

# Tame Plane Graphs

Gertrud Bauer and Tobias Nipkow

April 19, 2020

## Abstract

These theories present the verified enumeration of *tame* plane graphs as defined by Thomas C. Hales in his revised proof of the Kepler Conjecture. Compared with his original proof, the notion of tameness has become simpler, there are many more tame graphs, but much of the earlier verification [1] carries over. For more details see <http://code.google.com/p/flyspeck/> and the forthcoming book “Dense Sphere Packings: A Blueprint for Formal Proofs” by Hales.

## Contents

<b>1</b>	<b>Basic Functions Old and New</b>	<b>5</b>
1.1	HOL . . . . .	5
1.2	Lists . . . . .	5
1.3	<i>splitAt</i> . . . . .	10
1.4	<i>between</i> . . . . .	16
1.5	Tables . . . . .	17
<b>2</b>	<b>Isomorphisms Between Plane Graphs</b>	<b>19</b>
2.1	Equivalence of faces . . . . .	20
2.2	Homomorphism and isomorphism . . . . .	21
2.3	Isomorphism tests . . . . .	22
2.4	Elementhood and containment modulo . . . . .	28
<b>3</b>	<b>More Rotation</b>	<b>28</b>
<b>4</b>	<b>Graph</b>	<b>29</b>
4.1	Notation . . . . .	29
4.2	Faces . . . . .	29
4.3	Graphs . . . . .	31
4.4	Operations on graphs . . . . .	32
4.5	Navigation in graphs . . . . .	33
4.6	Code generator setup . . . . .	33

<b>5</b>	<b>Syntax for operations on immutable arrays</b>	<b>34</b>
5.1	Tabulation . . . . .	34
5.2	Access . . . . .	35
<b>6</b>	<b>Enumerating Patches</b>	<b>35</b>
<b>7</b>	<b>Subdividing a Face</b>	<b>37</b>
<b>8</b>	<b>Transitive Closure of Successor List Function</b>	<b>38</b>
<b>9</b>	<b>Plane Graph Enumeration</b>	<b>39</b>
<b>10</b>	<b>Properties of Graph Utilities</b>	<b>41</b>
10.1	<i>nextElem</i> . . . . .	42
10.2	<i>nextVertex</i> . . . . .	43
10.3	$\mathcal{E}$ . . . . .	43
10.4	Triangles . . . . .	44
10.5	Quadrilaterals . . . . .	44
10.6	No loops . . . . .	45
10.7	<i>between</i> . . . . .	45
<b>11</b>	<b>Properties of Patch Enumeration</b>	<b>46</b>
<b>12</b>	<b>Properties of Face Division</b>	<b>50</b>
12.1	Finality . . . . .	50
12.2	<i>is-prefix</i> . . . . .	51
12.3	<i>is-sublist</i> . . . . .	52
12.4	<i>is-nextElem</i> . . . . .	54
12.5	<i>nextElem</i> , <i>sublist</i> , <i>is-nextElem</i> . . . . .	55
12.6	<i>before</i> . . . . .	55
12.7	<i>between</i> . . . . .	57
12.8	<i>split-face</i> . . . . .	60
12.9	<i>verticesFrom</i> . . . . .	61
12.10	<i>splitFace</i> . . . . .	64
12.11	<i>removeNones</i> . . . . .	72
12.12	<i>natToVertexList</i> . . . . .	73
12.13	<i>indexToVertexList</i> . . . . .	73
12.14	<i>pre-subdivFace</i> (') . . . . .	75
<b>13</b>	<b>Invariants of (Plane) Graphs</b>	<b>80</b>
13.1	Rotation of face into normal form . . . . .	80
13.2	Minimal (plane) graph properties . . . . .	80
13.3	<i>containsDuplicateEdge</i> . . . . .	85
13.4	<i>replacefacesAt</i> . . . . .	85
13.5	<i>normFace</i> . . . . .	87

13.6	Invariants of <i>splitFace</i> . . . . .	89
13.7	Invariants of <i>makeFaceFinal</i> . . . . .	92
13.8	Invariants of <i>subdivFace'</i> . . . . .	93
13.9	Invariants of <i>Seed</i> . . . . .	94
13.10	Increasing properties of <i>subdivFace'</i> . . . . .	95
13.11	Main invariant theorems . . . . .	96
<b>14</b>	<b>Further Plane Graph Properties</b>	<b>97</b>
14.1	<i>final</i> . . . . .	97
14.2	<i>degree</i> . . . . .	97
14.3	Misc . . . . .	98
14.4	Increasing final faces . . . . .	98
14.5	Increasing vertices . . . . .	99
14.6	Increasing vertex degrees . . . . .	99
14.7	Increasing <i>except</i> . . . . .	99
14.8	Increasing edges . . . . .	99
14.9	Increasing final vertices . . . . .	99
14.10	Preservation of <i>facesAt</i> at final vertices . . . . .	100
14.11	Properties of <i>subdivFace'</i> . . . . .	100
<b>15</b>	<b>Summation Over Lists</b>	<b>101</b>
<b>16</b>	<b>Tameness</b>	<b>102</b>
16.1	Constants . . . . .	102
16.2	Separated sets of vertices . . . . .	103
16.3	Admissible weight assignments . . . . .	104
16.4	Tameness . . . . .	104
<b>17</b>	<b>Enumeration of Tame Plane Graphs</b>	<b>105</b>
<b>18</b>	<b>Tame Properties</b>	<b>107</b>
<b>19</b>	<b>Neglectable Final Graphs</b>	<b>108</b>
<b>20</b>	<b>Properties of Lower Bound Machinery</b>	<b>109</b>
<b>21</b>	<b>Correctness of Lower Bound for Final Graphs</b>	<b>114</b>
<b>22</b>	<b>Properties of Tame Graph Enumeration (1)</b>	<b>114</b>
<b>23</b>	<b>Properties of Tame Graph Enumeration (2)</b>	<b>117</b>
<b>24</b>	<b>Archive</b>	<b>123</b>
<b>25</b>	<b>Comparing Enumeration and Archive</b>	<b>123</b>

<b>26 Completeness of Archive Test</b>	<b>124</b>
<b>27 Completeness Proofs under hypothetical computations</b>	<b>126</b>
<b>Bibliography</b>	<b>128</b>

# 1 Basic Functions Old and New

```
theory ListAux  
imports Main  
begin
```

```
declare Let-def[simp]
```

## 1.1 HOL

```
lemma pairD:  $(a,b) = p \implies a = \text{fst } p \wedge b = \text{snd } p$   
 $\langle \text{proof} \rangle$ 
```

```
lemmas conj-aci = conj-comms conj-assoc conj-absorb conj-left-absorb
```

```
definition enum :: nat  $\Rightarrow$  nat set where  
[code-abbrev]: enum n = {.. $n$ }
```

```
lemma [code]:  
enum 0 = {}  
enum (Suc n) = insert n (enum n)  
 $\langle \text{proof} \rangle$ 
```

## 1.2 Lists

```
declare List.member-def[simp] list-all-iff[simp] list-ex-iff[simp]
```

### 1.2.1 length

```
notation length (| $-$ |)
```

```
lemma length3D:  $|xs| = 3 \implies \exists x y z. xs = [x, y, z]$   
 $\langle \text{proof} \rangle$ 
```

```
lemma length4D:  $|xs| = 4 \implies \exists a b c d. xs = [a, b, c, d]$   
 $\langle \text{proof} \rangle$ 
```

### 1.2.2 filter

```
lemma filter-emptyE[dest]:  $(\text{filter } P xs = []) \implies x \in \text{set } xs \implies \neg P x$   
 $\langle \text{proof} \rangle$ 
```

```
lemma filter-comm:  $[x \leftarrow xs. P x \wedge Q x] = [x \leftarrow xs. Q x \wedge P x]$   
 $\langle \text{proof} \rangle$ 
```

```
lemma filter-prop:  $x \in \text{set } [u \leftarrow ys . P u] \implies P x$   
 $\langle \text{proof} \rangle$ 
```

```
lemma filter-compl1:
```

$([x \leftarrow xs. P x] = []) = ([x \leftarrow xs. \neg P x] = xs)$  (**is** ?lhs = ?rhs)  
 <proof>

**lemma** [simp]:  $Not \circ (Not \circ P) = P$   
 <proof>

**lemma** filter-eqI:

$(\bigwedge v. v \in set\ vs \implies P\ v = Q\ v) \implies [v \leftarrow vs . P\ v] = [v \leftarrow vs . Q\ v]$   
 <proof>

**lemma** filter-simp:  $(\bigwedge x. x \in set\ xs \implies P\ x) \implies [x \leftarrow xs. P\ x \wedge Q\ x] = [x \leftarrow xs. Q\ x]$   
 <proof>

**lemma** filter-True-eq1:

$(length\ [y \leftarrow xs. P\ y] = length\ xs) \implies (\bigwedge y. y \in set\ xs \implies P\ y)$   
 <proof>

**lemma** [simp]:  $[f\ x. x <- xs, P\ x] = [f\ x. x <- [x \leftarrow xs. P\ x]]$   
 <proof>

### 1.2.3 concat

**syntax**

-concat ::  $idt \Rightarrow 'a\ list \Rightarrow 'a\ list \Rightarrow 'a\ list$  ( $\lfloor \_ \rfloor \_ \_ \_ 10$ )

**translations**

$\lfloor \_ \rfloor_{x \in xs} f == CONST\ concat\ [f.\ x <- xs]$

### 1.2.4 List product

**definition** listProd1 ::  $'a \Rightarrow 'b\ list \Rightarrow ('a \times 'b)\ list$  **where**  
 $listProd1\ a\ bs \equiv [(a,b). b <- bs]$

**definition** listProd ::  $'a\ list \Rightarrow 'b\ list \Rightarrow ('a \times 'b)\ list$  (**infix**  $\times 50$ ) **where**  
 $as \times bs \equiv \lfloor \_ \rfloor_{a \in as} listProd1\ a\ bs$

**lemma** set  $(xs \times ys) = (set\ xs) \times (set\ ys)$   
 <proof>

### 1.2.5 Minimum and maximum

**primrec** minimal::  $('a \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow 'a$  **where**  
 $minimal\ m\ (x \# xs) =$   
 (if  $xs = []$  then  $x$  else  
 let  $mxs = minimal\ m\ xs$  in  
 if  $m\ x \leq m\ mxs$  then  $x$  else  $mxs$ )

**lemma** minimal-in-set[simp]:  $xs \neq [] \implies minimal\ f\ xs : set\ xs$   
 <proof>

**primrec** *min-list* :: *nat list*  $\Rightarrow$  *nat* **where**  
*min-list* (*x#xs*) = (if *xs*=[] then *x* else *min* *x* (*min-list* *xs*))

**primrec** *max-list* :: *nat list*  $\Rightarrow$  *nat* **where**  
*max-list* (*x#xs*) = (if *xs*=[] then *x* else *max* *x* (*max-list* *xs*))

