ *distance* G u v = ∞

⟨proof⟩

**lemma** *dist-reachable*:

*distance* G u v < ∞ ⇒ reachable G u v

⟨proof⟩

**lemma** *reachable-dist-2*:

**assumes** reachable G u v

**obtains** p **where** Vwalk.vwalk-bet G u p v *distance* G u v = length p - 1

⟨proof⟩

**lemma** *triangle-ineq-reachable*:

**assumes** reachable G u v reachable G v w

**shows** *distance* G u w ≤ *distance* G u v + *distance* G v w

⟨proof⟩

**lemma** *triangle-ineq*:

*distance* G u w ≤ *distance* G u v + *distance* G v w

⟨proof⟩

**lemma** *distance-split*:

[[*distance* G u v ≠ ∞; *distance* G u v = *distance* G u w + *distance* G w v]] ⇒

∃ w'. reachable G u w' ∧ *distance* G u w' = *distance* G u w ∧

reachable G w' v ∧ *distance* G w' v = *distance* G w' v

⟨proof⟩

**lemma** *dist-inf*: v ∉ dVs G ⇒ *distance* G u v = ∞

⟨proof⟩

**lemma** *dist-inf-2*: v ∉ dVs G ⇒ *distance* G v u = ∞

⟨proof⟩

**lemma** *dist-eq*:  $\llbracket \bigwedge p. Vwalk.vwalk\text{-bet } G' u p v \implies Vwalk.vwalk\text{-bet } G u (map f p) v \rrbracket \implies$

$$distance\ G\ u\ v \leq distance\ G'\ u\ v$$

*<proof>*

**lemma** *distance-subset*:  $G \subseteq G' \implies distance\ G'\ u\ v \leq distance\ G\ u\ v$

*<proof>*

**lemma** *distance-neighbourhood*:

$$\llbracket v \in neighbourhood\ G\ u \rrbracket \implies distance\ G\ u\ v \leq 1$$

*<proof>*

## 4.2 Shortest Paths

**definition** *shortest-path*:: $(v \times v) \text{ set} \Rightarrow v \Rightarrow v \text{ list} \Rightarrow v \Rightarrow \text{bool}$  **where**  
*shortest-path*  $G\ u\ p\ v = (distance\ G\ u\ v = length\ p - 1 \wedge vwalk\text{-bet } G\ u\ p\ v)$

**lemma** *shortest-path-props[elim]*:

$$shortest\text{-path } G\ u\ p\ v \implies (\llbracket distance\ G\ u\ v = length\ p - 1; vwalk\text{-bet } G\ u\ p\ v \rrbracket \implies P) \implies P$$

*<proof>*

**lemma** *shortest-path-intro*:

$$\llbracket distance\ G\ u\ v = length\ p - 1; vwalk\text{-bet } G\ u\ p\ v \rrbracket \implies shortest\text{-path } G\ u\ p\ v$$

*<proof>*

**lemma** *shortest-path-vwalk*:  $shortest\text{-path } G\ u\ p\ v \implies vwalk\text{-bet } G\ u\ p\ v$

*<proof>*

**lemma** *shortest-path-dist*:  $shortest\text{-path } G\ u\ p\ v \implies distance\ G\ u\ v = length\ p - 1$

*<proof>*

**lemma** *shortest-path-split-1*:

$$\llbracket shortest\text{-path } G\ u\ (p1 @ x \# p2)\ v \rrbracket \implies shortest\text{-path } G\ x\ (x \# p2)\ v$$

*<proof>*

**lemma** *shortest-path-split-2*:

$$\llbracket shortest\text{-path } G\ u\ (p1 @ x \# p2)\ v \rrbracket \implies shortest\text{-path } G\ u\ (p1 @ [x])\ x$$

*<proof>*

**lemma** *shortest-path-split-distance*:

$$\llbracket shortest\text{-path } G\ u\ (p1 @ x \# p2)\ v \rrbracket \implies distance\ G\ u\ x \leq distance\ G\ u\ v$$

*<proof>*

**lemma** *shortest-path-split-distance'*:

$$\llbracket x \in set\ p; shortest\text{-path } G\ u\ p\ v \rrbracket \implies distance\ G\ u\ x \leq distance\ G\ u\ v$$

*<proof>*

**lemma** *shortest-path-exists*:  
**assumes** *reachable*  $G\ u\ v$   
**obtains**  $p$  **where** *shortest-path*  $G\ u\ p\ v$   
 $\langle$ *proof* $\rangle$

**lemma** *shortest-path-exists-2*:  
**assumes**  $\text{distance } G\ u\ v < \infty$   
**obtains**  $p$  **where** *shortest-path*  $G\ u\ p\ v$   
 $\langle$ *proof* $\rangle$

**lemma** *not-distinct-props*:  
 $\neg \text{distinct } xs \implies (\bigwedge x1\ x2\ xs1\ xs2\ xs3. \llbracket xs = xs1\ @\ x1\ \# \ xs2\ @\ x2\ \# \ xs3; x1 = x2 \rrbracket \implies P) \implies P$   
 $\langle$ *proof* $\rangle$

**lemma** *shortest-path-distinct*:  
*shortest-path*  $G\ u\ p\ v \implies \text{distinct } p$   
 $\langle$ *proof* $\rangle$

**lemma** *diet-eq'*:  $\llbracket \bigwedge p. \text{shortest-path } G'\ u\ p\ v \implies \text{shortest-path } G\ u\ (\text{map } f\ p)\ v \rrbracket$   
 $\implies$   
 $\text{distance } G\ u\ v \leq \text{distance } G'\ u\ v$   
 $\langle$ *proof* $\rangle$

**lemma** *distance-0*:  
 $(u = v \wedge v \in dVs\ G) \longleftrightarrow \text{distance } G\ u\ v = 0$   
 $\langle$ *proof* $\rangle$

**lemma** *distance-neighbourhood'*:  
 $\llbracket v \in \text{neighbourhood } G\ u \rrbracket \implies \text{distance } G\ x\ v \leq \text{distance } G\ x\ u + 1$   
 $\langle$ *proof* $\rangle$

**lemma** *Suc-le-length-iff-2*:  
 $(\text{Suc } n \leq \text{length } xs) = (\exists x\ ys. xs = ys\ @\ [x] \wedge n \leq \text{length } ys)$   
 $\langle$ *proof* $\rangle$

**lemma** *distance-parent*:  
 $\llbracket \text{distance } G\ u\ v < \infty; u \neq v \rrbracket \implies$   
 $\exists w. \text{distance } G\ u\ w + 1 = \text{distance } G\ u\ v \wedge v \in \text{neighbourhood } G\ w$   
 $\langle$ *proof* $\rangle$

### 4.3 Distance from a set of vertices

**definition** *distance-set*:: $(v \times v)$  *set*  $\Rightarrow v$  *set*  $\Rightarrow v \Rightarrow \text{enat}$  **where**  
 $\text{distance-set } G\ U\ v = (\text{INF } u \in U. \text{distance } G\ u\ v)$

**lemma** *dist-set-inf*:  $v \notin dVs\ G \implies distance\text{-}set\ G\ U\ v = \infty$   
 ⟨proof⟩

**lemma** *dist-set-mem[intro]*:  $u \in U \implies distance\text{-}set\ G\ U\ v \leq distance\ G\ u\ v$   
 ⟨proof⟩

**lemma** *dist-not-inf''*:  $\llbracket distance\text{-}set\ G\ U\ v \neq \infty; u \in U; distance\ G\ u\ v = distance\text{-}set\ G\ U\ v \rrbracket$   
 $\implies \exists p. vwalk\text{-}bet\ G\ u\ (u\#p)\ v \wedge length\ p = distance\ G\ u\ v \wedge$   
 $set\ p \cap U = \{\}$   
 ⟨proof⟩

**lemma** *dist-not-inf'''*:  
 $\llbracket distance\text{-}set\ G\ U\ v \neq \infty; u \in U; distance\ G\ u\ v = distance\text{-}set\ G\ U\ v \rrbracket$   
 $\implies \exists p. shortest\text{-}path\ G\ u\ (u\#p)\ v \wedge set\ p \cap U = \{\}$   
 ⟨proof⟩

**lemma** *distance-set-union*:  
 $distance\text{-}set\ G\ (U \cup V)\ v \leq distance\text{-}set\ G\ U\ v$   
 ⟨proof⟩

**lemma** *lt-lt-infnty*:  $x < (y::enat) \implies x < \infty$   
 ⟨proof⟩

**lemma** *finite-dist-nempty*:  
 $distance\text{-}set\ G\ V\ v \neq \infty \implies V \neq \{\}$   
 ⟨proof⟩

**lemma** *distance-set-wit*:  
**assumes**  $v \in V$   
**obtains**  $v'$  **where**  $v' \in V\ distance\text{-}set\ G\ V\ x = distance\ G\ v'\ x$   
 ⟨proof⟩

**lemma** *distance-set-wit'*:  
**assumes**  $distance\text{-}set\ G\ V\ v \neq \infty$   
**obtains**  $v'$  **where**  $v' \in V\ distance\text{-}set\ G\ V\ x = distance\ G\ v'\ x$   
 ⟨proof⟩

**lemma** *dist-set-not-inf*:  $distance\text{-}set\ G\ U\ v \neq \infty \implies \exists u \in U. distance\ G\ u\ v = distance\text{-}set\ G\ U\ v$   
 ⟨proof⟩

**lemma** *dist-not-inf'*:  $distance\text{-}set\ G\ U\ v \neq \infty \implies \exists u \in U. distance\ G\ u\ v = distance\text{-}set\ G\ U\ v \wedge reachable\ G\ u\ v$   
 ⟨proof⟩

**lemma** *distance-on-vwalk*:

$$\llbracket \text{distance-set } G \ U \ v = \text{distance } G \ u \ v; u \in U; \text{shortest-path } G \ u \ p \ v; w \in \text{set } p \rrbracket \\ \implies \text{distance-set } G \ U \ w = \text{distance } G \ u \ w$$

$\langle \text{proof} \rangle$

**lemma** *diff-le-self-enat*:  $m - n \leq (m::\text{enat})$

$\langle \text{proof} \rangle$

**lemma** *shortest-path-dist-set-union*:

$$\llbracket \text{distance-set } G \ U \ v = \text{distance } G \ u \ v; u \in U; \text{shortest-path } G \ u \ (p1 \ @ \ x \ \# \ p2) \rrbracket \\ v;$$

$$x \in V; \bigwedge v'. v' \in V \implies \text{distance-set } G \ U \ v' = \text{distance } G \ u \ x$$

$$\implies \text{distance-set } G \ (U \cup V) \ v = \text{distance-set } G \ U \ v - \text{distance } G \ u \ x$$

$\langle \text{proof} \rangle$

**lemma** *Inf-enat-def1*:

**fixes**  $K::\text{enat set}$

**assumes**  $K \neq \{\}$

**shows**  $\text{Inf } K \in K$

$\langle \text{proof} \rangle$

**lemma** *INF-plus-enat*:

$$V \neq \{\} \implies (\text{INF } v \in V. (f::'a \Rightarrow \text{enat}) \ v) + (x::\text{enat}) = (\text{INF } v \in V. f \ v + x)$$

$\langle \text{proof} \rangle$

**lemma** *distance-set-neighbourhood*:

$$\llbracket v \in \text{neighbourhood } G \ u; Vs \neq \{\} \rrbracket \implies \text{distance-set } G \ Vs \ v \leq \text{distance-set } G \ Vs \ u + 1$$

$\langle \text{proof} \rangle$

**lemma** *distance-set-parent*:

$$\llbracket \text{distance-set } G \ Vs \ v < \infty; Vs \neq \{\}; v \notin Vs \rrbracket \implies$$

$$\exists w. \text{distance-set } G \ Vs \ w + 1 = \text{distance-set } G \ Vs \ v \wedge v \in \text{neighbourhood } G \ w$$

$\langle \text{proof} \rangle$

**lemma** *distance-set-parent'*:

$$\llbracket 0 < \text{distance-set } G \ Vs \ v; \text{distance-set } G \ Vs \ v < \infty; Vs \neq \{\} \rrbracket \implies$$

$$\exists w. \text{distance-set } G \ Vs \ w + 1 = \text{distance-set } G \ Vs \ v \wedge v \in \text{neighbourhood } G \ w$$

$\langle \text{proof} \rangle$

**lemma** *distance-set-0[simp]*:  $\llbracket v \in dVs \ G \rrbracket \implies \text{distance-set } G \ Vs \ v = 0 \iff v \in Vs$

$\langle \text{proof} \rangle$

**lemma** *dist-set-leq*:

$$\llbracket \bigwedge u. u \in Vs \implies \text{distance } G \ u \ v \leq \text{distance } G' \ u \ v \rrbracket \implies \text{distance-set } G \ Vs \ v \leq \text{distance-set } G' \ Vs \ v$$

$\langle \text{proof} \rangle$

**lemma** *dist-set-eq*:

$\llbracket \bigwedge u. u \in Vs \implies \text{distance } G \ u \ v = \text{distance } G' \ u \ v \rrbracket \implies \text{distance-set } G \ Vs \ v = \text{distance-set } G' \ Vs \ v$   
*<proof>*

**lemma** *distance-set-subset*:  $G \subseteq G' \implies \text{distance-set } G' \ Vs \ v \leq \text{distance-set } G \ Vs \ v$

*<proof>*

**lemma** *vwalk-bet-dist-set*:

$\llbracket \text{Vwalk.vwalk-bet } G \ u \ p \ v; u \in U \rrbracket \implies \text{distance-set } G \ U \ v \leq \text{length } p - 1$   
*<proof>*

**end**

**theory** *Set-Addons*

**imports** *HOL-Data-Structures.Set-Specs*

**begin**

**context** *Set*

**begin**

**bundle** *automation* = *set-empty[simp] set-isin[simp] set-insert[simp] set-delete[simp]*  
*invar-empty[simp] invar-insert[simp] invar-delete[simp]*

**end**

**end**

**theory** *Map-Addons*

**imports** *HOL-Data-Structures.Map-Specs*

**begin**

**context** *Map*

**begin**

**bundle** *automation* = *map-empty[simp] map-update[simp] map-delete[simp] in-*  
*var-empty[simp]*  
*invar-update[simp] invar-delete[simp]*

**end**

**end**

**theory** *Set2-Addons*

**imports** *HOL-Data-Structures.Set-Specs*

**begin**

**context** *Set2*

**begin**

**bundle** *automation* =  
*set-union[simp] set-inter[simp]*  
*set-diff[simp] invar-union[simp]*  
*invar-inter[simp] invar-diff[simp]*

**notation** *inter* (**infixl**  $\cap_G$  100)

```

notation diff (infixl  $-_G$  100)
notation union (infixl  $\cup_G$  100)
end

end
theory Pair-Graph-Specs
  imports Vwalk Map-Addons Set-Addons
begin

```

## 5 A Digraph Representation for Efficient Executable Functions

We develop a locale modelling an abstract data type (ADT) which abstractly models a graph as an adjacency map: i.e. every vertex is mapped to a *set* of adjacent vertices, and this latter set is again modelled using the ADT of sets provided in Isabelle/HOL's distribution.