**lemma** *min-list-conv-Min*[*simp*]:  
*xs*  $\neq$  []  $\implies$  *min-list* *xs* = *Min* (*set* *xs*)  
 $\langle$ *proof* $\rangle$

**lemma** *max-list-conv-Max*[*simp*]:  
*xs*  $\neq$  []  $\implies$  *max-list* *xs* = *Max* (*set* *xs*)  
 $\langle$ *proof* $\rangle$

### 1.2.6 replace

**primrec** *replace* :: '*a*  $\Rightarrow$  '*a list*  $\Rightarrow$  '*a list*  $\Rightarrow$  '*a list* **where**  
*replace* *x* *ys* [] = []  
| *replace* *x* *ys* (*z#zs*) =  
(if *z* = *x* then *ys* @ *zs* else *z* # (*replace* *x* *ys* *zs*))

**primrec** *mapAt* :: *nat list*  $\Rightarrow$  ('*a*  $\Rightarrow$  '*a*)  $\Rightarrow$  ('*a list*  $\Rightarrow$  '*a list*) **where**  
*mapAt* [] *f* *as* = *as*  
| *mapAt* (*n#ns*) *f* *as* =  
(if *n* < |*as*| then *mapAt* *ns* *f* (*as*[*n*:= *f* (*as*!*n*)])  
else *mapAt* *ns* *f* *as*)

**lemma** *length-mapAt*[*simp*]:  $\bigwedge$ *xs*. *length*(*mapAt* *vs* *f* *xs*) = *length* *xs*  
 $\langle$ *proof* $\rangle$

**lemma** *length-replace1*[*simp*]: *length*(*replace* *x* [*y*] *xs*) = *length* *xs*  
 $\langle$ *proof* $\rangle$

**lemma** *replace-id*[*simp*]: *replace* *x* [*x*] *xs* = *xs*  
 $\langle$ *proof* $\rangle$

**lemma** *len-replace-ge-same*:  
*length* *ys*  $\geq$  1  $\implies$  *length*(*replace* *x* *ys* *xs*)  $\geq$  *length* *xs*  
 $\langle$ *proof* $\rangle$

**lemma** *len-replace-ge*[*simp*]:  
[[ *length* *ys*  $\geq$  1; *length* *xs*  $\geq$  *length* *zs* ]]  $\implies$   
*length*(*replace* *x* *ys* *xs*)  $\geq$  *length* *zs*  
 $\langle$ *proof* $\rangle$

**lemma** *replace-append*[*simp*]:

$replace\ x\ ys\ (as\ @\ bs) =$   
 $(if\ x \in set\ as\ then\ replace\ x\ ys\ as\ @\ bs\ else\ as\ @\ replace\ x\ ys\ bs)$   
 <proof>

**lemma** *distinct-set-replace*:  $distinct\ xs \implies$   
 $set\ (replace\ x\ ys\ xs) =$   
 $(if\ x \in set\ xs\ then\ (set\ xs - \{x\}) \cup set\ ys\ else\ set\ xs)$   
 <proof>

**lemma** *replace1*:  
 $f \in set\ (replace\ f'\ fs\ ls) \implies f \notin set\ ls \implies f \in set\ fs$   
 <proof>

**lemma** *replace2*:  
 $f' \notin set\ ls \implies replace\ f'\ fs\ ls = ls$   
 <proof>

**lemma** *replace3[intro]*:  
 $f' \in set\ ls \implies f \in set\ fs \implies f \in set\ (replace\ f'\ fs\ ls)$   
 <proof>

**lemma** *replace4*:  
 $f \in set\ ls \implies oldF \neq f \implies f \in set\ (replace\ oldF\ fs\ ls)$   
 <proof>

**lemma** *replace5*:  $f \in set\ (replace\ oldF\ newfs\ fs) \implies f \in set\ fs \vee f \in set\ newfs$   
 <proof>

**lemma** *replace6*:  $distinct\ oldfs \implies x \in set\ (replace\ oldF\ newfs\ oldfs) =$   
 $((x \neq oldF \vee oldF \in set\ newfs) \wedge ((oldF \in set\ oldfs \wedge x \in set\ newfs) \vee x \in set\ oldfs))$   
 <proof>

**lemma** *distinct-replace*:  
 $distinct\ fs \implies distinct\ newFs \implies set\ fs \cap set\ newFs \subseteq \{oldF\} \implies$   
 $distinct\ (replace\ oldF\ newFs\ fs)$   
 <proof>

**lemma** *replace-replace[simp]*:  $oldf \notin set\ newfs \implies distinct\ xs \implies$   
 $replace\ oldf\ newfs\ (replace\ oldf\ newfs\ xs) = replace\ oldf\ newfs\ xs$   
 <proof>

**lemma** *replace-distinct*:  $distinct\ fs \implies distinct\ newfs \implies oldf \in set\ fs \implies set\ newfs \cap set\ fs \subseteq \{oldf\} \implies$   
 $distinct\ (replace\ oldf\ newfs\ fs)$   
 <proof>



**lemma** *filter-replace2*:

$\llbracket \neg P x; \forall y \in \text{set } ys. \neg P y \rrbracket \implies$   
 $\text{filter } P (\text{replace } x \text{ } ys \text{ } xs) = \text{filter } P xs$   
*<proof>*

**lemma** *length-filter-replace1*:

$\llbracket x \in \text{set } xs; \neg P x \rrbracket \implies$   
 $\text{length}(\text{filter } P (\text{replace } x \text{ } ys \text{ } xs)) =$   
 $\text{length}(\text{filter } P xs) + \text{length}(\text{filter } P ys)$   
*<proof>*

**lemma** *length-filter-replace2*:

$\llbracket x \in \text{set } xs; P x \rrbracket \implies$   
 $\text{length}(\text{filter } P (\text{replace } x \text{ } ys \text{ } xs)) =$   
 $\text{length}(\text{filter } P xs) + \text{length}(\text{filter } P ys) - 1$   
*<proof>*

### 1.2.7 *distinct*

**lemma** *dist-at1*:  $\bigwedge c \text{ vs. } \text{distinct } vs \implies vs = a @ r \# b \implies vs = c @ r \# d \implies$   
 $a = c$   
*<proof>*

**lemma** *dist-at*:  $\text{distinct } vs \implies vs = a @ r \# b \implies vs = c @ r \# d \implies a = c$   
 $\wedge b = d$   
*<proof>*

**lemma** *dist-at2*:  $\text{distinct } vs \implies vs = a @ r \# b \implies vs = c @ r \# d \implies b = d$   
*<proof>*

**lemma** *distinct-split1*:  $\text{distinct } xs \implies xs = y @ [r] @ z \implies r \notin \text{set } y$   
*<proof>*

**lemma** *distinct-split2*:  $\text{distinct } xs \implies xs = y @ [r] @ z \implies r \notin \text{set } z$  *<proof>*

**lemma** *distinct-hd-not-cons*:  $\text{distinct } vs \implies \exists as \text{ } bs. vs = as @ x \# hd \text{ } vs \# bs$   
 $\implies \text{False}$   
*<proof>*

### 1.2.8 *Misc*

**lemma** *drop-last-in*:  $\bigwedge n. n < \text{length } ls \implies \text{last } ls \in \text{set } (\text{drop } n \text{ } ls)$   
*<proof>*

**lemma** *nth-last-Suc-n*:  $\text{distinct } ls \implies n < \text{length } ls \implies \text{last } ls = ls ! n \implies \text{Suc}$   
 $n = \text{length } ls$   
*<proof>*

### 1.2.9 rotate

**lemma** *plus-length1[simp]*:  $\text{rotate } (k + (\text{length } ls)) \text{ } ls = \text{rotate } k \text{ } ls$   
{proof}

**lemma** *plus-length2[simp]*:  $\text{rotate } ((\text{length } ls) + k) \text{ } ls = \text{rotate } k \text{ } ls$   
{proof}

**lemma** *rotate-minus1*:  $n > 0 \implies m > 0 \implies$   
 $\text{rotate } n \text{ } ls = \text{rotate } m \text{ } ms \implies \text{rotate } (n - 1) \text{ } ls = \text{rotate } (m - 1) \text{ } ms$   
{proof}

**lemma** *rotate-minus1'*:  $n > 0 \implies \text{rotate } n \text{ } ls = ms \implies$   
 $\text{rotate } (n - 1) \text{ } ls = \text{rotate } (\text{length } ms - 1) \text{ } ms$   
{proof}

**lemma** *rotate-inv1*:  $\bigwedge ms. n < \text{length } ls \implies \text{rotate } n \text{ } ls = ms \implies$   
 $ls = \text{rotate } ((\text{length } ls) - n) \text{ } ms$   
{proof}

**lemma** *rotate-conv-mod'[simp]*:  $\text{rotate } (n \bmod \text{length } ls) \text{ } ls = \text{rotate } n \text{ } ls$   
{proof}

**lemma** *rotate-inv2*:  $\text{rotate } n \text{ } ls = ms \implies$   
 $ls = \text{rotate } ((\text{length } ls) - (n \bmod \text{length } ls)) \text{ } ms$   
{proof}

**lemma** *rotate-id[simp]*:  $\text{rotate } ((\text{length } ls) - (n \bmod \text{length } ls)) (\text{rotate } n \text{ } ls) = ls$   
{proof}

**lemma** *nth-rotate1-Suc*:  $Suc \ n < \text{length } ls \implies ls!(Suc \ n) = (\text{rotate1 } ls)!n$   
{proof}

**lemma** *nth-rotate1-0*:  $ls!0 = (\text{rotate1 } ls)!(\text{length } ls - 1)$  {proof}

**lemma** *nth-rotate1*:  $0 < \text{length } ls \implies ls!((Suc \ n) \bmod (\text{length } ls)) = (\text{rotate1 } ls)!(n \bmod (\text{length } ls))$   
{proof}

**lemma** *rotate-Suc2[simp]*:  $\text{rotate } n (\text{rotate1 } xs) = \text{rotate } (Suc \ n) \text{ } xs$   
{proof}

**lemma** *nth-rotate*:  $\bigwedge ls. 0 < \text{length } ls \implies ls!((n+m) \bmod (\text{length } ls)) = (\text{rotate } m \text{ } ls)!(n \bmod (\text{length } ls))$   
{proof}

### 1.3 splitAt

**primrec** *splitAtRec* :: 'a  $\Rightarrow$  'a list  $\Rightarrow$  'a list  $\Rightarrow$  'a list  $\times$  'a list **where**  
*splitAtRec* c bs [] = (bs, [])

|  $splitAtRec\ c\ bs\ (a\#as) = (if\ a = c\ then\ (bs,\ as)$   
 $else\ splitAtRec\ c\ (bs@[a])\ as)$

**definition**  $splitAt :: 'a \Rightarrow 'a\ list \Rightarrow 'a\ list \times 'a\ list$  **where**  
 $splitAt\ c\ as \equiv splitAtRec\ c\ []\ as$

### 1.3.1 $splitAtRec$

**lemma**  $splitAtRec\ conv: \bigwedge bs.$   
 $splitAtRec\ x\ bs\ xs =$   
 $(bs\ @\ takeWhile\ (\lambda y. y \neq x)\ xs,\ tl(dropWhile\ (\lambda y. y \neq x)\ xs))$   
 $\langle proof \rangle$

**lemma**  $splitAtRec\ distinct\ fst: \bigwedge s. distinct\ vs \implies distinct\ s \implies (set\ s) \cap (set\ vs) = \{\}$   
 $\implies distinct\ (fst\ (splitAtRec\ ram1\ s\ vs))$   
 $\langle proof \rangle$

**lemma**  $splitAtRec\ distinct\ snd: \bigwedge s. distinct\ vs \implies distinct\ s \implies (set\ s) \cap (set\ vs) = \{\}$   
 $\implies distinct\ (snd\ (splitAtRec\ ram1\ s\ vs))$   
 $\langle proof \rangle$

**lemma**  $splitAtRec\ ram:$   
 $\bigwedge us\ a\ b. ram \in set\ vs \implies (a,\ b) = splitAtRec\ ram\ us\ vs \implies$   
 $us\ @\ vs = a\ @\ [ram]\ @\ b$   
 $\langle proof \rangle$

**lemma**  $splitAtRec\ notRam:$   
 $\bigwedge us. ram \notin set\ vs \implies splitAtRec\ ram\ us\ vs = (us\ @\ vs,\ [])$   
 $\langle proof \rangle$

**lemma**  $splitAtRec\ distinct: \bigwedge s. distinct\ vs \implies$   
 $distinct\ s \implies (set\ s) \cap (set\ vs) = \{\} \implies$   
 $set\ (fst\ (splitAtRec\ ram\ s\ vs)) \cap set\ (snd\ (splitAtRec\ ram\ s\ vs)) = \{\}$   
 $\langle proof \rangle$

### 1.3.2 $splitAt$

**lemma**  $splitAt\ conv:$   
 $splitAt\ x\ xs = (takeWhile\ (\lambda y. y \neq x)\ xs,\ tl(dropWhile\ (\lambda y. y \neq x)\ xs))$   
 $\langle proof \rangle$

**lemma**  $splitAt\ no\ ram[simp]:$   
 $ram \notin set\ vs \implies splitAt\ ram\ vs = (vs,\ [])$   
 $\langle proof \rangle$

**lemma**  $splitAt\ split:$   
 $ram \in set\ vs \implies (a,\ b) = splitAt\ ram\ vs \implies vs = a\ @\ ram\ \# b$   
 $\langle proof \rangle$

**lemma**  $splitAt\ ram:$

$ram \in set\ vs \implies vs = fst\ (splitAt\ ram\ vs) @ ram \# snd\ (splitAt\ ram\ vs)$   
 ⟨proof⟩

**lemma** *fst-splitAt-last*:

$\llbracket xs \neq []; distinct\ xs \rrbracket \implies fst\ (splitAt\ (last\ xs)\ xs) = butlast\ xs$   
 ⟨proof⟩

### 1.3.3 Sets

**lemma** *splitAtRec-union*:

$\bigwedge a\ b\ s. (a,b) = splitAtRec\ ram\ s\ vs \implies (set\ a \cup set\ b) - \{ram\} = (set\ vs \cup set\ s) - \{ram\}$   
 ⟨proof⟩

**lemma** *splitAt-subset-ab*:

$(a,b) = splitAt\ ram\ vs \implies set\ a \subseteq set\ vs \wedge set\ b \subseteq set\ vs$   
 ⟨proof⟩

**lemma** *splitAt-in-fst[dest]*:  $v \in set\ (fst\ (splitAt\ ram\ vs)) \implies v \in set\ vs$   
 ⟨proof⟩

**lemma** *splitAt-not1*:

$v \notin set\ vs \implies v \notin set\ (fst\ (splitAt\ ram\ vs))$  ⟨proof⟩

**lemma** *splitAt-in-snd[dest]*:  $v \in set\ (snd\ (splitAt\ ram\ vs)) \implies v \in set\ vs$   
 ⟨proof⟩

### 1.3.4 Distinctness

**lemma** *splitAt-distinct-ab-aux*:

$distinct\ vs \implies (a,b) = splitAt\ ram\ vs \implies distinct\ a \wedge distinct\ b$   
 ⟨proof⟩

**lemma** *splitAt-distinct-fst-aux[intro]*:

$distinct\ vs \implies distinct\ (fst\ (splitAt\ ram\ vs))$   
 ⟨proof⟩

**lemma** *splitAt-distinct-snd-aux[intro]*:

$distinct\ vs \implies distinct\ (snd\ (splitAt\ ram\ vs))$   
 ⟨proof⟩

**lemma** *splitAt-distinct-ab*:

$distinct\ vs \implies (a,b) = splitAt\ ram\ vs \implies set\ a \cap set\ b = \{\}$   
 ⟨proof⟩

**lemma** *splitAt-distinct-fst-snd*:

$distinct\ vs \implies set\ (fst\ (splitAt\ ram\ vs)) \cap set\ (snd\ (splitAt\ ram\ vs)) = \{\}$   
 ⟨proof⟩

**lemma** *splitAt-distinct-ram-fst[intro]*:

$distinct\ vs \implies ram \notin set\ (fst\ (splitAt\ ram\ vs))$   
 $\langle proof \rangle$

**lemma** *splitAt-distinct-ram-snd*[intro]:  
 $distinct\ vs \implies ram \notin set\ (snd\ (splitAt\ ram\ vs))$   
 $\langle proof \rangle$

**lemma** *splitAt-1*[simp]:  
 $splitAt\ ram\ [] = ([], [])$   $\langle proof \rangle$

**lemma** *splitAt-2*:  
 $v \in set\ vs \implies (a,b) = splitAt\ ram\ vs \implies v \in set\ a \vee v \in set\ b \vee v = ram$   
 $\langle proof \rangle$

**lemma** *splitAt-distinct-fst*:  $distinct\ vs \implies distinct\ (fst\ (splitAt\ ram1\ vs))$   
 $\langle proof \rangle$

**lemma** *splitAt-distinct-a*:  $distinct\ vs \implies (a,b) = splitAt\ ram\ vs \implies distinct\ a$   
 $\langle proof \rangle$

**lemma** *splitAt-distinct-snd*:  $distinct\ vs \implies distinct\ (snd\ (splitAt\ ram1\ vs))$   
 $\langle proof \rangle$

**lemma** *splitAt-distinct-b*:  $distinct\ vs \implies (a,b) = splitAt\ ram\ vs \implies distinct\ b$   
 $\langle proof \rangle$

**lemma** *splitAt-distinct*:  $distinct\ vs \implies set\ (fst\ (splitAt\ ram\ vs)) \cap set\ (snd\ (splitAt\ ram\ vs)) = \{\}$   
 $\langle proof \rangle$

**lemma** *splitAt-subset*:  $(a,b) = splitAt\ ram\ vs \implies (set\ a \subseteq set\ vs) \wedge (set\ b \subseteq set\ vs)$   
 $\langle proof \rangle$

### 1.3.5 *splitAt* composition

**lemma** *set-help*:  $v \in set\ (as\ @\ bs) \implies v \in set\ as \vee v \in set\ bs$   $\langle proof \rangle$

**lemma** *splitAt-elements*:  $ram1 \in set\ vs \implies ram2 \in set\ vs \implies ram2 \in set\ (fst\ (splitAt\ ram1\ vs)) \vee ram2 \in set\ [ram1] \vee ram2 \in set\ (snd\ (splitAt\ ram1\ vs))$   
 $\langle proof \rangle$

**lemma** *splitAt-ram3*:  $ram2 \notin set\ (fst\ (splitAt\ ram1\ vs)) \implies$   
 $ram1 \in set\ vs \implies ram2 \in set\ vs \implies ram1 \neq ram2 \implies$   
 $ram2 \in set\ (snd\ (splitAt\ ram1\ vs))$   $\langle proof \rangle$

**lemma** *splitAt-dist-ram*:  $distinct\ vs \implies$   
 $vs = a\ @\ ram\ \# \ b \implies (a,b) = splitAt\ ram\ vs$

*<proof>*

**lemma** *distinct-unique1*:  $distinct\ vs \implies ram \in set\ vs \implies \exists!s. vs = (fst\ s) @ ram \# (snd\ s)$   
*<proof>*

**lemma** *splitAt-dist-ram2*:  $distinct\ vs \implies vs = a @ ram1 \# b @ ram2 \# c \implies (a @ ram1 \# b, c) = splitAt\ ram2\ vs$   
*<proof>*

**lemma** *splitAt-dist-ram20*:  $distinct\ vs \implies vs = a @ ram1 \# b @ ram2 \# c \implies c = snd\ (splitAt\ ram2\ vs)$   
*<proof>*

**lemma** *splitAt-dist-ram21*:  $distinct\ vs \implies vs = a @ ram1 \# b @ ram2 \# c \implies (a, b) = splitAt\ ram1\ (fst\ (splitAt\ ram2\ vs))$   
*<proof>*

**lemma** *splitAt-dist-ram22*:  $distinct\ vs \implies vs = a @ ram1 \# b @ ram2 \# c \implies (c, []) = splitAt\ ram1\ (snd\ (splitAt\ ram2\ vs))$   
*<proof>*

**lemma** *splitAt-dist-ram1*:  $distinct\ vs \implies vs = a @ ram1 \# b @ ram2 \# c \implies (a, b @ ram2 \# c) = splitAt\ ram1\ vs$   
*<proof>*

**lemma** *splitAt-dist-ram10*:  $distinct\ vs \implies vs = a @ ram1 \# b @ ram2 \# c \implies a = fst\ (splitAt\ ram1\ vs)$   
*<proof>*

**lemma** *splitAt-dist-ram11*:  $distinct\ vs \implies vs = a @ ram1 \# b @ ram2 \# c \implies (a, []) = splitAt\ ram2\ (fst\ (splitAt\ ram1\ vs))$   
*<proof>*

**lemma** *splitAt-dist-ram12*:  $distinct\ vs \implies vs = a @ ram1 \# b @ ram2 \# c \implies (b, c) = splitAt\ ram2\ (snd\ (splitAt\ ram1\ vs))$   
*<proof>*

**lemma** *splitAt-dist-ram-all*:  
 $distinct\ vs \implies vs = a @ ram1 \# b @ ram2 \# c$   
 $\implies (a, b) = splitAt\ ram1\ (fst\ (splitAt\ ram2\ vs))$   
 $\wedge (c, []) = splitAt\ ram1\ (snd\ (splitAt\ ram2\ vs))$   
 $\wedge (a, []) = splitAt\ ram2\ (fst\ (splitAt\ ram1\ vs))$   
 $\wedge (b, c) = splitAt\ ram2\ (snd\ (splitAt\ ram1\ vs))$   
 $\wedge c = snd\ (splitAt\ ram2\ vs)$   
 $\wedge a = fst\ (splitAt\ ram1\ vs)$   
*<proof>*