We then show that this ADT can be implemented using existing implementations of the *set* ADT.

```

locale Set-Choose = set: Set
  where set = t-set for t-set ( $[-]_s$ ) +
fixes sel :: 's  $\Rightarrow$  'a

```

```

assumes choose [simp]:  $s \neq \text{empty} \implies \text{isin } s \text{ (sel } s)$ 

```

```

begin
context
  includes set.automation
begin

```

### 5.1 Abstraction lemmas

These are lemmas for automation. Their purpose is to remove any mention of the locale set ADT constructs and replace it with Isabelle's native sets.

```

lemma choose'[simp, intro, dest]:
   $s \neq \text{empty} \implies \text{invar } s \implies \text{sel } s \in \text{t-set } s$ 
   $\langle \text{proof} \rangle$ 

```

```

lemma choose''[intro]:
   $\text{invar } s \implies s \neq \text{empty} \implies \text{t-set } s \subseteq s' \implies \text{sel } s \in s'$ 
   $\langle \text{proof} \rangle$ 

```

```

lemma emptyD[dest]:
   $s = \text{empty} \implies \text{t-set } s = \{\}$ 
   $s \neq \text{empty} \implies \text{invar } s \implies \text{t-set } s \neq \{\}$ 
   $\text{empty} = s \implies \text{t-set } s = \{\}$ 
   $\text{empty} \neq s \implies \text{invar } s \implies \text{t-set } s \neq \{\}$ 

```

<proof>  
**end**  
**end**

**named-theorems** *Graph-Spec-Elims*  
**named-theorems** *Graph-Spec-Intros*  
**named-theorems** *Graph-Spec-Simps*

**locale** *Pair-Graph-Specs* =  
*adjmap*: *Map*  
**where** *update* = *update* **and** *invar* = *adjmap-inv* +

*vset*: *Set-Choose*  
**where** *empty* = *vset-empty* **and** *delete* = *vset-delete* **and** *invar* = *vset-inv*

**for** *update* :: 'v  $\Rightarrow$  'vset  $\Rightarrow$  'adjmap  $\Rightarrow$  'adjmap **and** *adjmap-inv* :: 'adjmap  $\Rightarrow$  bool  
**and**

*vset-empty* :: 'vset **and** *vset-delete* :: 'v  $\Rightarrow$  'vset  $\Rightarrow$  'vset **and**  
*vset-inv*

**begin**

**notation** *vset-empty* ( $\emptyset_V$ )  
**notation** *empty* ( $\emptyset_G$ )

**abbreviation** *isin'* (**infixl**  $\in_G$  50) **where** *isin'*  $G$   $v \equiv isin$   $v$   $G$   
**abbreviation** *not-isin'* (**infixl**  $\notin_G$  50) **where** *not-isin'*  $G$   $v \equiv \neg isin'$   $G$   $v$

**definition** *set-of-map* ( $m :: 'adjmap$ ) =  $\{(u,v). \text{case } (lookup\ m\ u) \text{ of } Some\ vs \Rightarrow v \in_G\ vs\}$

**definition** *graph-inv*  $G = (adjmap-inv\ G \wedge (\forall v\ vset. lookup\ G\ v = Some\ vset \longrightarrow vset-inv\ vset))$

**definition** *finite-graph*  $G = (finite\ \{v. (lookup\ G\ v) \neq None\})$

**definition** *finite-vsets* =  $(\forall vset. finite\ (t-set\ vset))$

**definition** *neighb*::'adjmap  $\Rightarrow$  'v  $\Rightarrow$  'vset **where**  
*(neighb*  $G$   $v) = (\text{case } (lookup\ G\ v) \text{ of } Some\ vset \Rightarrow vset \mid - \Rightarrow vset-empty)$

**lemmas** [*code*] = *neighb-def*

**notation** *neighb* ( $\mathcal{N}_G$  - - 100)

**definition** *digraph-abs* ( $[-]_G$ ) **where** *digraph-abs*  $G = \{(u,v). v \in_G (\mathcal{N}_G\ G\ u)\}$

**definition** *add-edge*  $G\ u\ v =$   
 (  
   *case* (*lookup*  $G\ u$ ) *of* *Some*  $vset \Rightarrow$   
   *let*  
      $vset = the\ (lookup\ G\ u);$   
      $vset' = insert\ v\ vset;$   
      $digraph' = update\ u\ vset'\ G$   
   *in*  
      $digraph'$   
 |  $- \Rightarrow$   
   *let*  
      $vset' = insert\ v\ vset-empty;$   
      $digraph' = update\ u\ vset'\ G$   
   *in*  
      $digraph'$   
 )

**definition** *delete-edge*  $G\ u\ v =$   
 (  
   *case* (*lookup*  $G\ u$ ) *of* *Some*  $vset \Rightarrow$   
   *let*  
      $vset = the\ (lookup\ G\ u);$   
      $vset' = vset-delete\ v\ vset;$   
      $digraph' = update\ u\ vset'\ G$   
   *in*  
      $digraph'$   
 |  $- \Rightarrow G$   
 )

**context** — *Locale properties*

**includes** *vset.set.automation* **and** *adjmap.automation*

**fixes**  $G::'adjmap$

**begin**

**lemma** *graph-invE[elim]*:

$graph-inv\ G \Longrightarrow (\llbracket adjmap-inv\ G; (\bigwedge v\ vset.\ lookup\ G\ v = Some\ vset \Longrightarrow vset-inv\ vset) \rrbracket \Longrightarrow P) \Longrightarrow P$   
*<proof>*

**lemma** *graph-invI[intro]*:

$\llbracket adjmap-inv\ G; (\bigwedge v\ vset.\ lookup\ G\ v = Some\ vset \Longrightarrow vset-inv\ vset) \rrbracket \Longrightarrow graph-inv\ G$   
*<proof>*

**lemma** *finite-graphE[elim]*:

$finite-graph\ G \Longrightarrow (finite\ \{v.\ (lookup\ G\ v) \neq None\} \Longrightarrow P) \Longrightarrow P$   
*<proof>*

**lemma** *finite-graphI*[*intro*]:  
 $finite \{v. (lookup \ G \ v) \neq None\} \implies finite-graph \ G$   
 $\langle proof \rangle$

**lemma** *finite-vssetsE*[*elim*]:  
 $finite-vssets \implies ((\bigwedge N. finite \ (t-set \ N)) \implies P) \implies P$   
 $\langle proof \rangle$

**lemma** *finite-vssetsI*[*intro*]:  
 $(\bigwedge N. finite \ (t-set \ N)) \implies finite-vssets$   
 $\langle proof \rangle$

**lemma** *neighbourhood-invars'*[*simp,dest*]:  
 $graph-inv \ G \implies vset-inv \ (\mathcal{N}_G \ G \ v)$   
 $\langle proof \rangle$

**lemma** *finite-graph*[*intro!*]:  
**assumes** *graph-inv*  $G$  *finite-graph*  $G$  *finite-vssets*  
**shows** *finite* (*digraph-abs*  $G$ )  
 $\langle proof \rangle$

**corollary** *finite-vertices*[*intro!*]:  
**assumes** *graph-inv*  $G$  *finite-graph*  $G$  *finite-vssets*  
**shows** *finite* (*dVs* (*digraph-abs*  $G$ ))  
 $\langle proof \rangle$

## 5.2 Abstraction lemmas

These are lemmas for automation. Their purpose is to remove any mention of the neighbourhood concept implemented using the locale constructs and replace it with abstract terms on pair graphs.

**lemma** *are-connected-abs*[*simp*]:  
 $graph-inv \ G \implies v \in t-set \ (\mathcal{N}_G \ G \ u) \longleftrightarrow (u,v) \in digraph-abs \ G$   
 $\langle proof \rangle$

**lemma** *are-connected-absD*[*dest*]:  
 $\llbracket v \in t-set \ (\mathcal{N}_G \ G \ u); graph-inv \ G \rrbracket \implies (u,v) \in digraph-abs \ G$   
 $\langle proof \rangle$

**lemma** *are-connected-absI*[*intro*]:  
 $\llbracket (u,v) \in digraph-abs \ G; graph-inv \ G \rrbracket \implies v \in t-set \ (\mathcal{N}_G \ G \ u)$   
 $\langle proof \rangle$

**lemma** *neighbourhood-absD*[*dest!*]:  
 $\llbracket t-set \ (\mathcal{N}_G \ G \ x) \neq \{\}; graph-inv \ G \rrbracket \implies x \in dVs \ (digraph-abs \ G)$   
 $\langle proof \rangle$

```

lemma neighbourhood-abs[simp]:
  graph-inv G  $\implies$  t-set ( $\mathcal{N}_G$  G u) = neighbourhood (digraph-abs G) u
  <proof>

lemma adjmap-inv-insert[intro]: graph-inv G  $\implies$  graph-inv (add-edge G u v)
  <proof>

lemma digraph-abs-insert[simp]: graph-inv G  $\implies$  digraph-abs (add-edge G u v) =
  Set.insert (u,v) (digraph-abs G)
  <proof>

lemma adjmap-inv-delete[intro]: graph-inv G  $\implies$  graph-inv (delete-edge G u v)
  <proof>

lemma digraph-abs-delete[simp]: graph-inv G  $\implies$  digraph-abs (delete-edge G u v)
  = (digraph-abs G) - {(u,v)}
  <proof>

end — Properties context

end

  Pair-Graph-Specs

end
theory DFS
  imports Pair-Graph-Specs Set2-Addons Set-Addons
begin

```

## 6 Depth-Frist Search

### 6.1 The program state

```
datatype return = Reachable | NotReachable
```

```
record ('ver, 'vset) DFS-state = stack:: 'ver list seen:: 'vset return:: return
```

### 6.2 Setup for automation

```

named-theorems call-cond-elim
named-theorems call-cond-intros
named-theorems ret-holds-intros
named-theorems invar-props-intros
named-theorems invar-props-elim
named-theorems invar-holds-intros
named-theorems state-rel-intros
named-theorems state-rel-holds-intros

```

### 6.3 A locale for fixing data structures and their implementations

locale *DFS* =

*Graph*: *Pair-Graph-Specs*  
**where** *lookup* = *lookup* +  
*set-ops*: *Set2 vset-empty vset-delete - t-set vset-inv insert*

**for** *lookup* :: '*adjmap* ⇒ '*v* ⇒ '*vset option* +

**fixes** *G*::'*adjmap* **and** *s*::'*v* **and** *t*::'*v*

**begin**

**definition** *DFS-axioms* = ( *Graph.graph-inv G* ∧ *Graph.finite-graph G* ∧ *Graph.finite-vsets*  
 ∧ *s* ∈ *dVs (Graph.digraph-abs G)*)

**abbreviation** *neighb' v* == *Graph.neighb G v*

**notation** *neighb' (N<sub>G</sub> - 100)*

### 6.4 Defining the Algorithm

**function** (*domintros*) *DFS*::('v, 'vset) *DFS-state* ⇒ ('v, 'vset) *DFS-state* **where**

*DFS dfs-state* =  
 (case (*stack dfs-state*) of (*v # stack-tl*) ⇒  
 (if *v = t* then  
 (*dfs-state* (|*return* := *Reachable*|))  
 else ((if (*N<sub>G</sub> v -<sub>G</sub> (seen dfs-state)*) ≠ ∅<sub>V</sub> then  
 let *u* = (*sel ((N<sub>G</sub> v) -<sub>G</sub> (seen dfs-state))*);  
   *stack'* = *u # (stack dfs-state)*;  
   *seen'* = *insert u (seen dfs-state)*  
 in  
 (*DFS (dfs-state* (|*stack* := *stack'*,  
   *seen* := *seen'* |)))  
 else  
 let *stack'* = *stack-tl* in  
*DFS (dfs-state* (|*stack* := *stack'*|)))  
 )  
 )  
 | - ⇒ (*dfs-state* (|*return* := *NotReachable*|))  
 )  
 {*proof*}

### 6.5 Setup for Reasoning About the Algorithm

**definition** *initial-state*::('v, 'vset) *DFS-state* **where**

*initial-state* = (|*stack* = [*s*], *seen* = *insert s ∅<sub>V</sub>*, *return* = *NotReachable*|)

**definition** *DFS-call-1-conds* *dfs-state* =  
 (case stack *dfs-state* of (*v* # *stack-tl*) ⇒  
 (if *v* = *t* then  
   False  
 else ((if (( $\mathcal{N}_G$  *v*)  $-_G$  (*seen dfs-state*)) ≠ ( $\emptyset_V$ ) then  
   True  
 else False)  
 )  
 )  
 | - ⇒ False  
 )