### 1.3.6 Mixed

**lemma** *fst-splitAt-rev*:

$distinct\ xs \implies x \in set\ xs \implies$   
 $fst(splitAt\ x\ (rev\ xs)) = rev(snd(splitAt\ x\ xs))$   
 $\langle proof \rangle$

**lemma** *snd-splitAt-rev*:

$distinct\ xs \implies x \in set\ xs \implies$   
 $snd(splitAt\ x\ (rev\ xs)) = rev(fst(splitAt\ x\ xs))$   
 $\langle proof \rangle$

**lemma** *splitAt-take[simp]*:  $distinct\ ls \implies i < length\ ls \implies fst\ (splitAt\ (ls!i)\ ls)$   
 $= take\ i\ ls$

$\langle proof \rangle$

**lemma** *splitAt-drop[simp]*:  $distinct\ ls \implies i < length\ ls \implies snd\ (splitAt\ (ls!i)\ ls)$   
 $= drop\ (Suc\ i)\ ls$

$\langle proof \rangle$

**lemma** *fst-splitAt-upt*:

$j \leq i \implies i < k \implies fst(splitAt\ i\ [j..<k]) = [j..<i]$   
 $\langle proof \rangle$

**lemma** *snd-splitAt-upt*:

$j \leq i \implies i < k \implies snd(splitAt\ i\ [j..<k]) = [i+1..<k]$   
 $\langle proof \rangle$

**lemma** *local-help1*:  $\bigwedge a\ vs.\ vs = c @ r \# d \implies vs = a @ r \# b \implies r \notin set\ a$   
 $\implies r \notin set\ b \implies a = c$

$\langle proof \rangle$

**lemma** *local-help*:  $vs = a @ r \# b \implies vs = c @ r \# d \implies r \notin set\ a \implies r \notin$   
 $set\ b \implies a = c \wedge b = d$

$\langle proof \rangle$

**lemma** *local-help'*:  $a @ r \# b = c @ r \# d \implies r \notin set\ a \implies r \notin set\ b \implies a =$   
 $c \wedge b = d$

$\langle proof \rangle$

**lemma** *splitAt-simp1*:  $ram \notin set\ a \implies ram \notin set\ b \implies fst\ (splitAt\ ram\ (a @ ram$   
 $\# b)) = a$

$\langle proof \rangle$

**lemma** *help'''-in*:  $\bigwedge xs.\ ram \in set\ b \implies fst\ (splitAtRec\ ram\ xs\ b) = xs @ fst$   
 $(splitAtRec\ ram\ []\ b)$

$\langle proof \rangle$

**lemma** *help'''-notin*:  $\bigwedge xs. ram \notin set\ b \implies fst\ (splitAtRec\ ram\ xs\ b) = xs\ @\ fst\ (splitAtRec\ ram\ []\ b)$   
 ⟨proof⟩

**lemma** *help'''*:  $fst\ (splitAtRec\ ram\ xs\ b) = xs\ @\ fst\ (splitAtRec\ ram\ []\ b)$   
 ⟨proof⟩

**lemma** *splitAt-simpA[simp]*:  $fst\ (splitAt\ ram\ (ram\ \# \ b)) = []$  ⟨proof⟩

**lemma** *splitAt-simpB[simp]*:  $ram \neq a \implies fst\ (splitAt\ ram\ (a\ \# \ b)) = a\ \# \ fst\ (splitAt\ ram\ b)$  ⟨proof⟩

**lemma** *splitAt-simpB'[simp]*:  $a \neq ram \implies fst\ (splitAt\ ram\ (a\ \# \ b)) = a\ \# \ fst\ (splitAt\ ram\ b)$  ⟨proof⟩

**lemma** *splitAt-simpC[simp]*:  $ram \notin set\ a \implies fst\ (splitAt\ ram\ (a\ @ \ b)) = a\ @ \ fst\ (splitAt\ ram\ b)$   
 ⟨proof⟩

**lemma** *help''''*:  $\bigwedge xs\ ys. snd\ (splitAtRec\ ram\ xs\ b) = snd\ (splitAtRec\ ram\ ys\ b)$   
 ⟨proof⟩

**lemma** *splitAt-simpD[simp]*:  $\bigwedge a. ram \neq a \implies snd\ (splitAt\ ram\ (a\ \# \ b)) = snd\ (splitAt\ ram\ b)$  ⟨proof⟩

**lemma** *splitAt-simpD'[simp]*:  $\bigwedge a. a \neq ram \implies snd\ (splitAt\ ram\ (a\ \# \ b)) = snd\ (splitAt\ ram\ b)$  ⟨proof⟩

**lemma** *splitAt-simpE[simp]*:  $snd\ (splitAt\ ram\ (ram\ \# \ b)) = b$  ⟨proof⟩

**lemma** *splitAt-simpF[simp]*:  $ram \notin set\ a \implies snd\ (splitAt\ ram\ (a\ @ \ b)) = snd\ (splitAt\ ram\ b)$   
 ⟨proof⟩

**lemma** *splitAt-rotate-pair-conv*:

$\bigwedge xs. [ \text{distinct } xs; x \in set\ xs ]$   
 $\implies snd\ (splitAt\ x\ (rotate\ n\ xs))\ @ \ fst\ (splitAt\ x\ (rotate\ n\ xs)) =$   
 $snd\ (splitAt\ x\ xs)\ @ \ fst\ (splitAt\ x\ xs)$

⟨proof⟩

#### 1.4 between

**definition** *between* :: 'a list  $\Rightarrow$  'a  $\Rightarrow$  'a  $\Rightarrow$  'a list **where**

*between* vs ram<sub>1</sub> ram<sub>2</sub>  $\equiv$

let (pre<sub>1</sub>, post<sub>1</sub>) = splitAt ram<sub>1</sub> vs in

if ram<sub>2</sub>  $\in$  set post<sub>1</sub>

then let (pre<sub>2</sub>, post<sub>2</sub>) = splitAt ram<sub>2</sub> post<sub>1</sub> in pre<sub>2</sub>

else let (pre<sub>2</sub>, post<sub>2</sub>) = splitAt ram<sub>2</sub> pre<sub>1</sub> in post<sub>1</sub> @ pre<sub>2</sub>

**lemma** *inbetween-inset*:

$x \in set(\text{between } xs\ a\ b) \implies x \in set\ xs$

⟨proof⟩



**lemma** *notinset-notinbetween*:

$x \notin \text{set } xs \implies x \notin \text{set}(\text{between } xs \ a \ b)$   
*<proof>*

**lemma** *set-between-id*:

$\text{distinct } xs \implies x \in \text{set } xs \implies$   
 $\text{set}(\text{between } xs \ x \ x) = \text{set } xs - \{x\}$   
*<proof>*

**lemma** *split-between*:

$\llbracket \text{distinct } vs; r \in \text{set } vs; v \in \text{set } vs; u \in \text{set}(\text{between } vs \ r \ v) \rrbracket \implies$   
 $\text{between } vs \ r \ v =$   
 $(\text{if } r=u \ \text{then } \llbracket \ \text{else } \text{between } vs \ r \ u \ @ \ [u] \rrbracket \ @ \ \text{between } vs \ u \ v$   
*<proof>*

## 1.5 Tables

**type-synonym** (*'a*, *'b*) *table* = (*'a* × *'b*) *list*

**definition** *isTable* :: (*'a* ⇒ *'b*) ⇒ *'a list* ⇒ (*'a*, *'b*) *table* ⇒ *bool* **where**  
 $\text{isTable } f \ \text{vs } t \equiv \forall p. p \in \text{set } t \longrightarrow \text{snd } p = f \ (\text{fst } p) \wedge \text{fst } p \in \text{set } \text{vs}$

**lemma** *isTable-eq*:  $\text{isTable } E \ \text{vs } ((a,b)\#ps) \implies b = E \ a$   
*<proof>*

**lemma** *isTable-subset*:

$\text{set } qs \subseteq \text{set } ps \implies \text{isTable } E \ \text{vs } ps \implies \text{isTable } E \ \text{vs } qs$   
*<proof>*

**lemma** *isTable-Cons*:  $\text{isTable } E \ \text{vs } ((a,b)\#ps) \implies \text{isTable } E \ \text{vs } ps$   
*<proof>*

**definition** *removeKey* :: *'a* ⇒ (*'a* × *'b*) *list* ⇒ (*'a* × *'b*) *list* **where**  
 $\text{removeKey } a \ ps \equiv [p \leftarrow ps. a \neq \text{fst } p]$

**primrec** *removeKeyList* :: *'a list* ⇒ (*'a* × *'b*) *list* ⇒ (*'a* × *'b*) *list* **where**  
 $\text{removeKeyList } [] \ ps = ps$   
 $|\ \text{removeKeyList } (w\#ws) \ ps = \text{removeKey } w \ (\text{removeKeyList } ws \ ps)$

**lemma** *removeKey-subset[simp]*:  $\text{set } (\text{removeKey } a \ ps) \subseteq \text{set } ps$   
*<proof>*

**lemma** *length-removeKey[simp]*:  $|\text{removeKey } w \ ps| \leq |ps|$   
*<proof>*

**lemma** *length-removeKeyList*:  
 $length (removeKeyList ws ps) \leq length ps$  (**is** ?P ws)  
⟨proof⟩

**lemma** *removeKeyList-subset[simp]*:  $set (removeKeyList ws ps) \subseteq set ps$   
⟨proof⟩

**lemma** *notin-removeKey1*:  $(a, b) \notin set (removeKey a ps)$   
⟨proof⟩

**lemma** *removeKeyList-eq*:  
 $removeKeyList as ps = [p \leftarrow ps. \forall a \in set as. a \neq fst p]$   
⟨proof⟩

**lemma** *removeKey-empty[simp]*:  $removeKey a [] = []$   
⟨proof⟩

**lemma** *removeKeyList-empty[simp]*:  $removeKeyList ps [] = []$   
⟨proof⟩

**lemma** *removeKeyList-cons[simp]*:  
 $removeKeyList ws (p\#ps)$   
 $= (if\ fst\ p \in\ set\ ws\ then\ removeKeyList\ ws\ ps\ else\ p\#(removeKeyList\ ws\ ps))$   
⟨proof⟩

**end**

**theory** *Quasi-Order*

**imports** *Main*

**begin**

**locale** *quasi-order* =

**fixes** *gle* :: 'a  $\Rightarrow$  'a  $\Rightarrow$  bool (**infix**  $\preceq$  60)

**assumes** *gle-refl[iff]*:  $x \preceq x$

**and** *gle-trans*:  $x \preceq y \Longrightarrow y \preceq z \Longrightarrow x \preceq z$

**begin**

**definition** *in-ql* :: 'a  $\Rightarrow$  'a set  $\Rightarrow$  bool (**infix**  $\in_{\preceq}$  60) **where**  
 $x \in_{\preceq} M \equiv \exists y \in M. x \preceq y$