**lemma** *DFS-call-1-conds*[*call-cond-elim*]:  
*DFS-call-1-conds dfs-state* ⇒  
 [[∃ *v stack-tl. stack dfs-state* = *v* # *stack-tl*;  
*hd (stack dfs-state)* ≠ *t*;  
 ( $\mathcal{N}_G$  (*hd (stack dfs-state)*))  $-_G$  (*seen dfs-state*) ≠ ( $\emptyset_V$ )] ⇒ *P*] ⇒  
*P*  
 ⟨*proof*⟩

**definition** *DFS-upd1 dfs-state* = (  
 let  
   *N* = ( $\mathcal{N}_G$  (*hd (stack dfs-state)*));  
   *u* = (*sel* (( $\mathcal{N}_G$  (*seen dfs-state*))));  
   *stack'* = *u* # (*stack dfs-state*);  
   *seen'* = *insert u (seen dfs-state)*  
 in  
*dfs-state* (*stack := stack', seen := seen'*)

**definition** *DFS-call-2-conds*::(*'v, 'vset*) *DFS-state* ⇒ *bool* **where**  
*DFS-call-2-conds dfs-state* =  
 (case stack *dfs-state* of (*v* # *stack-tl*) ⇒  
 (if *v* = *t* then  
   False  
 else (  
   (if (( $\mathcal{N}_G$  *v*)  $-_G$  (*seen dfs-state*)) ≠ ( $\emptyset_V$ ) then  
     False  
   else True)  
 )  
 )  
 | - ⇒ False  
 )

**lemma** *DFS-call-2-condsE*[*call-cond-elim*]:  
*DFS-call-2-conds dfs-state* ⇒  
 [[∃ *v stack-tl. stack dfs-state* = *v* # *stack-tl*;  
*hd (stack dfs-state)* ≠ *t*;

$(\mathcal{N}_G (\text{hd } (\text{stack } \text{dfs-state}))) -_G (\text{seen } \text{dfs-state}) = (\emptyset_V) \implies P \implies$   
 $P$   
 <proof>

**definition** *DFS-upd2* *dfs-state* =  
 $((\text{dfs-state } (\text{stack} := \text{tl } (\text{stack } \text{dfs-state}))))$

**definition** *DFS-ret-1-conds* *dfs-state* =  
 $(\text{case stack } \text{dfs-state} \text{ of } (v \# \text{stack-tl}) \Rightarrow$   
 $(\text{if } v = t \text{ then}$   
 $\quad \text{False}$   
 $\text{else } ($   
 $\quad (\text{if } ((\mathcal{N}_G v) -_G (\text{seen } \text{dfs-state})) \neq \emptyset_V \text{ then}$   
 $\quad \quad \text{False}$   
 $\quad \quad \text{else False})$   
 $\quad )$   
 $)$   
 $| - \Rightarrow \text{True}$   
 $)$

**lemma** *DFS-ret-1-conds*[*call-cond-elim*]:  
 $\text{DFS-ret-1-conds } \text{dfs-state} \implies$   
 $\llbracket \text{stack } \text{dfs-state} = [] \rrbracket \implies P \implies$   
 $P$   
 <proof>

**lemma** *DFS-call-4-condsI*[*call-cond-intros*]:  
 $\llbracket \text{stack } \text{dfs-state} = [] \rrbracket \implies \text{DFS-ret-1-conds } \text{dfs-state}$   
 <proof>

**definition** *DFS-ret1* *dfs-state* =  $(\text{dfs-state } (\text{return} := \text{NotReachable}))$

**definition** *DFS-ret-2-conds* *dfs-state* =  
 $(\text{case stack } \text{dfs-state} \text{ of } (v \# \text{stack-tl}) \Rightarrow$   
 $(\text{if } v = t \text{ then}$   
 $\quad \text{True}$   
 $\text{else } ($   
 $\quad (\text{if } (\mathcal{N}_G v -_G (\text{seen } \text{dfs-state})) \neq \emptyset_V \text{ then}$   
 $\quad \quad \text{False}$   
 $\quad \quad \text{else False})$   
 $\quad )$   
 $)$   
 $| - \Rightarrow \text{False}$   
 $)$

**lemma** *DFS-ret-2-conds*[*call-cond-elim*]:  
 $\text{DFS-ret-2-conds } \text{dfs-state} \implies$

$\llbracket \bigwedge v \text{ stack-tl. } \llbracket \text{stack dfs-state} = v \# \text{stack-tl};$   
 $(\text{hd } (\text{stack dfs-state})) = t \rrbracket \implies P \rrbracket \implies$   
 $P$   
 $\langle \text{proof} \rangle$

**lemma** *DFS-ret-2-condsI*[*call-cond-intros*]:

$\bigwedge v \text{ stack-tl. } \llbracket \text{stack dfs-state} = v \# \text{stack-tl}; (\text{hd } (\text{stack dfs-state})) = t \rrbracket \implies$   
 $\text{DFS-ret-2-conds dfs-state}$   
 $\langle \text{proof} \rangle$

**definition** *DFS-ret2*  $\text{dfs-state} = (\text{dfs-state } (\text{return} := \text{Reachable}))$

**lemma** *DFS-cases*:

**assumes** *DFS-call-1-conds*  $\text{dfs-state} \implies P$   
 $\text{DFS-call-2-conds } \text{dfs-state} \implies P$   
 $\text{DFS-ret-1-conds } \text{dfs-state} \implies P$   
 $\text{DFS-ret-2-conds } \text{dfs-state} \implies P$   
**shows**  $P$   
 $\langle \text{proof} \rangle$

**lemma** *DFS-simps*:

**assumes** *DFS-dom*  $\text{dfs-state}$   
**shows** *DFS-call-1-conds*  $\text{dfs-state} \implies \text{DFS } \text{dfs-state} = \text{DFS } (\text{DFS-upd1 } \text{dfs-state})$   
 $\text{DFS-call-2-conds } \text{dfs-state} \implies \text{DFS } \text{dfs-state} = \text{DFS } (\text{DFS-upd2 } \text{dfs-state})$   
 $\text{DFS-ret-1-conds } \text{dfs-state} \implies \text{DFS } \text{dfs-state} = \text{DFS-ret1 } \text{dfs-state}$   
 $\text{DFS-ret-2-conds } \text{dfs-state} \implies \text{DFS } \text{dfs-state} = \text{DFS-ret2 } \text{dfs-state}$   
 $\langle \text{proof} \rangle$

**lemma** *DFS-induct*:

**assumes** *DFS-dom*  $\text{dfs-state}$   
**assumes**  $\bigwedge \text{dfs-state. } \llbracket \text{DFS-dom } \text{dfs-state};$   
 $\text{DFS-call-1-conds } \text{dfs-state} \implies P (\text{DFS-upd1 } \text{dfs-state});$   
 $\text{DFS-call-2-conds } \text{dfs-state} \implies P (\text{DFS-upd2 } \text{dfs-state}) \rrbracket \implies P$   
 $\text{dfs-state}$   
**shows**  $P \text{ dfs-state}$   
 $\langle \text{proof} \rangle$

**lemma** *DFS-domintros*:

**assumes** *DFS-call-1-conds*  $\text{dfs-state} \implies \text{DFS-dom } (\text{DFS-upd1 } \text{dfs-state})$   
**assumes** *DFS-call-2-conds*  $\text{dfs-state} \implies \text{DFS-dom } (\text{DFS-upd2 } \text{dfs-state})$   
**shows** *DFS-dom*  $\text{dfs-state}$   
 $\langle \text{proof} \rangle$

## 6.6 Loop Invariants

**definition** *invar-well-formed*:: $(v, vset) \text{ DFS-state} \Rightarrow \text{bool}$  **where**  
 $\text{invar-well-formed } \text{dfs-state} = \text{vset-inv } (\text{seen } \text{dfs-state})$

**definition** *invar-stack-walk*:: $(v, vset) \text{ DFS-state} \Rightarrow \text{bool}$  **where**

$invar\text{-}stack\text{-}walk\ dfs\text{-}state = (Vwalk.vwalk (Graph.digraph\text{-}abs\ G) (rev (stack\ dfs\text{-}state)))$

**definition**  $invar\text{-}seen\text{-}stack::('v, 'vset)\ DFS\text{-}state \Rightarrow bool$  **where**

$invar\text{-}seen\text{-}stack\ dfs\text{-}state \longleftrightarrow$   
 $distinct (stack\ dfs\text{-}state)$   
 $\wedge set (stack\ dfs\text{-}state) \subseteq t\text{-}set (seen\ dfs\text{-}state)$   
 $\wedge t\text{-}set (seen\ dfs\text{-}state) \subseteq dVs (Graph.digraph\text{-}abs\ G)$

**definition**  $invar\text{-}s\text{-}in\text{-}stack::('v, 'vset)\ DFS\text{-}state \Rightarrow bool$  **where**

$invar\text{-}s\text{-}in\text{-}stack\ dfs\text{-}state \longleftrightarrow$   
 $(stack (dfs\text{-}state) \neq [] \longrightarrow last (stack\ dfs\text{-}state) = s)$

**definition**  $invar\text{-}visited\text{-}through\text{-}seen::('v, 'vset)\ DFS\text{-}state \Rightarrow bool$  **where**

$invar\text{-}visited\text{-}through\text{-}seen\ dfs\text{-}state =$   
 $(\forall v \in t\text{-}set (seen\ dfs\text{-}state).$   
 $(\forall p. Vwalk.vwalk\text{-}bet (Graph.digraph\text{-}abs\ G) v\ p\ t \wedge distinct\ p \longrightarrow (set\ p \cap$   
 $set (stack\ dfs\text{-}state) \neq \{\})))$

**definition**  $call\text{-}1\text{-}measure::('v, 'vset)\ DFS\text{-}state \Rightarrow nat$  **where**

$call\text{-}1\text{-}measure\ dfs\text{-}state = card (dVs (Graph.digraph\text{-}abs\ G) - t\text{-}set (seen\ dfs\text{-}state))$

**definition**  $call\text{-}2\text{-}measure::('v, 'vset)\ DFS\text{-}state \Rightarrow nat$  **where**

$call\text{-}2\text{-}measure\ dfs\text{-}state = card (set (stack\ dfs\text{-}state))$

**definition**  $DFS\text{-}term\text{-}rel::('v, 'vset)\ DFS\text{-}state \times ('v, 'vset)\ DFS\text{-}state) set$  **where**

$DFS\text{-}term\text{-}rel = (call\text{-}1\text{-}measure) <*\text{mlex}*> (call\text{-}2\text{-}measure) <*\text{mlex}*> \{\}$

**end**

**locale**  $DFS\text{-}thms = DFS +$

**assumes**  $DFS\text{-}axioms: DFS\text{-}axioms$

**begin**

**context**

**includes**  $set\text{-}ops.\text{automation}$  **and**  $Graph.adjmap.\text{automation}$  **and**  $Graph.vset.set.\text{automation}$

**begin**

**lemma**  $graph\text{-}inv[simp, intro]:$

$Graph.graph\text{-}inv\ G$   
 $Graph.finite\text{-}graph\ G$   
 $Graph.finite\text{-}vsets$

$\langle proof \rangle$

**lemma**  $s\text{-}in\text{-}G[simp, intro]: s \in dVs (Graph.digraph\text{-}abs\ G)$

$\langle proof \rangle$

**lemma** *finite-neighbourhoods*[*simp*]:

*finite (t-set N)*  
*<proof>*

**lemmas** *simps*[*simp*] = *Graph.neighbourhood-abs*[*OF graph-inv(1)*] *Graph.are-connected-abs*[*OF graph-inv(1)*]

**lemma** *invar-well-formed-props*[*invar-props-elim*s]:

*invar-well-formed dfs-state*  $\implies$   
*([[vset-inv (seen dfs-state)]]  $\implies$  P)  $\implies$*   
*P*  
*<proof>*

**lemma** *invar-well-formed-intro*[*invar-props-intros*]: *[[vset-inv (seen dfs-state)]]*  
 $\implies$  *invar-well-formed dfs-state*

*<proof>*

**lemma** *invar-well-formed-holds-1*[*invar-holds-intros*]:

*[[DFS-call-1-conds dfs-state; invar-well-formed dfs-state]]  $\implies$*   
*invar-well-formed (DFS-upd1 dfs-state)*  
*<proof>*

**lemma** *invar-well-formed-holds-2*[*invar-holds-intros*]: *[[DFS-call-2-conds dfs-state;*  
*invar-well-formed dfs-state]]  $\implies$  invar-well-formed (DFS-upd2 dfs-state)*

*<proof>*

**lemma** *invar-well-formed-holds-4*[*invar-holds-intros*]: *[[DFS-ret-1-conds dfs-state;*  
*invar-well-formed dfs-state]]  $\implies$  invar-well-formed (DFS-ret1 dfs-state)*

*<proof>*

**lemma** *invar-well-formed-holds-5*[*invar-holds-intros*]: *[[DFS-ret-2-conds dfs-state;*  
*invar-well-formed dfs-state]]  $\implies$  invar-well-formed (DFS-ret2 dfs-state)*

*<proof>*

**lemma** *invar-well-formed-holds*[*invar-holds-intros*]:

**assumes** *DFS-dom dfs-state invar-well-formed dfs-state*

**shows** *invar-well-formed (DFS dfs-state)*

*<proof>*

**lemma** *invar-stack-walk-props*[*invar-props-elim*s]:

*invar-stack-walk dfs-state  $\implies$*   
*((Vwalk.vwalk (Graph.digraph-abs G) (rev (stack dfs-state))))  $\implies$  P  $\implies$  P*  
*<proof>*

**lemma** *invar-stack-walk-intro*[*invar-props-intros*]: *Vwalk.vwalk (Graph.digraph-abs*  
*G) (rev (stack dfs-state))  $\implies$  invar-stack-walk dfs-state*

*<proof>*

**lemma** *invar-stack-walk-holds-1*[*invar-holds-intros*]:  
**assumes** *DFS-call-1-conds dfs-state invar-well-formed dfs-state invar-stack-walk dfs-state*  
**shows** *invar-stack-walk (DFS-upd1 dfs-state)*  
 ⟨*proof*⟩

**lemma** *invar-stack-walk-holds-2*[*invar-holds-intros*]:  $\llbracket \text{DFS-call-2-conds } \text{dfs-state}; \text{invar-stack-walk } \text{dfs-state} \rrbracket \implies \text{invar-stack-walk } (\text{DFS-upd2 } \text{dfs-state})$   
 ⟨*proof*⟩

**lemma** *invar-stack-walk-holds-4*[*invar-holds-intros*]:  $\llbracket \text{DFS-ret-1-conds } \text{dfs-state}; \text{invar-stack-walk } \text{dfs-state} \rrbracket \implies \text{invar-stack-walk } (\text{DFS-ret1 } \text{dfs-state})$   
 ⟨*proof*⟩

**lemma** *invar-2-holds-5*[*invar-holds-intros*]:  $\llbracket \text{DFS-ret-2-conds } \text{dfs-state}; \text{invar-stack-walk } \text{dfs-state} \rrbracket \implies \text{invar-stack-walk } (\text{DFS-ret2 } \text{dfs-state})$   
 ⟨*proof*⟩

**lemma** *invar-2-holds*[*invar-holds-intros*]:  
**assumes** *DFS-dom dfs-state invar-well-formed dfs-state invar-stack-walk dfs-state*  
**shows** *invar-stack-walk (DFS dfs-state)*  
 ⟨*proof*⟩

**lemma** *invar-seen-stack-props*[*invar-props-elim*s]:  
*invar-seen-stack dfs-state*  $\implies$   
 ( $\llbracket \text{distinct } (\text{stack } \text{dfs-state}); \text{set } (\text{stack } \text{dfs-state}) \subseteq \text{t-set } (\text{seen } \text{dfs-state}); \text{t-set } (\text{seen } \text{dfs-state}) \subseteq \text{dVs } (\text{Graph.digraph-abs } G) \rrbracket \implies P$ )  $\implies P$   
 ⟨*proof*⟩

**lemma** *invar-seen-stack-intro*[*invar-props-intros*]:  
 $\llbracket \text{distinct } (\text{stack } \text{dfs-state}); \text{set } (\text{stack } \text{dfs-state}) \subseteq \text{t-set } (\text{seen } \text{dfs-state}); \text{t-set } (\text{seen } \text{dfs-state}) \subseteq \text{dVs } (\text{Graph.digraph-abs } G) \rrbracket \implies \text{invar-seen-stack } \text{dfs-state}$   
 ⟨*proof*⟩

**lemma** *invar-seen-stack-holds-1*[*invar-holds-intros*]:  
 $\llbracket \text{DFS-call-1-conds } \text{dfs-state}; \text{invar-well-formed } \text{dfs-state}; \text{invar-seen-stack } \text{dfs-state} \rrbracket$   
 $\implies \text{invar-seen-stack } (\text{DFS-upd1 } \text{dfs-state})$   
 ⟨*proof*⟩

**lemma** *invar-seen-stack-holds-2*[*invar-holds-intros*]:  
 $\llbracket \text{DFS-call-2-conds } \text{dfs-state}; \text{invar-well-formed } \text{dfs-state}; \text{invar-seen-stack } \text{dfs-state} \rrbracket$   
 $\implies \text{invar-seen-stack } (\text{DFS-upd2 } \text{dfs-state})$   
 ⟨*proof*⟩

**lemma** *invar-seen-stack-holds-4*[*invar-holds-intros*]:  
 $\llbracket \text{DFS-ret-1-conds } \text{dfs-state}; \text{invar-seen-stack } \text{dfs-state} \rrbracket \implies$

*invar-seen-stack* (*DFS-ret1* *dfs-state*)  
⟨*proof*⟩

**lemma** *invar-seen-stack-holds-5*[*invar-holds-intros*]:  $\llbracket \text{DFS-ret-2-conds } \text{dfs-state}; \text{invar-seen-stack } \text{dfs-state} \rrbracket \implies \text{invar-seen-stack } (\text{DFS-ret2 } \text{dfs-state})$   
⟨*proof*⟩

**lemma** *invar-seen-stack-holds*[*invar-holds-intros*]:  
**assumes** *DFS-dom* *dfs-state invar-well-formed* *dfs-state invar-seen-stack* *dfs-state*  
**shows** *invar-seen-stack* (*DFS* *dfs-state*)  
⟨*proof*⟩

**lemma** *invar-s-in-stack-props*[*invar-props-elim*]:  
*invar-s-in-stack* *dfs-state*  $\implies$   
 $(\llbracket (\text{stack } (\text{dfs-state}) \neq [] \implies \text{last } (\text{stack } \text{dfs-state}) = s) \rrbracket \implies P) \implies P$   
⟨*proof*⟩

**lemma** *invar-s-in-stack-intro*[*invar-props-intros*]:  
 $\llbracket (\text{stack } (\text{dfs-state}) \neq [] \implies \text{last } (\text{stack } \text{dfs-state}) = s) \rrbracket \implies \text{invar-s-in-stack } \text{dfs-state}$   
⟨*proof*⟩

**lemma** *invar-s-in-stack-holds-1*[*invar-holds-intros*]:  
 $\llbracket \text{DFS-call-1-conds } \text{dfs-state}; \text{invar-well-formed } \text{dfs-state}; \text{invar-s-in-stack } \text{dfs-state} \rrbracket$   
 $\implies \text{invar-s-in-stack } (\text{DFS-upd1 } \text{dfs-state})$   
⟨*proof*⟩

**lemma** *invar-s-in-stack-holds-2*[*invar-holds-intros*]:  
 $\llbracket \text{DFS-call-2-conds } \text{dfs-state}; \text{invar-well-formed } \text{dfs-state}; \text{invar-s-in-stack } \text{dfs-state} \rrbracket$   
 $\implies$   
*invar-s-in-stack* (*DFS-upd2* *dfs-state*)  
⟨*proof*⟩

**lemma** *invar-s-in-stack-holds-4*[*invar-holds-intros*]:  
 $\llbracket \text{DFS-ret-1-conds } \text{dfs-state}; \text{invar-s-in-stack } \text{dfs-state} \rrbracket \implies$   
*invar-s-in-stack* (*DFS-ret1* *dfs-state*)  
⟨*proof*⟩

**lemma** *invar-s-in-stack-holds-5*[*invar-holds-intros*]:  $\llbracket \text{DFS-ret-2-conds } \text{dfs-state}; \text{invar-s-in-stack } \text{dfs-state} \rrbracket \implies \text{invar-s-in-stack } (\text{DFS-ret2 } \text{dfs-state})$   
⟨*proof*⟩

**lemma** *invar-s-in-stack-holds*[*invar-holds-intros*]:  
**assumes** *DFS-dom* *dfs-state invar-well-formed* *dfs-state invar-s-in-stack* *dfs-state*  
**shows** *invar-s-in-stack* (*DFS* *dfs-state*)  
⟨*proof*⟩

**lemma** *invar-visited-through-seen-props*[*elim!*]:  

$$\begin{aligned} & \text{invar-visited-through-seen } \text{dfs-state} \implies \\ & \left( \left[ \bigwedge v p. \left[ v \in t\text{-set (seen dfs-state)} \right]; \right. \right. \\ & \quad \left. \left( V\text{walk.vwalk-bet (Graph.digraph-abs } G) v p t \right); \text{distinct } p \right] \implies \\ & \quad \left. \text{set } p \cap \text{set (stack dfs-state)} \neq \{\} \right] \implies P \implies P \end{aligned}$$
  
 ⟨*proof*⟩

**lemma** *invar-visited-through-seen-intro*[*invar-props-intros*]:  

$$\begin{aligned} & \left[ \bigwedge v p. \left[ v \in t\text{-set (seen dfs-state)} \right]; \right. \\ & \quad \left. \left( V\text{walk.vwalk-bet (Graph.digraph-abs } G) v p t \right); \text{distinct } p \right] \implies \\ & \quad \text{set } p \cap \text{set (stack dfs-state)} \neq \{\} \implies \text{invar-visited-through-seen } \text{dfs-state} \end{aligned}$$
  
 ⟨*proof*⟩

## 6.7 Proofs that the Invariants Hold

**lemma** *invar-visited-through-seen-holds-1*[*invar-holds-intros*]:  

$$\begin{aligned} & \left[ \text{DFS-call-1-conds } \text{dfs-state}; \text{invar-well-formed } \text{dfs-state}; \text{invar-seen-stack } \text{dfs-state}; \right. \\ & \quad \left. \text{invar-visited-through-seen } \text{dfs-state} \right] \\ & \implies \text{invar-visited-through-seen (DFS-upd1 } \text{dfs-state)} \end{aligned}$$
  
 ⟨*proof*⟩

**lemma** *invar-visited-through-seen-holds-2*[*invar-holds-intros*]:  

$$\begin{aligned} & \left[ \text{DFS-call-2-conds } \text{dfs-state}; \text{invar-well-formed } \text{dfs-state}; \text{invar-seen-stack } \text{dfs-state}; \right. \\ & \quad \left. \text{invar-visited-through-seen } \text{dfs-state} \right] \implies \text{invar-visited-through-seen (DFS-upd2} \\ & \text{dfs-state)} \end{aligned}$$
  
 ⟨*proof*⟩

**lemma** *invar-visited-through-seen-holds-4*[*invar-holds-intros*]:  $\left[ \text{DFS-ret-1-conds } \text{dfs-state}; \right.$   

$$\left. \text{invar-visited-through-seen } \text{dfs-state} \right] \implies \text{invar-visited-through-seen (DFS-ret1 } \text{dfs-state)}$$
  
 ⟨*proof*⟩

**lemma** *invar-visited-through-seen-holds-5*[*invar-holds-intros*]:  $\left[ \text{DFS-ret-2-conds } \text{dfs-state}; \right.$   

$$\left. \text{invar-visited-through-seen } \text{dfs-state} \right] \implies \text{invar-visited-through-seen (DFS-ret2 } \text{dfs-state)}$$
  
 ⟨*proof*⟩

**lemma** *invar-visited-through-seen-holds*[*invar-holds-intros*]:  
**assumes** *DFS-dom* *dfs-state* *invar-well-formed* *dfs-state* *invar-seen-stack* *dfs-state*  
*invar-visited-through-seen* *dfs-state*  
**shows** *invar-visited-through-seen* (*DFS* *dfs-state*)  
 ⟨*proof*⟩

**definition** *state-rel-1* *dfs-state-1* *dfs-state-2*  

$$= ( t\text{-set (seen } \text{dfs-state-1}) \subseteq t\text{-set (seen } \text{dfs-state-2}))$$

**lemma** *state-rel-1-props*[*elim!*]: *state-rel-1* *dfs-state-1* *dfs-state-2*  $\implies$

$(t\text{-set (seen dfs-state-1)} \subseteq t\text{-set (seen dfs-state-2)}) \implies$

$P) \implies P$   
 $\langle \text{proof} \rangle$

**lemma** *state-rel-1-intro*[*state-rel-intros*]:  
 $\llbracket t\text{-set (seen dfs-state-1)} \subseteq t\text{-set (seen dfs-state-2)} \rrbracket \implies \text{state-rel-1 dfs-state-1}$   
 $\text{dfs-state-2}$   
 $\langle \text{proof} \rangle$

**lemma** *state-rel-1-trans*:  
 $\llbracket \text{state-rel-1 dfs-state-1 dfs-state-2}; \text{state-rel-1 dfs-state-2 dfs-state-3} \rrbracket \implies$   
 $\text{state-rel-1 dfs-state-1 dfs-state-3}$   
 $\langle \text{proof} \rangle$

**lemma** *state-rel-1-holds-1*[*state-rel-holds-intros*]:  
 $\llbracket \text{DFS-call-1-conds dfs-state}; \text{invar-well-formed dfs-state} \rrbracket \implies \text{state-rel-1 dfs-state}$   
 $(\text{DFS-upd1 dfs-state})$   
 $\langle \text{proof} \rangle$

**lemma** *state-rel-1-holds-2*[*state-rel-holds-intros*]:  
 $\llbracket \text{DFS-call-2-conds dfs-state}; \text{invar-well-formed dfs-state} \rrbracket \implies \text{state-rel-1 dfs-state}$   
 $(\text{DFS-upd2 dfs-state})$   
 $\langle \text{proof} \rangle$

**lemma** *state-rel-1-holds-4*[*state-rel-holds-intros*]:  
 $\llbracket \text{DFS-ret-1-conds dfs-state} \rrbracket \implies \text{state-rel-1 dfs-state } (\text{DFS-ret1 dfs-state})$   
 $\langle \text{proof} \rangle$

**lemma** *state-rel-1-holds-5*[*state-rel-holds-intros*]:  
 $\llbracket \text{DFS-ret-2-conds dfs-state} \rrbracket \implies \text{state-rel-1 dfs-state } (\text{DFS-ret2 dfs-state})$   
 $\langle \text{proof} \rangle$

**lemma** *state-rel-1-holds*[*state-rel-holds-intros*]:  
**assumes** *DFS-dom dfs-state invar-well-formed dfs-state*  
**shows** *state-rel-1 dfs-state (DFS dfs-state)*  
 $\langle \text{proof} \rangle$