**definition** *subsetq-ql* :: 'a set  $\Rightarrow$  'a set  $\Rightarrow$  bool (**infix**  $\subseteq_{\preceq}$  60) **where**  
 $M \subseteq_{\preceq} N \equiv \forall x \in M. x \in_{\preceq} N$

**definition** *seteq-ql* :: 'a set  $\Rightarrow$  'a set  $\Rightarrow$  bool (**infix**  $=_{\preceq}$  60) **where**  
 $M =_{\preceq} N \equiv M \subseteq_{\preceq} N \wedge N \subseteq_{\preceq} M$

**lemmas** *defs* = *in-ql-def subseteq-ql-def seteq-ql-def*

**lemma** *subsetq-ql-refl[simp]*:  $M \subseteq_{\preceq} M$   
⟨proof⟩

**lemma** *subsetq-ql-trans*:  $A \subseteq_{\preceq} B \Longrightarrow B \subseteq_{\preceq} C \Longrightarrow A \subseteq_{\preceq} C$

*<proof>*

**lemma** *empty-subseteq-qle[simp]*:  $\{\} \subseteq_{\preceq} A$   
*<proof>*

**lemma** *subseteq-qleI2*:  $(\bigwedge x. x \in M \implies \exists y \in N. x \preceq y) \implies M \subseteq_{\preceq} N$   
*<proof>*

**lemma** *subseteq-qleD2*:  $M \subseteq_{\preceq} N \implies x \in M \implies \exists y \in N. x \preceq y$   
*<proof>*

**lemma** *seteq-qle-refl[iff]*:  $A =_{\preceq} A$   
*<proof>*

**lemma** *seteq-qle-trans*:  $A =_{\preceq} B \implies B =_{\preceq} C \implies A =_{\preceq} C$   
*<proof>*

**end**

**end**

## 2 Isomorphisms Between Plane Graphs

**theory** *PlaneGraphIso*  
**imports** *Main Quasi-Order*  
**begin**

**lemma** *image-image-id-if[simp]*:  $(\bigwedge x. f(f x) = x) \implies f \circ f \circ M = M$   
*<proof>*

**declare** *not-None-eq [iff] not-Some-eq [iff]*

The symbols  $\cong$  and  $\simeq$  are overloaded. They denote congruence and isomorphism on arbitrary types. On lists (representing faces of graphs),  $\cong$  means congruence modulo rotation;  $\simeq$  is currently unused. On graphs,  $\simeq$  means isomorphism and is a weaker version of  $\cong$  (proper isomorphism):  $\simeq$  also allows to reverse the orientation of all faces.

**consts**  
*pr-isomorphic* ::  $'a \Rightarrow 'a \Rightarrow \text{bool}$  (**infix**  $\cong$  60)

**definition** *Iso* ::  $('a \text{ list} * 'a \text{ list}) \text{ set}$  ( $\{\cong\}$ ) **where**  
 $\{\cong\} \equiv \{(F_1, F_2). F_1 \cong F_2\}$

**lemma** *[iff]*:  $((x,y) \in \{\cong\}) = x \cong y$   
*<proof>*

A plane graph is a set or list (for executability) of faces (hence  $Fgraph$  and  $fgraph$ ) and a face is a list of nodes:

**type-synonym**  $'a Fgraph = 'a list set$   
**type-synonym**  $'a fgraph = 'a list list$

## 2.1 Equivalence of faces

Two faces are equivalent modulo rotation:

**overloading**  $congs \equiv pr\text{-isomorphic} :: 'a list \Rightarrow 'a list \Rightarrow bool$   
**begin**  
**definition**  $F_1 \cong (F_2 :: 'a list) \equiv \exists n. F_2 = rotate\ n\ F_1$   
**end**

**lemma**  $congs\text{-refl}[iff]: (xs :: 'a list) \cong xs$   
 $\langle proof \rangle$

**lemma**  $congs\text{-sym}: \text{assumes } A: (xs :: 'a list) \cong ys \text{ shows } ys \cong xs$   
 $\langle proof \rangle$

**lemma**  $congs\text{-trans}: (xs :: 'a list) \cong ys \Longrightarrow ys \cong zs \Longrightarrow xs \cong zs$   
 $\langle proof \rangle$

**lemma**  $equiv\text{-EqF}: equiv\ (UNIV :: 'a list set) \{\cong\}$   
 $\langle proof \rangle$

**lemma**  $congs\text{-distinct}: F_1 \cong F_2 \Longrightarrow distinct\ F_2 = distinct\ F_1$   
 $\langle proof \rangle$

**lemma**  $congs\text{-length}: F_1 \cong F_2 \Longrightarrow length\ F_2 = length\ F_1$   
 $\langle proof \rangle$

**lemma**  $congs\text{-pres-nodes}: F_1 \cong F_2 \Longrightarrow set\ F_1 = set\ F_2$   
 $\langle proof \rangle$

**lemma**  $congs\text{-map}: F_1 \cong F_2 \Longrightarrow map\ f\ F_1 \cong map\ f\ F_2$   
 $\langle proof \rangle$

**lemma**  $congs\text{-map-eq-iff}: inj\text{-on } f\ (set\ xs \cup set\ ys) \Longrightarrow (map\ f\ xs \cong map\ f\ ys) = (xs \cong ys)$   
 $\langle proof \rangle$

**lemma**  $list\text{-cong-rev-iff}[simp]: (rev\ xs \cong rev\ ys) = (xs \cong ys)$   
 $\langle proof \rangle$

**lemma** *singleton-list-cong-eq-iff*[simp]:  
 $(\{xs::'a\ list\} // \{\cong\} = \{ys\} // \{\cong\}) = (xs \cong ys)$   
 <proof>

## 2.2 Homomorphism and isomorphism

**definition** *is-pr-Hom* :: ('a  $\Rightarrow$  'b)  $\Rightarrow$  'a Fgraph  $\Rightarrow$  'b Fgraph  $\Rightarrow$  bool **where**  
*is-pr-Hom*  $\varphi$   $Fs_1$   $Fs_2 \equiv (\text{map } \varphi \text{ ` } Fs_1) // \{\cong\} = Fs_2 // \{\cong\}$

**definition** *is-pr-Iso* :: ('a  $\Rightarrow$  'b)  $\Rightarrow$  'a Fgraph  $\Rightarrow$  'b Fgraph  $\Rightarrow$  bool **where**  
*is-pr-Iso*  $\varphi$   $Fs_1$   $Fs_2 \equiv \text{is-pr-Hom } \varphi$   $Fs_1$   $Fs_2 \wedge \text{inj-on } \varphi (\bigcup F \in Fs_1. \text{set } F)$

**definition** *is-pr-iso* :: ('a  $\Rightarrow$  'b)  $\Rightarrow$  'a fgraph  $\Rightarrow$  'b fgraph  $\Rightarrow$  bool **where**  
*is-pr-iso*  $\varphi$   $Fs_1$   $Fs_2 \equiv \text{is-pr-Iso } \varphi$  (set  $Fs_1$ ) (set  $Fs_2$ )

Homomorphisms preserve the set of nodes.

**lemma** *UN-subset-iff*:  $((\bigcup i \in I. f\ i) \subseteq B) = (\forall i \in I. f\ i \subseteq B)$   
 <proof>

**declare** *Image-Collect-case-prod*[simp del]

**lemma** *pr-Hom-pres-face-nodes*:  
*is-pr-Hom*  $\varphi$   $Fs_1$   $Fs_2 \implies (\bigcup F \in Fs_1. \{\varphi \text{ ` } (\text{set } F)\}) = (\bigcup F \in Fs_2. \{\text{set } F\})$   
 <proof>

**lemma** *pr-Hom-pres-nodes*:  
**assumes** *is-pr-Hom*  $\varphi$   $Fs_1$   $Fs_2$   
**shows**  $\varphi \text{ ` } (\bigcup F \in Fs_1. \text{set } F) = (\bigcup F \in Fs_2. \text{set } F)$   
 <proof>

Therefore isomorphisms preserve cardinality of node set.

**lemma** *pr-Iso-same-no-nodes*:  
 $\llbracket \text{is-pr-Iso } \varphi$   $Fs_1$   $Fs_2; \text{finite } Fs_1 \rrbracket$   
 $\implies \text{card}(\bigcup F \in Fs_1. \text{set } F) = \text{card}(\bigcup F \in Fs_2. \text{set } F)$   
 <proof>

**lemma** *pr-iso-same-no-nodes*:  
*is-pr-iso*  $\varphi$   $Fs_1$   $Fs_2 \implies \text{card}(\bigcup F \in \text{set } Fs_1. \text{set } F) = \text{card}(\bigcup F \in \text{set } Fs_2. \text{set } F)$   
 <proof>

Isomorphisms preserve the number of faces.

**lemma** *pr-iso-same-no-faces*:  
**assumes** *dist1*: *distinct*  $Fs_1$  **and** *dist2*: *distinct*  $Fs_2$   
**and** *inj1*: *inj-on*  $(\lambda xs. \{xs\} // \{\cong\})$  (set  $Fs_1$ )  
**and** *inj2*: *inj-on*  $(\lambda xs. \{xs\} // \{\cong\})$  (set  $Fs_2$ ) **and** *iso*: *is-pr-iso*  $\varphi$   $Fs_1$   $Fs_2$   
**shows**  $\text{length } Fs_1 = \text{length } Fs_2$   
 <proof>

**lemma** *is-Hom-distinct*:

$\llbracket \text{is-pr-Hom } \varphi \text{ } F_{s_1} \text{ } F_{s_2}; \forall F \in F_{s_1}. \text{distinct } F; \forall F \in F_{s_2}. \text{distinct } F \rrbracket$   
 $\implies \forall F \in F_{s_1}. \text{distinct}(\text{map } \varphi \text{ } F)$   
 <proof>

**lemma** *Collect-congs-eq-iff[simp]*:

$\text{Collect } ((\cong) \text{ } x) = \text{Collect } ((\cong) \text{ } y) \longleftrightarrow (x \cong (y::'a \text{ list}))$   
 <proof>

**lemma** *is-pr-Hom-trans*: **assumes**  $f: \text{is-pr-Hom } f \text{ } A \text{ } B$  **and**  $g: \text{is-pr-Hom } g \text{ } B \text{ } C$   
**shows**  $\text{is-pr-Hom } (g \circ f) \text{ } A \text{ } C$   
 <proof>

**lemma** *is-pr-Hom-rev*:

$\text{is-pr-Hom } \varphi \text{ } A \text{ } B \implies \text{is-pr-Hom } \varphi \text{ } (\text{rev } 'A) \text{ } (\text{rev } 'B)$   
 <proof>

A kind of recursion rule, a first step towards executability:

**lemma** *is-pr-Iso-rec*:

$\llbracket \text{inj-on } (\lambda xs. \{xs\} // \{\cong\}) \text{ } F_{s_1}; \text{inj-on } (\lambda xs. \{xs\} // \{\cong\}) \text{ } F_{s_2}; F_1 \in F_{s_1} \rrbracket \implies$   
 $\text{is-pr-Iso } \varphi \text{ } F_{s_1} \text{ } F_{s_2} =$   
 $(\exists F_2 \in F_{s_2}. \text{length } F_1 = \text{length } F_2 \wedge \text{is-pr-Iso } \varphi \text{ } (F_{s_1} - \{F_1\}) \text{ } (F_{s_2} - \{F_2\}))$   
 $\wedge (\exists n. \text{map } \varphi \text{ } F_1 = \text{rotate } n \text{ } F_2)$   
 $\wedge \text{inj-on } \varphi \text{ } (\bigcup F \in F_{s_1}. \text{set } F)$   
 <proof>

**lemma** *is-iso-Cons*:

$\llbracket \text{distinct } (F_1 \# F_{s_1}'); \text{distinct } F_{s_2};$   
 $\text{inj-on } (\lambda xs. \{xs\} // \{\cong\}) \text{ } (\text{set } (F_1 \# F_{s_1}')); \text{inj-on } (\lambda xs. \{xs\} // \{\cong\}) \text{ } (\text{set } F_{s_2}) \rrbracket$   
 $\implies$   
 $\text{is-pr-iso } \varphi \text{ } (F_1 \# F_{s_1}') \text{ } F_{s_2} =$   
 $(\exists F_2 \in \text{set } F_{s_2}. \text{length } F_1 = \text{length } F_2 \wedge \text{is-pr-iso } \varphi \text{ } F_{s_1}' \text{ } (\text{remove1 } F_2 \text{ } F_{s_2}))$   
 $\wedge (\exists n. \text{map } \varphi \text{ } F_1 = \text{rotate } n \text{ } F_2)$   
 $\wedge \text{inj-on } \varphi \text{ } (\text{set } F_1 \cup (\bigcup F \in \text{set } F_{s_1}'. \text{set } F))$   
 <proof>

## 2.3 Isomorphism tests

**lemma** *map-upd-submap*:

$x \notin \text{dom } m \implies (m(x \mapsto y) \subseteq_m m') = (m' x = \text{Some } y \wedge m \subseteq_m m')$   
 <proof>

**lemma** *map-of-zip-submap*:  $\llbracket \text{length } xs = \text{length } ys; \text{distinct } xs \rrbracket \implies$

$(\text{map-of } (\text{zip } xs \text{ } ys) \subseteq_m \text{Some } \circ f) = (\text{map } f \text{ } xs = ys)$   
 <proof>

**primrec** *pr-iso-test0* :: ('a  $\rightarrow$  'b)  $\Rightarrow$  'a fgraph  $\Rightarrow$  'b fgraph  $\Rightarrow$  bool **where**  
*pr-iso-test0* m [] Fs2 = (Fs2 = [])  
| *pr-iso-test0* m (F1#Fs1) Fs2 =  
( $\exists F_2 \in \text{set } Fs_2. \text{length } F_1 = \text{length } F_2 \wedge$   
( $\exists n. \text{let } m' = \text{map-of}(\text{zip } F_1 (\text{rotate } n F_2)) \text{ in}$   
if  $m \subseteq_m m ++ m' \wedge \text{inj-on } (m ++ m') (\text{dom}(m ++ m'))$   
then *pr-iso-test0* (m ++ m') Fs1 (remove1 F2 Fs2) else False))

**lemma** *map-compatI*: [ $f \subseteq_m \text{Some} \circ h; g \subseteq_m \text{Some} \circ h$ ]  $\Longrightarrow$   $f \subseteq_m f ++ g$   
<proof>

**lemma** *inj-on-map-addI1*:  
[ $\text{inj-on } m A; m \subseteq_m m ++ m'; A \subseteq \text{dom } m$ ]  $\Longrightarrow$   $\text{inj-on } (m ++ m') A$   
<proof>

**lemma** *map-image-eq*: [ $A \subseteq \text{dom } m; m \subseteq_m m'$ ]  $\Longrightarrow$   $m \text{ ' } A = m' \text{ ' } A$   
<proof>

**lemma** *inj-on-map-add-Un*:  
[ $\text{inj-on } m (\text{dom } m); \text{inj-on } m' (\text{dom } m'); m \subseteq_m \text{Some} \circ f; m' \subseteq_m \text{Some} \circ f;$   
 $\text{inj-on } f (\text{dom } m' \cup \text{dom } m); A = \text{dom } m'; B = \text{dom } m$ ]  
 $\Longrightarrow$   $\text{inj-on } (m ++ m') (A \cup B)$   
<proof>

**lemma** *map-of-zip-eq-SomeD*:  $\text{length } xs = \text{length } ys \Longrightarrow$   
 $\text{map-of } (\text{zip } xs ys) x = \text{Some } y \Longrightarrow y \in \text{set } ys$   
<proof>

**lemma** *inj-on-map-of-zip*:  
[ $\text{length } xs = \text{length } ys; \text{distinct } ys$ ]  
 $\Longrightarrow$   $\text{inj-on } (\text{map-of } (\text{zip } xs ys)) (\text{set } xs)$   
<proof>

**lemma** *pr-iso-test0-correct*:  $\bigwedge m Fs_2.$   
[ $\forall F \in \text{set } Fs_1. \text{distinct } F; \forall F \in \text{set } Fs_2. \text{distinct } F;$   
 $\text{distinct } Fs_1; \text{inj-on } (\lambda xs. \{xs\} // \{\cong\}) (\text{set } Fs_1);$   
 $\text{distinct } Fs_2; \text{inj-on } (\lambda xs. \{xs\} // \{\cong\}) (\text{set } Fs_2); \text{inj-on } m (\text{dom } m)$ ]  
 $\Longrightarrow$   
*pr-iso-test0* m Fs1 Fs2 =  
( $\exists \varphi. \text{is-pr-iso } \varphi Fs_1 Fs_2 \wedge m \subseteq_m \text{Some} \circ \varphi \wedge$   
 $\text{inj-on } \varphi (\text{dom } m \cup (\bigcup F \in \text{set } Fs_1. \text{set } F))$ )  
<proof>

**corollary** *pr-iso-test0-corr*:  
[ $\forall F \in \text{set } Fs_1. \text{distinct } F; \forall F \in \text{set } Fs_2. \text{distinct } F;$   
 $\text{distinct } Fs_1; \text{inj-on } (\lambda xs. \{xs\} // \{\cong\}) (\text{set } Fs_1);$   
 $\text{distinct } Fs_2; \text{inj-on } (\lambda xs. \{xs\} // \{\cong\}) (\text{set } Fs_2)$ ]  
 $\Longrightarrow$   
*pr-iso-test0* Map.empty Fs1 Fs2 = ( $\exists \varphi. \text{is-pr-iso } \varphi Fs_1 Fs_2$ )  
<proof>

Now we bound the number of rotations needed. We have to exclude the empty face  $\square$  to be able to restrict the search to  $n < \text{length } xs$  (which would otherwise be vacuous).

```

primrec pr-iso-test1 :: ('a  $\rightarrow$  'b)  $\Rightarrow$  'a fgraph  $\Rightarrow$  'b fgraph  $\Rightarrow$  bool where
  pr-iso-test1 m  $\square$  Fs2 = (Fs2 =  $\square$ )
| pr-iso-test1 m (F1#Fs1) Fs2 =
  ( $\exists$  F2  $\in$  set Fs2. length F1 = length F2  $\wedge$ 
   ( $\exists$  n < length F2. let m' = map-of(zip F1 (rotate n F2)) in
    if m  $\subseteq_m$  m ++ m'  $\wedge$  inj-on (m ++ m') (dom(m ++ m'))
    then pr-iso-test1 (m ++ m') Fs1 (remove1 F2 Fs2) else False))

```

**lemma** test0-conv-test1:

```

 $\bigwedge$  m Fs2.  $\square \notin$  set Fs2  $\Longrightarrow$  pr-iso-test1 m Fs1 Fs2 = pr-iso-test0 m Fs1 Fs2
<proof>

```

Thus correctness carries over to *pr-iso-test1*:

**corollary** pr-iso-test1-corr:

```

 $\llbracket \forall F \in$  set Fs1. distinct F;  $\forall F \in$  set Fs2. distinct F;  $\square \notin$  set Fs2;
  distinct Fs1; inj-on ( $\lambda xs.$  {xs} // { $\cong$ }) (set Fs1);
  distinct Fs2; inj-on ( $\lambda xs.$  {xs} // { $\cong$ }) (set Fs2)  $\rrbracket \Longrightarrow$ 
  pr-iso-test1 Map.empty Fs1 Fs2 = ( $\exists \varphi.$  is-pr-iso  $\varphi$  Fs1 Fs2)
<proof>

```

### 2.3.1 Implementing maps by lists

The representation are lists of pairs with no repetition in the first or second component.

**definition** oneone :: ('a \* 'b)list  $\Rightarrow$  bool **where**

```

oneone xys  $\equiv$  distinct(map fst xys)  $\wedge$  distinct(map snd xys)

```

**declare** oneone-def[simp]

**type-synonym**

```

('a,'b)tester = ('a * 'b)list  $\Rightarrow$  ('a * 'b)list  $\Rightarrow$  bool

```

**type-synonym**

```

('a,'b)merger = ('a * 'b)list  $\Rightarrow$  ('a * 'b)list  $\Rightarrow$  ('a * 'b)list

```

**primrec** pr-iso-test2 :: ('a,'b)tester  $\Rightarrow$  ('a,'b)merger  $\Rightarrow$

```

  ('a * 'b)list  $\Rightarrow$  'a fgraph  $\Rightarrow$  'b fgraph  $\Rightarrow$  bool where
  pr-iso-test2 tst mrg I  $\square$  Fs2 = (Fs2 =  $\square$ )
| pr-iso-test2 tst mrg I (F1#Fs1) Fs2 =
  ( $\exists$  F2  $\in$  set Fs2. length F1 = length F2  $\wedge$ 
   ( $\exists$  n < length F2. let I' = zip F1 (rotate n F2) in
    if tst I' I
    then pr-iso-test2 tst mrg (mrg I' I) Fs1 (remove1 F2 Fs2) else False))

```

**lemma** notin-range-map-of:

```

y  $\notin$  snd ' set xys  $\Longrightarrow$  Some y  $\notin$  range(map-of xys)
<proof>

```



**lemma** *inj-on-map-upd*:

$\llbracket \text{inj-on } m \text{ (dom } m); \text{ Some } y \notin \text{range } m \rrbracket \implies \text{inj-on } (m(x \mapsto y)) \text{ (dom } m)$   
 $\langle \text{proof} \rangle$

**lemma** [*simp*]:

$\text{distinct}(\text{map snd } xys) \implies \text{inj-on } (\text{map-of } xys) \text{ (dom}(\text{map-of } xys))$   
 $\langle \text{proof} \rangle$

**lemma** *lem*:  $\text{Ball } (\text{set } xs) P \implies \text{Ball } (\text{set } (\text{remove1 } x \text{ } xs)) P = \text{True}$