**lemma** *DFS-ret-1*[*ret-holds-intros*]: *DFS-ret-1-conds (dfs-state)  $\implies$  DFS-ret-1-conds*  
 $(\text{DFS-ret1 dfs-state})$   
 $\langle \text{proof} \rangle$

**lemma** *ret1-holds*[*ret-holds-intros*]:  
**assumes** *DFS-dom dfs-state return (DFS dfs-state) = NotReachable*  
**shows** *DFS-ret-1-conds (DFS dfs-state)*  
 $\langle \text{proof} \rangle$

**lemma** *DFS-correct-ret-1*:  
 $\llbracket \text{invar-visited-through-seen dfs-state}; \text{DFS-ret-1-conds dfs-state}; u \in t\text{-set (seen}$   
 $\text{dfs-state}) \rrbracket$

$\implies \nexists p. \text{distinct } p \wedge \text{vwalk-bet } (\text{Graph.digraph-abs } G) \ u \ p \ t$   
 ⟨proof⟩

**lemma** *DFS-ret-2*[*ret-holds-intros*]: *DFS-ret-2-conds* (*dfs-state*)  $\implies$  *DFS-ret-2-conds* (*DFS-ret2 dfs-state*)  
 ⟨proof⟩

**lemma** *ret2-holds*[*ret-holds-intros*]:  
**assumes** *DFS-dom dfs-state return (DFS dfs-state) = Reachable*  
**shows** *DFS-ret-2-conds (DFS dfs-state)*  
 ⟨proof⟩

**lemma** *DFS-correct-ret-2*:  
 [[*invar-stack-walk dfs-state; DFS-ret-2-conds dfs-state*]]  
 $\implies \text{vwalk-bet } (\text{Graph.digraph-abs } G) \ (\text{last } (\text{stack } \text{dfs-state})) \ (\text{rev } (\text{stack } \text{dfs-state})) \ t$   
 ⟨proof⟩

## 6.8 Termination

**named-theorems** *termination-intros*

**declare** *termination-intros*

**lemma** *in-prod-rel*[*intro!, termination-intros*]:  
 [[*f1 a = f1 a'; (a, a') ∈ f2 <\*mlex\*> r*]]  $\implies (a, a') \in (f1 \ <*mlex*> f2 \ <*mlex*> r)$   
 ⟨proof⟩

**definition** *less-rel* =  $\{(x::\text{nat}, y::\text{nat}). x < y\}$

**lemma** *wf-less-rel*[*intro!*]: *wf less-rel*  
 ⟨proof⟩

**lemma** *call-1-measure-nonsym*[*simp*]: (*call-1-measure dfs-state, call-1-measure dfs-state*)  
 $\notin \text{less-rel}$   
 ⟨proof⟩

**lemma** *call-1-terminates*[*termination-intros*]:  
 [[*DFS-call-1-conds dfs-state; invar-well-formed dfs-state; invar-seen-stack dfs-state*]]  
 $\implies$   
 (*DFS-upd1 dfs-state, dfs-state*)  $\in \text{call-1-measure } \ <*mlex*> \ r$   
 ⟨proof⟩

**lemma** *call-2-measure-nonsym*[*simp*]: (*call-2-measure dfs-state, call-2-measure dfs-state*)  
 $\notin \text{less-rel}$   
 ⟨proof⟩

**lemma** *call-2-measure-1*[*termination-intros*]:

[[DFS-call-2-conds dfs-state; invar-well-formed dfs-state]]  $\implies$   
 call-1-measure dfs-state = call-1-measure (DFS-upd2 dfs-state)  
 <proof>

**lemma** call-2-terminates[termination-intros]:  
 [[DFS-call-2-conds dfs-state; invar-well-formed dfs-state; invar-seen-stack dfs-state]]  
 $\implies$   
 (DFS-upd2 dfs-state, dfs-state)  $\in$  call-2-measure <\*mlex\*> r  
 <proof>

**lemma** wf-term-rel: wf DFS-term-rel  
 <proof>

**lemma** in-DFS-term-rel[termination-intros]:  
 [[DFS-call-1-conds dfs-state; invar-well-formed dfs-state; invar-seen-stack dfs-state]]  
 $\implies$   
 (DFS-upd1 dfs-state, dfs-state)  $\in$  DFS-term-rel  
 [[DFS-call-2-conds dfs-state; invar-well-formed dfs-state; invar-seen-stack dfs-state]]  
 $\implies$   
 (DFS-upd2 dfs-state, dfs-state)  $\in$  DFS-term-rel  
 <proof>

**lemma** DFS-terminates[termination-intros]:  
 assumes invar-well-formed dfs-state invar-seen-stack dfs-state  
 shows DFS-dom dfs-state  
 <proof>

## 6.9 Final Correctness Theorems

**lemma** initial-state-props[invar-holds-intros, termination-intros]:  
 invar-well-formed (initial-state) invar-stack-walk (initial-state) invar-seen-stack  
 (initial-state)  
 invar-visited-through-seen (initial-state) invar-s-in-stack initial-state  
 DFS-dom initial-state  
 <proof>

**lemma** DFS-correct-1:  
 assumes return (DFS initial-state) = NotReachable  
 shows  $\nexists p. \text{distinct } p \wedge \text{vwalk-bet } (\text{Graph.digraph-abs } G) s p t$   
 <proof>

**lemma** DFS-correct-2:  
 assumes return (DFS initial-state) = Reachable  
 shows vwalk-bet (Graph.digraph-abs G) s (rev (stack (DFS initial-state))) t  
 <proof>  
 end  
 end  
 end

```

theory BFS-2
  imports Pair-Graph-Specs Dist Set2-Addons More-Lists
begin

```

## 7 Breadth-First Search

### 7.1 The Program State

```

record ('parents, 'vset) BFS-state = parents:: 'parents current:: 'vset visited:: 'vset

```

### 7.2 Setup for Automation

```

named-theorems call-cond-elim
named-theorems call-cond-intros
named-theorems ret-holds-intros
named-theorems invar-props-intros
named-theorems invar-props-elim
named-theorems invar-holds-intros
named-theorems state-rel-intros
named-theorems state-rel-holds-intros

```

### 7.3 A locale for fixing data structures and their implementations

```

locale BFS =
  Graph: Pair-Graph-Specs
  where lookup = lookup +
  set-ops: Set2 vset-empty vset-delete - t-set vset-inv insert

```

```

for lookup :: 'adjmap  $\Rightarrow$  'ver  $\Rightarrow$  'vset option +

```

**fixes**

```

srcs::'vset and
G::'adjmap and expand-tree::'adjmap  $\Rightarrow$  'vset  $\Rightarrow$  'vset  $\Rightarrow$  'adjmap and
next-frontier::'vset  $\Rightarrow$  'vset  $\Rightarrow$  'vset

```

**assumes**

```

  expand-tree[simp]:
   $\llbracket \text{Graph.graph-inv BFS-tree; vset-inv frontier; vset-inv vis; Graph.graph-inv } G \rrbracket$ 
 $\Rightarrow$ 
  Graph.graph-inv (expand-tree BFS-tree frontier vis)
 $\llbracket \text{Graph.graph-inv BFS-tree; vset-inv frontier; vset-inv vis; Graph.graph-inv } G \rrbracket$ 
 $\Rightarrow$ 
  Graph.digraph-abs (expand-tree BFS-tree frontier vis) =
  (Graph.digraph-abs BFS-tree)  $\cup$ 
   $\{(u,v) \mid u \ v. u \in t\text{-set } (\text{frontier}) \wedge$ 
   $v \in (\text{Pair-Graph.neighbourhood } (\text{Graph.digraph-abs } G) \ u -$ 
   $t\text{-set } \text{vis})\}$  and

```

$next\text{-frontier}[simp]:$   
 $\llbracket vset\text{-inv } frontier; vset\text{-inv } vis; Graph.\text{graph-inv } G \rrbracket \implies vset\text{-inv } (next\text{-frontier } frontier \text{ vis})$   
 $\llbracket vset\text{-inv } frontier; vset\text{-inv } vis; Graph.\text{graph-inv } G \rrbracket \implies$   
 $t\text{-set } (next\text{-frontier } frontier \text{ vis}) =$   
 $(\bigcup \{Pair\text{-Graph.neighbourhood } (Graph.\text{digraph-abs } G) \ u \mid u . u \in t\text{-set } frontier\}) - t\text{-set } vis$

**begin**

**definition** *BFS-axiom*  $\longleftrightarrow$

$Graph.\text{graph-inv } G \wedge Graph.\text{finite-graph } G \wedge Graph.\text{finite-vsets } \wedge$   
 $t\text{-set } srcs \subseteq dVs \ (Graph.\text{digraph-abs } G) \wedge$   
 $(\forall u. \text{finite } (Pair\text{-Graph.neighbourhood } (Graph.\text{digraph-abs } G) \ u)) \wedge$   
 $t\text{-set } srcs \neq \{\}$   $\wedge vset\text{-inv } srcs$

**abbreviation**  $neighb' \equiv Graph.\text{neighb } G$

**notation**  $neighb' (\mathcal{N}_G - 100)$

## 7.4 The Algorithm Definition

**function** (*domintros*) *BFS*::('adjmap, 'vset) *BFS-state*  $\Rightarrow$  ('adjmap, 'vset) *BFS-state*  
**where**

$BFS \text{ BFS-state} =$   
 $($   
 $\quad \text{if current BFS-state} \neq \emptyset_V \text{ then}$   
 $\quad \text{let}$   
 $\quad \quad \text{visited}' = \text{visited BFS-state} \cup_G \text{current BFS-state};$   
 $\quad \quad \text{parents}' = \text{expand-tree } (\text{parents BFS-state}) \ (\text{current BFS-state}) \ \text{visited}';$   
 $\quad \quad \text{current}' = \text{next-frontier } (\text{current BFS-state}) \ \text{visited}'$   
 $\quad \text{in}$   
 $\quad \quad BFS \ (\text{BFS-state } (\text{parents} := \text{parents}', \text{visited} := \text{visited}', \text{current} :=$   
 $\text{current}'))$   
 $\quad \text{else}$   
 $\quad \quad BFS\text{-state})$   
 $\langle \text{proof} \rangle$

## 7.5 Setup for Reasoning About the Algorithm

**definition** *BFS-call-1-conds*  $bfs\text{-state} = ( (\text{current } bfs\text{-state}) \neq \emptyset_V)$

**definition** *BFS-upd1* *BFS-state* =

$($   
 $\quad \text{let}$   
 $\quad \quad \text{visited}' = \text{visited BFS-state} \cup_G \text{current BFS-state};$   
 $\quad \quad \text{parents}' = \text{expand-tree } (\text{parents BFS-state}) \ (\text{current BFS-state}) \ \text{visited}';$   
 $\quad \quad \text{current}' = \text{next-frontier } (\text{current BFS-state}) \ \text{visited}'$   
 $\quad \text{in}$   
 $\quad \quad BFS\text{-state } (\text{parents} := \text{parents}', \text{visited} := \text{visited}', \text{current} := \text{current}')$   
 $)$

**definition**  $BFS-ret-1-conds\ bfs-state = ((current\ bfs-state) = \emptyset_V)$

**abbreviation**  $BFS-ret1\ bfs-state \equiv bfs-state$

**lemma**  $DFS-call-1-conds[call-cond-elim]$ :

$BFS-call-1-conds\ bfs-state \implies$   
 $\llbracket (current\ bfs-state) \neq \emptyset_V \implies P \rrbracket$   
 $\implies P$   
 $\langle proof \rangle$

**lemma**  $BFS-ret-1-conds[call-cond-elim]$ :

$BFS-ret-1-conds\ bfs-state \implies$   
 $\llbracket (current\ bfs-state) = \emptyset_V \implies P \rrbracket$   
 $\implies P$   
 $\langle proof \rangle$

**lemma**  $BFS-ret-1-condsI[call-cond-intros]$ :

$\llbracket (current\ bfs-state) = \emptyset_V \rrbracket \implies BFS-ret-1-conds\ bfs-state$   
 $\langle proof \rangle$

**lemma**  $BFS-cases$ :

**assumes**  $BFS-call-1-conds\ bfs-state \implies P$   
 $BFS-ret-1-conds\ bfs-state \implies P$   
**shows**  $P$   
 $\langle proof \rangle$

**lemma**  $BFS-simps$ :

**assumes**  $BFS-dom\ BFS-state$   
**shows**  $BFS-call-1-conds\ BFS-state \implies BFS\ BFS-state = BFS\ (BFS-upd1\ BFS-state)$   
 $BFS-ret-1-conds\ BFS-state \implies BFS\ BFS-state = BFS-ret1\ BFS-state$   
 $\langle proof \rangle$

**lemma**  $BFS-induct$ :

**assumes**  $BFS-dom\ bfs-state$   
**assumes**  $\bigwedge bfs-state. \llbracket BFS-dom\ bfs-state;$   
 $(BFS-call-1-conds\ bfs-state \implies P\ (BFS-upd1\ bfs-state)) \rrbracket$   
 $\implies P\ bfs-state$   
**shows**  $P\ bfs-state$   
 $\langle proof \rangle$

**lemma**  $BFS-domintros$ :

**assumes**  $BFS-call-1-conds\ BFS-state \implies BFS-dom\ (BFS-upd1\ BFS-state)$   
**shows**  $BFS-dom\ BFS-state$   
 $\langle proof \rangle$

## 7.6 The Loop Invariants

**definition** *invar-well-formed*::('adjmap, 'vset) BFS-state  $\Rightarrow$  bool **where**  
*invar-well-formed* bfs-state = (  
     vset-inv (visited bfs-state)  $\wedge$  vset-inv (current bfs-state)  $\wedge$   
     Graph.graph-inv (parents bfs-state)  $\wedge$   
     finite (t-set (current bfs-state))  $\wedge$  finite (t-set (visited bfs-state)))

**definition** *invar-subsets*::('adjmap, 'vset) BFS-state  $\Rightarrow$  bool **where**  
*invar-subsets* bfs-state = (  
     Graph.digraph-abs (parents bfs-state)  $\subseteq$  Graph.digraph-abs G  $\wedge$   
     t-set (visited bfs-state)  $\subseteq$  dVs (Graph.digraph-abs G)  $\wedge$   
     t-set (current bfs-state)  $\subseteq$  dVs (Graph.digraph-abs G)  $\wedge$   
     dVs (Graph.digraph-abs (parents bfs-state))  $\subseteq$  t-set (visited bfs-state)  $\cup$  t-set  
     (current bfs-state)  $\wedge$   
     t-set srcs  $\subseteq$  t-set (visited bfs-state)  $\cup$  t-set (current bfs-state))

**definition** *invar-3-1* bfs-state =  
     ( $\forall v \in$  t-set (current bfs-state).  $\forall u. u \in$  t-set (current bfs-state)  $\longleftrightarrow$   
     distance-set (Graph.digraph-abs G) (t-set srcs) v =  
     distance-set (Graph.digraph-abs G) (t-set srcs) u)

**definition** *invar-3-2* bfs-state =  
     ( $\forall v \in$  t-set (current bfs-state).  $\forall u \in$  t-set (visited bfs-state)  $\cup$  t-set (current bfs-state).  
     distance-set (Graph.digraph-abs G) (t-set srcs) u  $\leq$   
     distance-set (Graph.digraph-abs G) (t-set srcs) v)

**definition** *invar-3-3* bfs-state =  
     ( $\forall v \in$  t-set (visited bfs-state).  
     neighbourhood (Graph.digraph-abs G) v  $\subseteq$  t-set (visited bfs-state)  $\cup$  t-set  
     (current bfs-state))

**definition** *invar-dist-bounded*::('adjmap, 'vset) BFS-state  $\Rightarrow$  bool **where**  
*invar-dist-bounded* bfs-state =  
     ( $\forall v \in$  t-set (visited bfs-state)  $\cup$  t-set (current bfs-state).  
      $\forall u. distance-set (Graph.digraph-abs G) (t-set srcs) u \leq$   
     distance-set (Graph.digraph-abs G) (t-set srcs) v  
      $\longrightarrow u \in$  t-set (visited bfs-state)  $\cup$  t-set (current bfs-state))

**definition** *invar-goes-through-current*::('adjmap, 'vset) BFS-state  $\Rightarrow$  bool **where**  
*invar-goes-through-current* bfs-state =  
     ( $\forall u \in$  t-set (visited bfs-state)  $\cup$  t-set (current bfs-state).  
      $\forall v. v \notin$  t-set (visited bfs-state)  $\cup$  t-set (current bfs-state)  $\longrightarrow$   
     ( $\forall p. Vwalk.vwalk-bet (Graph.digraph-abs G) u p v \longrightarrow$   
     set p  $\cap$  t-set (current bfs-state)  $\neq$  {}))

**definition** *invar-dist*::('adjmap, 'vset) BFS-state  $\Rightarrow$  bool **where**  
*invar-dist* bfs-state =  
     ( $\forall v \in$  dVs (Graph.digraph-abs G) - t-set srcs.  
     (v  $\in$  (t-set (visited bfs-state)  $\cup$  t-set (current bfs-state))  $\longrightarrow$  distance-set

(*Graph.digraph-abs* *G*) (*t-set srcs*) *v* =  
*distance-set* (*Graph.digraph-abs* (*parents bfs-state*)) (*t-set srcs*) *v*)

**definition** *invar-parents-shortest-paths*::('adjmap, 'vset) *BFS-state*  $\Rightarrow$  *bool* **where**  
*invar-parents-shortest-paths* *bfs-state* =  
 ( $\forall u \in t\text{-set } srcs.$   
 $\forall p \ v. Vwalk.vwalk\text{-bet}$  (*Graph.digraph-abs* (*parents bfs-state*)) *u p v*  $\longrightarrow$   
*length* *p* - 1 = *distance-set* (*Graph.digraph-abs* *G*) (*t-set srcs*) *v*)

## 7.7 Termination Measures and Relation

**definition** *call-1-measure-1*::('adjmap, 'vset) *BFS-state*  $\Rightarrow$  *nat* **where**  
*call-1-measure-1* *bfs-state* =  
*card* (*dVs* (*Graph.digraph-abs* *G*) - ((*t-set* (*visited bfs-state*))  $\cup$  *t-set* (*current*  
*bfs-state*)))

**definition** *call-1-measure-2*::('adjmap, 'vset) *BFS-state*  $\Rightarrow$  *nat* **where**  
*call-1-measure-2* *bfs-state* =  
*card* (*t-set* (*current bfs-state*))

**definition** *BFS-term-rel*::(('adjmap, 'vset) *BFS-state*  $\times$  ('adjmap, 'vset) *BFS-state*)  
*set* **where**  
*BFS-term-rel* = *call-1-measure-1*  $\langle *mlex* \rangle$  *call-1-measure-2*  $\langle *mlex* \rangle$  { }

**definition** *initial-state* = (*parents* = *empty*, *current* = *srcs*, *visited* =  $\emptyset_V$ )

**lemmas**[*code*] = *initial-state-def*

**context**

**includes** *Graph.adjmap.automation* **and** *Graph.vset.set.automation* **and** *set-ops.automation*  
**assumes** *BFS-axiom*

**begin**

**lemma** *graph-inv*[*simp*]:

*Graph.graph-inv* *G*

*Graph.finite-graph* *G*

*Graph.finite-vsets* **and**

*srcs-in-G*[*simp,intro*]:

*t-set* *srcs*  $\subseteq$  *dVs* (*Graph.digraph-abs* *G*) **and**

*finite-vset*:

*finite* (*Pair-Graph.neighbourhood* (*Graph.digraph-abs* *G*) *u*) **and**

*srcs-invar*[*simp*]:

*t-set* *srcs*  $\neq$  { }

*vset-inv* *srcs*

$\langle$ *proof* $\rangle$

**lemma** *invar-well-formed-props*[*invar-props-elim*]:

*invar-well-formed* *bfs-state*  $\implies$

( $\llbracket$ *vset-inv* (*visited* *bfs-state*) ; *vset-inv* (*current* *bfs-state*) ;

$Graph.graph-inv$  ( $parents$   $bfs-state$ );  
 $finite$  ( $t-set$  ( $current$   $bfs-state$ ));  $finite$  ( $t-set$  ( $visited$   $bfs-state$ ))]]  $\implies P$   
 $\implies P$   
 <proof>

**lemma** *invar-well-formed-intro*[*invar-props-intros*]:  
 [[ $vset-inv$  ( $visited$   $bfs-state$ );  $vset-inv$  ( $current$   $bfs-state$ );  
 $Graph.graph-inv$  ( $parents$   $bfs-state$ );  
 $finite$  ( $t-set$  ( $current$   $bfs-state$ ));  $finite$  ( $t-set$  ( $visited$   $bfs-state$ ))]]  
 $\implies invar-well-formed$   $bfs-state$   
 <proof>

**lemma** *finite-simp*:  
 $\{(u,v). u \in front \wedge v \in (Pair-Graph.neighbourhood (Graph.digraph-abs G) u) \wedge v \notin vis\} =$   
 $\{(u,v). u \in front\} \cap \{(u,v). v \in (Pair-Graph.neighbourhood (Graph.digraph-abs G) u)\} - \{(u,v) . v \in vis\}$   
 $finite \{(u, v) | v . v \in (s u)\} = finite (s u)$   
 <proof>

**lemma** *invar-well-formed-holds-upd1*[*invar-holds-intros*]:  
 [[ $BFS-call-1-conds$   $bfs-state$ ;  $invar-well-formed$   $bfs-state$ ]]  $\implies invar-well-formed$   
 ( $BFS-upd1$   $bfs-state$ )  
 <proof>

**lemma** *invar-well-formed-holds-ret-1*[*invar-holds-intros*]:  
 [[ $BFS-ret-1-conds$   $bfs-state$ ;  $invar-well-formed$   $bfs-state$ ]]  $\implies invar-well-formed$   
 ( $BFS-ret1$   $bfs-state$ )  
 <proof>

**lemma** *invar-well-formed-holds*[*invar-holds-intros*]:  
**assumes**  $BFS-dom$   $bfs-state$   $invar-well-formed$   $bfs-state$   
**shows**  $invar-well-formed$  ( $BFS$   $bfs-state$ )  
 <proof>

**lemma** *invar-subsets-props*[*invar-props-elim*s]:  
 $invar-subsets$   $bfs-state$   $\implies$   
 ([[ $Graph.digraph-abs$  ( $parents$   $bfs-state$ )  $\subseteq Graph.digraph-abs G$ ;  
 $t-set$  ( $visited$   $bfs-state$ )  $\subseteq dVs$  ( $Graph.digraph-abs G$ );  
 $t-set$  ( $current$   $bfs-state$ )  $\subseteq dVs$  ( $Graph.digraph-abs G$ );  
 $dVs$  ( $Graph.digraph-abs$  ( $parents$   $bfs-state$ ))  $\subseteq t-set$  ( $visited$   $bfs-state$ )  $\cup t-set$   
 ( $current$   $bfs-state$ );  
 $t-set$   $srcs$   $\subseteq t-set$  ( $visited$   $bfs-state$ )  $\cup t-set$  ( $current$   $bfs-state$ )]])  $\implies P$   
 $\implies P$   
 <proof>

**lemma** *invar-subsets-intro*[*invar-props-intros*]:  
 [[ $Graph.digraph-abs$  ( $parents$   $bfs-state$ )  $\subseteq Graph.digraph-abs G$ ;  
 $t-set$  ( $visited$   $bfs-state$ )  $\subseteq dVs$  ( $Graph.digraph-abs G$ );

$t\text{-set } (\text{current bfs-state}) \subseteq dVs \text{ (Graph.digraph-abs } G\text{);}$   
 $dVs \text{ (Graph.digraph-abs (parents bfs-state))} \subseteq t\text{-set (visited bfs-state)} \cup t\text{-set (current bfs-state);}$   
 $t\text{-set srcs} \subseteq t\text{-set (visited bfs-state)} \cup t\text{-set (current bfs-state)}\]]$   
 $\implies \text{invar-subsets bfs-state}$   
 <proof>

**lemma** *invar-subsets-holds-upd1*[*invar-holds-intros*]:  
 $\llbracket \text{BFS-call-1-conds bfs-state; invar-well-formed bfs-state; invar-subsets bfs-state} \rrbracket$   
 $\implies \text{invar-subsets (BFS-upd1 bfs-state)}$   
 <proof>

**lemma** *invar-subsets-holds-ret-1*[*invar-holds-intros*]:  
 $\llbracket \text{BFS-ret-1-conds bfs-state; invar-subsets bfs-state} \rrbracket \implies \text{invar-subsets (BFS-ret1 bfs-state)}$   
 <proof>

**lemma** *invar-subsets-holds*[*invar-holds-intros*]:  
**assumes** *BFS-dom bfs-state invar-well-formed bfs-state invar-subsets bfs-state*  
**shows** *invar-subsets (BFS bfs-state)*  
 <proof>

**lemma** *invar-3-1-props*[*invar-props-elim*]:  
 $\text{invar-3-1 bfs-state} \implies$   
 $\llbracket \llbracket v \in t\text{-set (current bfs-state); } u \in t\text{-set (current bfs-state)} \rrbracket \implies$   
 $\text{distance-set (Graph.digraph-abs } G\text{) (t-set srcs) } v =$   
 $\text{distance-set (Graph.digraph-abs } G\text{) (t-set srcs) } u;$   
 $\llbracket v \in t\text{-set (current bfs-state);}$   
 $\text{distance-set (Graph.digraph-abs } G\text{) (t-set srcs) } v =$   
 $\text{distance-set (Graph.digraph-abs } G\text{) (t-set srcs) } u \rrbracket \implies$   
 $u \in t\text{-set (current bfs-state)} \rrbracket \implies P$   
 $\implies P$   
 <proof>

**lemma** *invar-3-1-intro*[*invar-props-intros*]:  
 $\llbracket \bigwedge u v. \llbracket v \in t\text{-set (current bfs-state); } u \in t\text{-set (current bfs-state)} \rrbracket \implies$   
 $\text{distance-set (Graph.digraph-abs } G\text{) (t-set srcs) } v =$   
 $\text{distance-set (Graph.digraph-abs } G\text{) (t-set srcs) } u;$   
 $\bigwedge u v. \llbracket v \in t\text{-set (current bfs-state); distance-set (Graph.digraph-abs } G\text{) (t-set srcs) } v =$   
 $\text{distance-set (Graph.digraph-abs } G\text{) (t-set srcs) } u \rrbracket \implies$   
 $u \in t\text{-set (current bfs-state)} \rrbracket$   
 $\implies \text{invar-3-1 bfs-state}$   
 <proof>