$\langle \text{proof} \rangle$

**lemma** *pr-iso-test2-conv-1*:

$\bigwedge I \text{ } Fs_2.$   
 $\llbracket \forall I I'. \text{oneone } I \longrightarrow \text{oneone } I' \longrightarrow$   
 $\quad \text{tst } I' I = (\text{let } m = \text{map-of } I; m' = \text{map-of } I'$   
 $\quad \quad \text{in } m \subseteq_m m ++ m' \wedge \text{inj-on } (m ++ m') \text{ (dom}(m ++ m')));$   
 $\forall I I'. \text{oneone } I \longrightarrow \text{oneone } I' \longrightarrow \text{tst } I' I$   
 $\quad \longrightarrow \text{map-of}(\text{mrg } I' I) = \text{map-of } I ++ \text{map-of } I';$   
 $\forall I I'. \text{oneone } I \wedge \text{oneone } I' \longrightarrow \text{tst } I' I \longrightarrow \text{oneone } (\text{mrg } I' I);$   
 $\text{oneone } I;$   
 $\forall F \in \text{set } Fs_1. \text{distinct } F; \forall F \in \text{set } Fs_2. \text{distinct } F \rrbracket \implies$   
 $\text{pr-iso-test2 } \text{tst } \text{mrg } I \text{ } Fs_1 \text{ } Fs_2 = \text{pr-iso-test1 } (\text{map-of } I) \text{ } Fs_1 \text{ } Fs_2$   
 $\langle \text{proof} \rangle$

A simple implementation

**definition** *compat* :: ('a,'b)tester **where**

$\text{compat } I I' ==$   
 $\forall (x,y) \in \text{set } I. \forall (x',y') \in \text{set } I'. (x = x') = (y = y')$

**lemma** *image-map-upd*:

$x \notin \text{dom } m \implies m(x \mapsto y) \text{ ' } A = m \text{ ' } (A - \{x\}) \cup (\text{if } x \in A \text{ then } \{\text{Some } y\} \text{ else } \{\})$   
 $\langle \text{proof} \rangle$

**lemma** *image-map-of-conv-Image*:

$\bigwedge A. \llbracket \text{distinct}(\text{map fst } xys) \rrbracket$   
 $\implies \text{map-of } xys \text{ ' } A = \text{Some } \text{ ' } (\text{set } xys \text{ " } A) \cup (\text{if } A \subseteq \text{fst } \text{ ' } \text{set } xys \text{ then } \{\} \text{ else } \{\text{None}\})$   
 $\langle \text{proof} \rangle$

**lemma** [*simp*]:  $m ++ m' \text{ ' } (\text{dom } m' - A) = m' \text{ ' } (\text{dom } m' - A)$

$\langle \text{proof} \rangle$

**declare** *Diff-subset* [*iff*]

**lemma** *compat-correct*:

$\llbracket \text{oneone } I; \text{oneone } I' \rrbracket \implies$   
 $\text{compat } I' I = (\text{let } m = \text{map-of } I; m' = \text{map-of } I'$   
 $\text{in } m \subseteq_m m ++ m' \wedge \text{inj-on } (m ++ m') (\text{dom}(m ++ m')))$   
 $\langle \text{proof} \rangle$

**corollary** *compat-corr*:

$\forall I I'. \text{oneone } I \longrightarrow \text{oneone } I' \longrightarrow$   
 $\text{compat } I' I = (\text{let } m = \text{map-of } I; m' = \text{map-of } I'$   
 $\text{in } m \subseteq_m m ++ m' \wedge \text{inj-on } (m ++ m') (\text{dom}(m ++ m')))$   
 $\langle \text{proof} \rangle$

**definition** *merge0* :: ('a,'b)merger **where**

$\text{merge0 } I' I \equiv [xy \leftarrow I'. \text{fst } xy \notin \text{fst } ' \text{ set } I] @ I$

**lemma** *help1*:

$\text{distinct}(\text{map } \text{fst } xys) \implies \text{map-of } (\text{filter } P \ xys) =$   
 $\text{map-of } xys \mid' \{x. \exists y. (x,y) \in \text{set } xys \wedge P(x,y)\}$   
 $\langle \text{proof} \rangle$

**lemma** *merge0-correct*:

$\forall I I'. \text{oneone } I \longrightarrow \text{oneone } I' \longrightarrow \text{compat } I' I$   
 $\longrightarrow \text{map-of}(\text{merge0 } I' I) = \text{map-of } I ++ \text{map-of } I'$   
 $\langle \text{proof} \rangle$

**lemma** *merge0-inv*:

$\forall I I'. \text{oneone } I \wedge \text{oneone } I' \longrightarrow \text{compat } I' I \longrightarrow \text{oneone } (\text{merge0 } I' I)$   
 $\langle \text{proof} \rangle$

**corollary** *pr-iso-test2-corr*:

$\llbracket \forall F \in \text{set } Fs_1. \text{distinct } F; \forall F \in \text{set } Fs_2. \text{distinct } F; [] \notin \text{set } Fs_2;$   
 $\text{distinct } Fs_1; \text{inj-on } (\lambda xs. \{xs\} / / \{\cong\}) (\text{set } Fs_1);$   
 $\text{distinct } Fs_2; \text{inj-on } (\lambda xs. \{xs\} / / \{\cong\}) (\text{set } Fs_2) \rrbracket \implies$   
 $\text{pr-iso-test2 } \text{compat } \text{merge0 } [] \ Fs_1 \ Fs_2 = (\exists \varphi. \text{is-pr-iso } \varphi \ Fs_1 \ Fs_2)$   
 $\langle \text{proof} \rangle$

Implementing merge as a recursive function:

**primrec** *merge* :: ('a,'b)merger **where**

$\text{merge } [] \ I = I$   
 $\mid \text{merge } (xy \# xys) \ I = (\text{let } (x,y) = xy \text{ in}$   
 $\text{if } \forall (x',y') \in \text{set } I. x \neq x' \text{ then } xy \# \text{merge } xys \ I \text{ else } \text{merge } xys \ I)$

**lemma** *merge-conv-merge0*:  $\text{merge } I' I = \text{merge0 } I' I$

$\langle \text{proof} \rangle$

**primrec** *pr-iso-test-rec* :: ('a \* 'b)list  $\Rightarrow$  'a fgraph  $\Rightarrow$  'b fgraph  $\Rightarrow$  bool **where**

$\text{pr-iso-test-rec } I \ [] \ Fs_2 = (Fs_2 = [])$

|  $pr\text{-iso-test-rec } I (F_1 \# F_{s_1}) F_{s_2} =$   
 $(\exists F_2 \in \text{set } F_{s_2}. \text{length } F_1 = \text{length } F_2 \wedge$   
 $(\exists n < \text{length } F_2. \text{let } I' = \text{zip } F_1 (\text{rotate } n F_2) \text{ in}$   
 $\text{compat } I' I \wedge pr\text{-iso-test-rec } (\text{merge } I' I) F_{s_1} (\text{remove1 } F_2 F_{s_2})))$

**lemma**  $pr\text{-iso-test-rec-conv-2}$ :

$\bigwedge I F_{s_2}. pr\text{-iso-test-rec } I F_{s_1} F_{s_2} = pr\text{-iso-test2 } \text{compat } \text{merge0 } I F_{s_1} F_{s_2}$   
 $\langle \text{proof} \rangle$

**corollary**  $pr\text{-iso-test-rec-corr}$ :

$\llbracket \forall F \in \text{set } F_{s_1}. \text{distinct } F; \forall F \in \text{set } F_{s_2}. \text{distinct } F; \llbracket \notin \text{set } F_{s_2};$   
 $\text{distinct } F_{s_1}; \text{inj-on } (\lambda xs. \{xs\} // \{\cong\}) (\text{set } F_{s_1});$   
 $\text{distinct } F_{s_2}; \text{inj-on } (\lambda xs. \{xs\} // \{\cong\}) (\text{set } F_{s_2}) \rrbracket \implies$   
 $pr\text{-iso-test-rec } \llbracket F_{s_1} F_{s_2} = (\exists \varphi. \text{is-pr-iso } \varphi F_{s_1} F_{s_2})$   
 $\langle \text{proof} \rangle$

**definition**  $pr\text{-iso-test} :: 'a \text{ fgraph} \Rightarrow 'b \text{ fgraph} \Rightarrow \text{bool}$  **where**

$pr\text{-iso-test } F_{s_1} F_{s_2} = pr\text{-iso-test-rec } \llbracket F_{s_1} F_{s_2}$

**corollary**  $pr\text{-iso-test-correct}$ :

$\llbracket \forall F \in \text{set } F_{s_1}. \text{distinct } F; \forall F \in \text{set } F_{s_2}. \text{distinct } F; \llbracket \notin \text{set } F_{s_2};$   
 $\text{distinct } F_{s_1}; \text{inj-on } (\lambda xs. \{xs\} // \{\cong\}) (\text{set } F_{s_1});$   
 $\text{distinct } F_{s_2}; \text{inj-on } (\lambda xs. \{xs\} // \{\cong\}) (\text{set } F_{s_2}) \rrbracket \implies$   
 $pr\text{-iso-test } F_{s_1} F_{s_2} = (\exists \varphi. \text{is-pr-iso } \varphi F_{s_1} F_{s_2})$   
 $\langle \text{proof} \rangle$

## 2.3.2 ‘Improper’ Isomorphisms

**definition**  $is\text{-Iso} :: ('a \Rightarrow 'b) \Rightarrow 'a \text{ Fgraph} \Rightarrow 'b \text{ Fgraph} \Rightarrow \text{bool}$  **where**

$is\text{-Iso } \varphi F_{s_1} F_{s_2} \equiv is\text{-pr-Iso } \varphi F_{s_1} F_{s_2} \vee is\text{-pr-Iso } \varphi F_{s_1} (\text{rev } ' F_{s_2})$

**definition**  $is\text{-iso} :: ('a \Rightarrow 'b) \Rightarrow 'a \text{ fgraph} \Rightarrow 'b \text{ fgraph} \Rightarrow \text{bool}$  **where**

$is\text{-iso } \varphi F_{s_1} F_{s_2} \equiv is\text{-Iso } \varphi (\text{set } F_{s_1}) (\text{set } F_{s_2})$

**definition**  $iso\text{-fgraph} :: 'a \text{ fgraph} \Rightarrow 'a \text{ fgraph} \Rightarrow \text{bool}$  (**infix**  $\simeq$  60) **where**

$g_1 \simeq g_2 \equiv \exists \varphi. is\text{-iso } \varphi g_1 g_2$

**lemma**  $iso\text{-fgraph-trans}$ : **assumes**  $f \simeq (g :: 'a \text{ fgraph})$  **and**  $g \simeq h$  **shows**  $f \simeq h$

$\langle \text{proof} \rangle$

**definition**  $iso\text{-test} :: 'a \text{ fgraph} \Rightarrow 'b \text{ fgraph} \Rightarrow \text{bool}$  **where**

$iso\text{-test } g_1 g_2 \iff pr\text{-iso-test } g_1 g_2 \vee pr\text{-iso-test } g_1 (\text{map rev } g_2)$