**lemma** *invar-3-2-props*[*elim*]:  
 $\text{invar-3-2 bfs-state} \implies$   
 $\llbracket \bigwedge v u. \llbracket v \in t\text{-set (current bfs-state); } u \in t\text{-set (visited bfs-state)} \cup t\text{-set (current bfs-state)} \rrbracket \implies$

$$\begin{aligned} & \text{distance-set } (\text{Graph.digraph-abs } G) (t\text{-set srcs}) u \leq \\ & \text{distance-set } (\text{Graph.digraph-abs } G) (t\text{-set srcs}) v \implies P \\ & \implies P \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** *invar-3-2-intro*[*invar-props-intros*]:

$$\begin{aligned} & \llbracket \bigwedge v u. \llbracket v \in t\text{-set } (\text{current bfs-state}); u \in t\text{-set } (\text{visited bfs-state}) \cup t\text{-set } (\text{current} \\ & \text{bfs-state}) \rrbracket \implies \\ & \text{distance-set } (\text{Graph.digraph-abs } G) (t\text{-set srcs}) u \leq \\ & \text{distance-set } (\text{Graph.digraph-abs } G) (t\text{-set srcs}) v \\ & \implies \text{invar-3-2 bfs-state} \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** *invar-3-3-props*[*invar-props-elim*s]:

$$\begin{aligned} & \text{invar-3-3 bfs-state} \implies \\ & \llbracket \bigwedge v. \llbracket v \in t\text{-set } (\text{visited bfs-state}) \rrbracket \implies \\ & \text{neighbourhood } (\text{Graph.digraph-abs } G) v \subseteq t\text{-set } (\text{visited bfs-state}) \cup t\text{-set} \\ & (\text{current bfs-state}) \rrbracket \implies P \\ & \implies P \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** *invar-3-3-intro*[*invar-props-intros*]:

$$\begin{aligned} & \llbracket \bigwedge v. \llbracket v \in t\text{-set } (\text{visited bfs-state}) \rrbracket \implies \\ & \text{neighbourhood } (\text{Graph.digraph-abs } G) v \subseteq t\text{-set } (\text{visited bfs-state}) \cup t\text{-set} \\ & (\text{current bfs-state}) \rrbracket \\ & \implies \text{invar-3-3 bfs-state} \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** *invar-dist-bounded-props*[*invar-props-elim*s]:

$$\begin{aligned} & \text{invar-dist-bounded bfs-state} \implies \\ & \llbracket \bigwedge u v. \llbracket v \in t\text{-set } (\text{visited bfs-state}) \cup t\text{-set } (\text{current bfs-state}); \\ & \text{distance-set } (\text{Graph.digraph-abs } G) (t\text{-set srcs}) u \leq \text{distance-set} \\ & (\text{Graph.digraph-abs } G) (t\text{-set srcs}) v \rrbracket \implies \\ & u \in t\text{-set } (\text{visited bfs-state}) \cup t\text{-set } (\text{current bfs-state}) \rrbracket \implies P \\ & \implies P \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** *invar-dist-bounded-intro*[*invar-props-intros*]:

$$\begin{aligned} & \llbracket \bigwedge u v. \llbracket v \in t\text{-set } (\text{visited bfs-state}) \cup t\text{-set } (\text{current bfs-state}); \\ & \text{distance-set } (\text{Graph.digraph-abs } G) (t\text{-set srcs}) u \leq \text{distance-set} \\ & (\text{Graph.digraph-abs } G) (t\text{-set srcs}) v \rrbracket \implies \\ & u \in t\text{-set } (\text{visited bfs-state}) \cup t\text{-set } (\text{current bfs-state}) \rrbracket \\ & \implies \text{invar-dist-bounded bfs-state} \\ & \langle \text{proof} \rangle \end{aligned}$$

**definition** *invar-current-reachable bfs-state* =

$$\begin{aligned} & (\forall v \in t\text{-set } (\text{visited bfs-state}) \cup t\text{-set } (\text{current bfs-state}). \\ & \text{distance-set } (\text{Graph.digraph-abs } G) (t\text{-set srcs}) v < \infty) \end{aligned}$$

**lemma** *invar-current-reachable-props*[*invar-props-elim*]:  
*invar-current-reachable bfs-state*  $\implies$   
 $\llbracket \bigwedge v. \llbracket v \in t\text{-set } (visited\ bfs\text{-state}) \cup t\text{-set } (current\ bfs\text{-state}) \rrbracket \implies$   
 $\quad distance\text{-set } (Graph.digraph\text{-abs } G) (t\text{-set } srcs) v < \infty \rrbracket \implies P$   
 $\implies P$   
 $\langle proof \rangle$

**lemma** *invar-current-reachable-intro*[*invar-props-intro*]:  
 $\llbracket \bigwedge v. \llbracket v \in t\text{-set } (visited\ bfs\text{-state}) \cup t\text{-set } (current\ bfs\text{-state}) \rrbracket \implies$   
 $\quad distance\text{-set } (Graph.digraph\text{-abs } G) (t\text{-set } srcs) v < \infty \rrbracket$   
 $\implies invar\text{-current-reachable } bfs\text{-state}$   
 $\langle proof \rangle$

## 7.8 Proofs that the Invariants Hold

**lemma** *invar-current-reachable-holds-upd1*[*invar-holds-intro*]:  
 $\llbracket BFS\text{-call-1-conds } bfs\text{-state}; invar\text{-well-formed } bfs\text{-state}; invar\text{-subsets } bfs\text{-state};$   
 $invar\text{-current-reachable } bfs\text{-state} \rrbracket$   
 $\implies invar\text{-current-reachable } (BFS\text{-upd1 } bfs\text{-state})$   
 $\langle proof \rangle$

**lemma** *invar-current-reachable-holds-ret-1*[*invar-holds-intro*]:  
 $\llbracket BFS\text{-ret-1-conds } bfs\text{-state}; invar\text{-current-reachable } bfs\text{-state} \rrbracket \implies invar\text{-current-reachable}$   
 $(BFS\text{-ret1 } bfs\text{-state})$   
 $\langle proof \rangle$

**lemma** *dist-current-plus-1-new*:  
**assumes**  
 $invar\text{-well-formed } bfs\text{-state } invar\text{-subsets } bfs\text{-state } invar\text{-dist-bounded } bfs\text{-state}$   
 $v \in neighbourhood (Graph.digraph\text{-abs } G) v'$   
 $v' \in t\text{-set } (current\ bfs\text{-state})$   
 $v \in t\text{-set } (current\ (BFS\text{-upd1 } bfs\text{-state}))$   
**shows**  $distance\text{-set } (Graph.digraph\text{-abs } G) (t\text{-set } srcs) v =$   
 $distance\text{-set } (Graph.digraph\text{-abs } G) (t\text{-set } srcs) v' + 1$  (**is**  $?dv = ?dv' +$   
 $1$ )  
 $\langle proof \rangle$

**lemma** *plus-lt-enat*:  $\llbracket (a::enat) \neq \infty; b < c \rrbracket \implies a + b < a + c$   
 $\langle proof \rangle$

**lemma** *plus-one-side-lt-enat*:  $\llbracket (a::enat) \neq \infty; 0 < b \rrbracket \implies a < a + b$   
 $\langle proof \rangle$

**lemma** *invar-3-1-holds-upd1-new*[*invar-holds-intro*]:  
 $\llbracket BFS\text{-call-1-conds } bfs\text{-state}; invar\text{-well-formed } bfs\text{-state}; invar\text{-subsets } bfs\text{-state};$   
 $invar\text{-3-1 } bfs\text{-state};$   
 $invar\text{-3-2 } bfs\text{-state}; invar\text{-dist-bounded } bfs\text{-state}; invar\text{-current-reachable } bfs\text{-state} \rrbracket$   
 $\implies invar\text{-3-1 } (BFS\text{-upd1 } bfs\text{-state})$   
 $\langle proof \rangle$

**lemma** *invar-3-1-holds-ret-1*[*invar-holds-intros*]:  
 $\llbracket \text{BFS-ret-1-conds } \text{bfs-state}; \text{invar-3-3 } \text{bfs-state} \rrbracket \implies \text{invar-3-3 } (\text{BFS-ret1 } \text{bfs-state})$   
 ⟨*proof*⟩

**lemma** *invar-3-2-holds-upd1-new*[*invar-holds-intros*]:  
 $\llbracket \text{BFS-call-1-conds } \text{bfs-state}; \text{invar-well-formed } \text{bfs-state}; \text{invar-subsets } \text{bfs-state};$   
 $\text{invar-3-1 } \text{bfs-state};$   
 $\text{invar-3-2 } \text{bfs-state}; \text{invar-dist-bounded } \text{bfs-state}; \text{invar-current-reachable } \text{bfs-state} \rrbracket$   
 $\implies \text{invar-3-2 } (\text{BFS-upd1 } \text{bfs-state})$   
 ⟨*proof*⟩

**lemma** *invar-3-2-holds-ret-1*[*invar-holds-intros*]:  
 $\llbracket \text{BFS-ret-1-conds } \text{bfs-state}; \text{invar-3-3 } \text{bfs-state} \rrbracket \implies \text{invar-3-3 } (\text{BFS-ret1 } \text{bfs-state})$   
 ⟨*proof*⟩

**lemma** *invar-3-3-holds-upd1*[*invar-holds-intros*]:  
 $\llbracket \text{BFS-call-1-conds } \text{bfs-state}; \text{invar-well-formed } \text{bfs-state}; \text{invar-subsets } \text{bfs-state};$   
 $\text{invar-3-3 } \text{bfs-state} \rrbracket \implies \text{invar-3-3 } (\text{BFS-upd1 } \text{bfs-state})$   
 ⟨*proof*⟩

**lemma** *invar-3-3-holds-ret-1*[*invar-holds-intros*]:  
 $\llbracket \text{BFS-ret-1-conds } \text{bfs-state}; \text{invar-3-3 } \text{bfs-state} \rrbracket \implies \text{invar-3-3 } (\text{BFS-ret1 } \text{bfs-state})$   
 ⟨*proof*⟩

**lemma** *invar-dist-bounded-holds-upd1*[*invar-holds-intros*]:  
 $\llbracket \text{BFS-call-1-conds } \text{bfs-state}; \text{invar-well-formed } \text{bfs-state}; \text{invar-subsets } \text{bfs-state};$   
 $\text{invar-3-1 } \text{bfs-state}; \text{invar-3-2 } \text{bfs-state}; \text{invar-3-3 } \text{bfs-state}; \text{invar-dist-bounded}$   
 $\text{bfs-state};$   
 $\text{invar-current-reachable } \text{bfs-state} \rrbracket \implies$   
 $\text{invar-dist-bounded } (\text{BFS-upd1 } \text{bfs-state})$   
 ⟨*proof*⟩

**lemma** *invar-dist-bounded-holds-ret-1*[*invar-holds-intros*]:  
 $\llbracket \text{BFS-ret-1-conds } \text{bfs-state}; \text{invar-dist-bounded } \text{bfs-state} \rrbracket \implies \text{invar-dist-bounded}$   
 $(\text{BFS-ret1 } \text{bfs-state})$   
 ⟨*proof*⟩

**lemma** *invar-dist-props*[*invar-props-elim*s]:  
 $\text{invar-dist } \text{bfs-state} \implies v \in dVs (\text{Graph.digraph-abs } G) - t\text{-set } \text{srcs} \implies$   
 $\llbracket$   
 $\llbracket v \in (t\text{-set } (\text{visited } \text{bfs-state}) \cup t\text{-set } (\text{current } \text{bfs-state})) \implies \text{distance-set}$   
 $(\text{Graph.digraph-abs } G) (t\text{-set } \text{srcs}) v =$   
 $\text{distance-set } (\text{Graph.digraph-abs } (\text{parents } \text{bfs-state})) (t\text{-set } \text{srcs}) v \rrbracket \implies P$   
 $\rrbracket$   
 $\implies P$   
 ⟨*proof*⟩

**lemma** *invar-dist-intro*[*invar-props-intros*]:

$$\begin{aligned} & \llbracket \bigwedge v. \llbracket v \in dVs \text{ (Graph.digraph-abs } G) - t\text{-set srcs; } v \in t\text{-set (visited bfs-state)} \cup \\ & t\text{-set (current bfs-state)} \rrbracket \implies \\ & \quad (\text{distance-set (Graph.digraph-abs } G) \text{ (t-set srcs) } v = \\ & \quad \text{distance-set (Graph.digraph-abs (parents bfs-state)) (t-set srcs) } v) \rrbracket \\ & \implies \text{invar-dist bfs-state} \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** *invar-goes-through-current-props*[*invar-props-elim*]:

$$\begin{aligned} & \text{invar-goes-through-current bfs-state} \implies \\ & \llbracket \bigwedge u v p. \llbracket u \in t\text{-set (visited bfs-state)} \cup t\text{-set (current bfs-state); } \\ & \quad v \notin t\text{-set (visited bfs-state)} \cup t\text{-set (current bfs-state); } \\ & \quad \text{Vwalk.vwalk-bet (Graph.digraph-abs } G) \text{ } u \text{ } p \text{ } v \rrbracket \\ & \implies \text{set } p \cap t\text{-set (current bfs-state) } \neq \{\} \rrbracket \\ & \implies P \rrbracket \\ & \implies P \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** *invar-goes-through-current-intro*[*invar-props-intros*]:

$$\begin{aligned} & \llbracket \bigwedge u v p. \llbracket u \in t\text{-set (visited bfs-state)} \cup t\text{-set (current bfs-state); } \\ & \quad v \notin t\text{-set (visited bfs-state)} \cup t\text{-set (current bfs-state); } \\ & \quad \text{Vwalk.vwalk-bet (Graph.digraph-abs } G) \text{ } u \text{ } p \text{ } v \rrbracket \\ & \implies \text{set } p \cap t\text{-set (current bfs-state) } \neq \{\} \rrbracket \\ & \implies \text{invar-goes-through-current bfs-state} \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** *invar-goes-through-active-holds-upd1*[*invar-holds-intros*]:

$$\begin{aligned} & \llbracket \text{BFS-call-1-conds bfs-state; invar-well-formed bfs-state; invar-subsets bfs-state; } \\ & \quad \text{invar-goes-through-current bfs-state} \rrbracket \\ & \implies \text{invar-goes-through-current (BFS-upd1 bfs-state)} \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** *invar-goes-through-current-holds-ret-1*[*invar-holds-intros*]:

$$\begin{aligned} & \llbracket \text{BFS-ret-1-conds bfs-state; invar-goes-through-current bfs-state} \rrbracket \implies \text{invar-goes-through-current} \\ & (\text{BFS-ret1 bfs-state}) \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** *invar-goes-through-current-holds*[*invar-holds-intros*]:

$$\begin{aligned} & \text{assumes } \text{BFS-dom bfs-state invar-well-formed bfs-state invar-subsets bfs-state} \\ & \quad \text{invar-goes-through-current bfs-state} \\ & \text{shows } \text{invar-goes-through-current (BFS bfs-state)} \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** *invar-dist-holds-upd1-new*[*invar-holds-intros*]:

$$\begin{aligned} & \llbracket \text{BFS-call-1-conds bfs-state; invar-well-formed bfs-state; invar-subsets bfs-state; } \\ & \quad \text{invar-dist-bounded bfs-state; invar-dist bfs-state} \rrbracket \\ & \implies \text{invar-dist (BFS-upd1 bfs-state)} \end{aligned}$$

*<proof>*

**lemma** *invar-dist-holds-ret-1*[*invar-holds-intros*]:

$\llbracket \text{BFS-ret-1-conds } bfs\text{-state}; \text{invar-dist } bfs\text{-state} \rrbracket \implies \text{invar-dist } (\text{BFS-ret1 } bfs\text{-state})$

*<proof>*

**lemma** *invar-dist-holds*[*invar-holds-intros*]:

**assumes** *BFS-dom* *bfs-state* *invar-well-formed* *bfs-state* *invar-subsets* *bfs-state*  
*invar-3-1* *bfs-state* *invar-3-2* *bfs-state* *invar-3-3* *bfs-state* *invar-dist-bounded*

*bfs-state*

*invar-dist* *bfs-state* *invar-current-reachable* *bfs-state*

**shows** *invar-dist* (*BFS* *bfs-state*)

*<proof>*

**definition** *invar-current-no-out* *bfs-state* =

$(\forall u \in t\text{-set}(\text{current } bfs\text{-state}).$

$\forall v. (u, v) \notin \text{Graph.digraph-abs } (\text{parents } bfs\text{-state}))$

**lemma** *invar-current-no-out-props*[*invar-props-elim*s]:

*invar-current-no-out* *bfs-state*  $\implies$

$(\llbracket \bigwedge u v. u \in t\text{-set}(\text{current } bfs\text{-state}) \implies (u, v) \notin \text{Graph.digraph-abs } (\text{parents } bfs\text{-state}) \rrbracket$   
 $\implies P)$

$\implies P$

*<proof>*

**lemma** *invar-current-no-out-intro*[*invar-props-intros*]:

$\llbracket \bigwedge u v. u \in t\text{-set}(\text{current } bfs\text{-state}) \implies (u, v) \notin \text{Graph.digraph-abs } (\text{parents } bfs\text{-state}) \rrbracket$

$\implies \text{invar-current-no-out } bfs\text{-state}$

*<proof>*

**lemma** *invar-current-no-out-holds-upd1*[*invar-holds-intros*]:

$\llbracket \text{BFS-call-1-conds } bfs\text{-state}; \text{invar-well-formed } bfs\text{-state}; \text{invar-subsets } bfs\text{-state};$   
*invar-current-no-out* *bfs-state*  $\rrbracket$

$\implies \text{invar-current-no-out } (\text{BFS-upd1 } bfs\text{-state})$

*<proof>*

**lemma** *invar-current-no-out-holds-ret-1*[*invar-holds-intros*]:

$\llbracket \text{BFS-ret-1-conds } bfs\text{-state}; \text{invar-current-no-out } bfs\text{-state} \rrbracket \implies \text{invar-current-no-out}$   
 $(\text{BFS-ret1 } bfs\text{-state})$

*<proof>*

**lemma** *invar-current-no-out-holds*[*invar-holds-intros*]:

**assumes** *BFS-dom* *bfs-state* *invar-well-formed* *bfs-state* *invar-subsets* *bfs-state*  
*invar-current-no-out* *bfs-state*

**shows** *invar-current-no-out* (*BFS* *bfs-state*)

*<proof>*

**lemma** *invar-parents-shortest-paths-props*[*invar-props-elim*s]:

*invar-parents-shortest-paths* *bfs-state*  $\implies$

$$\begin{aligned} & \llbracket \bigwedge u p v. \llbracket u \in t\text{-set srcs}; V\text{walk.vwalk-bet } (Graph.digraph-abs (parents bfs\text{-state})) \\ & u p v \rrbracket \implies \\ & \quad \text{length } p - 1 = \text{distance-set } (Graph.digraph-abs G) (t\text{-set srcs}) v \rrbracket \implies P \\ & \implies P \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** *invar-parents-shortest-paths-intro*[*invar-props-intros*]:  

$$\begin{aligned} & \llbracket \bigwedge u p v. \llbracket u \in t\text{-set srcs}; V\text{walk.vwalk-bet } (Graph.digraph-abs (parents bfs\text{-state})) \\ & u p v \rrbracket \implies \\ & \quad \text{length } p - 1 = \text{distance-set } (Graph.digraph-abs G) (t\text{-set srcs}) v \rrbracket \\ & \implies \text{invar-parents-shortest-paths } bfs\text{-state} \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** *invar-parents-shortest-paths-holds-upd1*[*invar-holds-intros*]:  

$$\begin{aligned} & \llbracket \text{BFS-call-1-conds } bfs\text{-state}; \text{invar-well-formed } bfs\text{-state}; \text{invar-subsets } bfs\text{-state}; \\ & \text{invar-current-no-out } bfs\text{-state}; \\ & \quad \text{invar-dist-bounded } bfs\text{-state}; \text{invar-parents-shortest-paths } bfs\text{-state} \rrbracket \\ & \implies \text{invar-parents-shortest-paths } (\text{BFS-upd1 } bfs\text{-state}) \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** *invar-parents-shortest-paths-holds-ret-1*[*invar-holds-intros*]:  

$$\begin{aligned} & \llbracket \text{BFS-ret-1-conds } bfs\text{-state}; \text{invar-parents-shortest-paths } bfs\text{-state} \rrbracket \implies \\ & \quad \text{invar-parents-shortest-paths } (\text{BFS-ret1 } bfs\text{-state}) \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** *invar-parents-shortest-paths-holds*[*invar-holds-intros*]:  
**assumes** *BFS-dom* *bfs-state* *invar-well-formed* *bfs-state* *invar-subsets* *bfs-state*  
*invar-current-no-out* *bfs-state* *invar-3-1* *bfs-state*  
*invar-3-2* *bfs-state* *invar-3-3* *bfs-state*  
*invar-dist-bounded* *bfs-state* *invar-current-reachable* *bfs-state*  
*invar-parents-shortest-paths* *bfs-state*  
**shows** *invar-parents-shortest-paths* (*BFS* *bfs-state*)  
 $\langle \text{proof} \rangle$

**lemma** *BFS-ret-1*[*ret-holds-intros*]:  

$$\text{BFS-ret-1-conds } (bfs\text{-state}) \implies \text{BFS-ret-1-conds } (\text{BFS-ret1 } bfs\text{-state})$$
 $\langle \text{proof} \rangle$

**lemma** *ret1-holds*[*ret-holds-intros*]:  
**assumes** *BFS-dom* *bfs-state*  
**shows** *BFS-ret-1-conds* (*BFS* *bfs-state*)  
 $\langle \text{proof} \rangle$

**lemma** *BFS-correct-1-ret-1*:  

$$\begin{aligned} & \llbracket \text{invar-subsets } bfs\text{-state}; \text{invar-goes-through-current } bfs\text{-state}; \text{BFS-ret-1-conds } bfs\text{-state}; \\ & u \in t\text{-set srcs}; t \notin t\text{-set } (visited\text{-state}) \rrbracket \\ & \implies \nexists p. \text{vwalk-bet } (Graph.digraph-abs G) u p t \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** *BFS-correct-2-ret-1*:

$\llbracket \text{invar-well-formed bfs-state}; \text{invar-subsets bfs-state}; \text{invar-dist bfs-state}; \text{BFS-ret-1-conds bfs-state};$

$t \in t\text{-set } (\text{visited bfs-state}) - t\text{-set srcs} \rrbracket$   
 $\implies \text{distance-set } (\text{Graph.digraph-abs } G) (t\text{-set srcs}) t =$   
 $\text{distance-set } (\text{Graph.digraph-abs } (\text{parents bfs-state})) (t\text{-set srcs}) t$   
 $\langle \text{proof} \rangle$

**lemma** *BFS-correct-3-ret-1*:

$\llbracket \text{invar-parents-shortest-paths bfs-state}; \text{BFS-ret-1-conds bfs-state};$   
 $u \in t\text{-set srcs}; \text{Vwalk.vwalk-bet } (\text{Graph.digraph-abs } (\text{parents bfs-state})) u p v \rrbracket$   
 $\implies \text{length } p - 1 = \text{distance-set } (\text{Graph.digraph-abs } G) (t\text{-set srcs}) v$   
 $\langle \text{proof} \rangle$

## 7.9 Termination

**named-theorems** *termination-intros*

**declare** *termination-intros*[*intro!*]

**lemma** *in-prod-rell*[*intro!,termination-intros*]:

$\llbracket f1 a = f1 a'; (a, a') \in f2 \langle *mlex* \rangle r \rrbracket \implies (a, a') \in (f1 \langle *mlex* \rangle f2 \langle *mlex* \rangle$   
 $r)$   
 $\langle \text{proof} \rangle$

**definition** *less-rel* =  $\{(x::nat, y::nat). x < y\}$

**lemma** *wf-less-rel*[*intro!*]: *wf less-rel*

$\langle \text{proof} \rangle$

**definition** *state-measure-rel call-measure* = *inv-image less-rel call-measure*

**lemma** *call-1-measure-nonsym*[*simp*]:

$(\text{call-1-measure-1 BFS-state}, \text{call-1-measure-1 BFS-state}) \notin \text{less-rel}$   
 $\langle \text{proof} \rangle$

**lemma** *call-1-terminates*[*termination-intros*]:

**assumes** *BFS-call-1-conds BFS-state invar-well-formed BFS-state invar-subsets*  
*BFS-state*

*invar-current-no-out BFS-state*

**shows**  $(\text{BFS-upd1 BFS-state}, \text{BFS-state}) \in$

*call-1-measure-1*  $\langle *mlex* \rangle$  *call-1-measure-2*  $\langle *mlex* \rangle r$

$\langle \text{proof} \rangle$

**including** *Graph.adjmap.automation* **and** *Graph.vset.set.automation*

$\langle \text{proof} \rangle$

**lemma** *wf-term-rel*: *wf BFS-term-rel*

$\langle \text{proof} \rangle$

**lemma** *in-BFS-term-rel*[*termination-intros*]:  
 $\llbracket \text{BFS-call-1-conds } \text{BFS-state}; \text{invar-well-formed } \text{BFS-state}; \text{invar-subsets } \text{BFS-state}; \text{invar-current-no-out } \text{BFS-state} \rrbracket \implies$   
 $(\text{BFS-upd1 } \text{BFS-state}, \text{BFS-state}) \in \text{BFS-term-rel}$   
 ⟨*proof*⟩

**lemma** *BFS-terminates*[*termination-intros*]:  
**assumes** *invar-well-formed BFS-state invar-subsets BFS-state invar-current-no-out BFS-state*  
**shows** *BFS-dom BFS-state*  
 ⟨*proof*⟩

**lemma** *not-vwalk-bet-empty*[*simp*]:  $\neg \text{Vwalk.vwalk-bet } (\text{Graph.digraph-abs empty})$   
 $u \ p \ v$   
 ⟨*proof*⟩

**lemma** *not-edge-in-empty*[*simp*]:  $(u, v) \notin (\text{Graph.digraph-abs empty})$   
 ⟨*proof*⟩

## 7.10 Final Correctness Theorems

**lemma** *initial-state-props*[*invar-holds-intros, termination-intros, simp*]:  
*invar-well-formed (initial-state) (is ?g1)*  
*invar-subsets (initial-state) (is ?g2)*  
*invar-current-no-out (initial-state) (is ?g3)*  
*BFS-dom initial-state (is ?g4)*  
*invar-dist initial-state (is ?g5)*  
*invar-3-1 initial-state*  
*invar-3-2 initial-state*  
*invar-3-3 initial-state*  
*invar-dist-bounded initial-state*  
*invar-current-reachable initial-state*  
*invar-goes-through-current initial-state*  
*invar-current-no-out initial-state*  
*invar-parents-shortest-paths initial-state*  
 ⟨*proof*⟩

**lemma** *BFS-correct-1*:  
 $\llbracket u \in t\text{-set } \text{srcs}; t \notin t\text{-set } (\text{visited } (\text{BFS } \text{initial-state})) \rrbracket$   
 $\implies \nexists p. \text{vwalk-bet } (\text{Graph.digraph-abs } G) \ u \ p \ t$   
 ⟨*proof*⟩

**lemma** *BFS-correct-2*:  
 $\llbracket t \in t\text{-set } (\text{visited } (\text{BFS } \text{initial-state})) - t\text{-set } \text{srcs} \rrbracket$   
 $\implies \text{distance-set } (\text{Graph.digraph-abs } G) \ (t\text{-set } \text{srcs}) \ t =$   
 $\text{distance-set } (\text{Graph.digraph-abs } (\text{parents } (\text{BFS } \text{initial-state}))) \ (t\text{-set } \text{srcs}) \ t$   
 ⟨*proof*⟩

**lemma** *BFS-correct-3*:

$\llbracket u \in t\text{-set } srcs; Vwalk.vwalk\text{-bet } (Graph.digraph\text{-abs } (parents (BFS\ initial\text{-state})))$   
 $u\ p\ v \rrbracket \implies length\ p - 1 = distance\text{-set } (Graph.digraph\text{-abs } G) (t\text{-set } srcs)\ v$   
 $\langle proof \rangle$

**end**

context

**end**

locale *BFS*

**end**