**theorem**  $iso\text{-correct}$ :

$\llbracket \forall F \in \text{set } F_{s_1}. \text{distinct } F; \forall F \in \text{set } F_{s_2}. \text{distinct } F; \llbracket \notin \text{set } F_{s_2};$   
 $\text{distinct } F_{s_1}; \text{inj-on } (\lambda xs. \{xs\} // \{\cong\}) (\text{set } F_{s_1});$

$distinct\ Fs_2; inj\text{-}on\ (\lambda xs.\{xs\}/\{\cong\})\ (set\ Fs_2)\ \mathbb{I}\ \Longrightarrow$   
 $iso\text{-}test\ Fs_1\ Fs_2 = (Fs_1 \simeq Fs_2)$   
 <proof>

**lemma** *iso-fgraph-refl*[*iff*]:  $g \simeq g$   
 <proof>

## 2.4 Elementhood and containment modulo

**interpretation** *qle-gr*: *quasi-order* ( $\simeq$ )  
 <proof>

**abbreviation** *qle-gr-in* :: 'a fgraph  $\Rightarrow$  'a fgraph set  $\Rightarrow$  bool (**infix**  $\in_{\simeq}$  60)

**where**  $x \in_{\simeq} M \equiv qle\text{-}gr.in\text{-}qle\ x\ M$

**abbreviation** *qle-gr-sub* :: 'a fgraph set  $\Rightarrow$  'a fgraph set  $\Rightarrow$  bool (**infix**  $\subseteq_{\simeq}$  60)

**where**  $x \subseteq_{\simeq} M \equiv qle\text{-}gr.subseteq\text{-}qle\ x\ M$

**abbreviation** *qle-gr-eq* :: 'a fgraph set  $\Rightarrow$  'a fgraph set  $\Rightarrow$  bool (**infix**  $=_{\simeq}$  60)

**where**  $x =_{\simeq} M \equiv qle\text{-}gr.seteq\text{-}qle\ x\ M$

end

## 3 More Rotation

**theory** *Rotation*

**imports** *ListAux PlaneGraphIso*

**begin**

**definition** *rotate-to* :: 'a list  $\Rightarrow$  'a  $\Rightarrow$  'a list **where**  
 $rotate\text{-}to\ vs\ v \equiv v \# snd\ (splitAt\ v\ vs) @ fst\ (splitAt\ v\ vs)$

**definition** *rotate-min* :: nat list  $\Rightarrow$  nat list **where**  
 $rotate\text{-}min\ vs \equiv rotate\text{-}to\ vs\ (min\text{-}list\ vs)$

**lemma** *cong-rotate-to*:

$x \in set\ xs \Longrightarrow xs \cong rotate\text{-}to\ xs\ x$   
 <proof>

**lemma** *face-cong-if-norm-eq*:

$\mathbb{I}\ rotate\text{-}min\ xs = rotate\text{-}min\ ys; xs \neq []; ys \neq []\ \mathbb{I}\ \Longrightarrow xs \cong ys$   
 <proof>

**lemma** *norm-eq-if-face-cong*:

$\mathbb{I}\ xs \cong ys; distinct\ xs; xs \neq []\ \mathbb{I}\ \Longrightarrow rotate\text{-}min\ xs = rotate\text{-}min\ ys$   
 <proof>

**lemma** *norm-eq-iff-face-cong*:

$\mathbb{I}\ distinct\ xs; xs \neq []; ys \neq []\ \mathbb{I}\ \Longrightarrow$   
 $(rotate\text{-}min\ xs = rotate\text{-}min\ ys) = (xs \cong ys)$

*<proof>*

**lemma** *inj-on-rotate-min-iff*:

**assumes**  $\forall vs \in A. \text{distinct } vs \ \square \notin A$

**shows** *inj-on rotate-min*  $A = \text{inj-on } (\lambda vs. \{vs\} // \{\cong\}) A$

*<proof>*

**end**

## 4 Graph

**theory** *Graph*

**imports** *Rotation*

**begin**

**syntax**

*-UNION1*  $:: \text{pttrns} \Rightarrow 'b \text{ set} \Rightarrow 'b \text{ set} \quad ((\exists \cup (\text{unbreakable}).) / -) [0, 10]$   
*10)*

*-INTER1*  $:: \text{pttrns} \Rightarrow 'b \text{ set} \Rightarrow 'b \text{ set} \quad ((\exists \cap (\text{unbreakable}).) / -) [0, 10]$   
*10)*

*-UNION*  $:: \text{pttrn} \Rightarrow 'a \text{ set} \Rightarrow 'b \text{ set} \Rightarrow 'b \text{ set} \quad ((\exists \cup (\text{unbreakable}). \in -) / -) [0,$   
*0, 10] 10)*

*-INTER*  $:: \text{pttrn} \Rightarrow 'a \text{ set} \Rightarrow 'b \text{ set} \Rightarrow 'b \text{ set} \quad ((\exists \cap (\text{unbreakable}). \in -) / -) [0,$   
*0, 10] 10)*

### 4.1 Notation

**type-synonym** *vertex* = *nat*

**consts**

*vertices*  $:: 'a \Rightarrow \text{vertex list}$

*edges*  $:: 'a \Rightarrow (\text{vertex} \times \text{vertex}) \text{ set } (\mathcal{E})$

**abbreviation** *vertices-set*  $:: 'a \Rightarrow \text{vertex set } (\mathcal{V})$  **where**

$\mathcal{V} f \equiv \text{set } (\text{vertices } f)$

### 4.2 Faces

We represent faces by (distinct) lists of vertices and a face type.

**datatype** *facetyp* = *Final* | *Nonfinal*

**datatype** *face* = *Face* (*vertex list*) *facetyp*

**consts** *final*  $:: 'a \Rightarrow \text{bool}$

**consts** *type*  $:: 'a \Rightarrow \text{facetyp}$

**overloading**

```

    final-face ≡ final :: face ⇒ bool
    type-face ≡ type :: face ⇒ facetype
    vertices-face ≡ vertices :: face ⇒ vertex list
    cong-face ≡ pr-isomorphic :: face ⇒ face ⇒ bool
begin

primrec final-face where
    final (Face vs f) = (case f of Final ⇒ True | Nonfinal ⇒ False)

primrec type-face where
    type (Face vs f) = f

primrec vertices-face where
    vertices (Face vs f) = vs

definition cong-face :: face ⇒ face ⇒ bool
    where (f1 :: face) ≅ f2 ≡ vertices f1 ≅ vertices f2

end

The following operation makes a face final.

definition setFinal :: face ⇒ face where
    setFinal f ≡ Face (vertices f) Final

The function nextVertex (written  $f \cdot v$ ) is based on nextElem, that returns
the successor of an element in a list.

primrec nextElem :: 'a list ⇒ 'a ⇒ 'a ⇒ 'a where
    nextElem [] b x = b
| nextElem (a#as) b x =
    (if x=a then (case as of [] ⇒ b | (a'#as') ⇒ a') else nextElem as b x)

definition nextVertex :: face ⇒ vertex ⇒ vertex where
    f · ≡ let vs = vertices f in nextElem vs (hd vs)

nextVertices is  $n$ -fold application of nextvertex.

definition nextVertices :: face ⇒ nat ⇒ vertex ⇒ vertex where
    fn · v ≡ (f · ^ n) v

lemma nextV2: f2 · v = f · (f · v)
⟨proof⟩
overloading op-vertices ≡ Graph.op :: vertex list ⇒ vertex list
begin
    definition (vs::vertex list)op ≡ rev vs
end

overloading op-graph ≡ Graph.op :: face ⇒ face
begin
    primrec op-graph where (Face vs f)op = Face (rev vs) f
end

```

*<proof><proof>*

**definition** *prevVertex* :: *face*  $\Rightarrow$  *vertex*  $\Rightarrow$  *vertex* **where**  
*f*<sup>-1</sup> · *v*  $\equiv$  (*let vs = vertices f in nextElem (rev vs) (last vs) v*)

**abbreviation**

*triangle* :: *face*  $\Rightarrow$  *bool* **where**  
*triangle f* == *|vertices f| = 3*

### 4.3 Graphs

**datatype** *graph* = *Graph (face list) nat face list list nat list*

**primrec** *faces* :: *graph*  $\Rightarrow$  *face list* **where**  
*faces (Graph fs n f h)* = *fs*

**abbreviation**

*Faces* :: *graph*  $\Rightarrow$  *face set* (*F*) **where**  
*F g* == *set(faces g)*

**primrec** *countVertices* :: *graph*  $\Rightarrow$  *nat* **where**  
*countVertices (Graph fs n f h)* = *n*

**overloading**

*vertices-graph*  $\equiv$  *vertices* :: *graph*  $\Rightarrow$  *vertex list*

**begin**

**primrec** *vertices-graph* **where** *vertices (Graph fs n f h)* = *[0 ..< n]*

**end**

**lemma** *vertices-graph*: *vertices g* = *[0 ..< countVertices g]*  
*<proof>*

**lemma** *in-vertices-graph*:  
*v*  $\in$  *set (vertices g)* = (*v* < *countVertices g*)  
*<proof>*

**lemma** *len-vertices-graph*:  
*|vertices g|* = *countVertices g*  
*<proof>*

**primrec** *faceListAt* :: *graph*  $\Rightarrow$  *face list list* **where**  
*faceListAt (Graph fs n f h)* = *f*

**definition** *facesAt* :: *graph*  $\Rightarrow$  *vertex*  $\Rightarrow$  *face list* **where**  
*facesAt g v*  $\equiv$  ~~*if v  $\in$  set(vertices g) then*~~ *faceListAt g ! v* ~~*else []*~~

**primrec** *heights* :: *graph*  $\Rightarrow$  *nat list* **where**  
*heights (Graph fs n f h)* = *h*

**definition** *height* :: *graph*  $\Rightarrow$  *vertex*  $\Rightarrow$  *nat* **where**  
*height* *g* *v*  $\equiv$  *heights* *g* ! *v*

**definition** *graph* :: *nat*  $\Rightarrow$  *graph* **where**  
*graph* *n*  $\equiv$   
 (let *vs* = [0 ..< *n*];  
   *fs* = [ *Face vs Final*, *Face (rev vs) Nonfinal* ]  
 in (*Graph* *fs* *n* (*replicate* *n* *fs*) (*replicate* *n* 0)))

#### 4.4 Operations on graphs

final graph, final / nonfinal faces

**definition** *finals* :: *graph*  $\Rightarrow$  *face list* **where**  
*finals* *g*  $\equiv$  [*f*  $\leftarrow$  *faces* *g*. *final* *f*]

**definition** *nonFinals* :: *graph*  $\Rightarrow$  *face list* **where**  
*nonFinals* *g*  $\equiv$  [*f*  $\leftarrow$  *faces* *g*.  $\neg$  *final* *f*]

**definition** *countNonFinals* :: *graph*  $\Rightarrow$  *nat* **where**  
*countNonFinals* *g*  $\equiv$  |*nonFinals* *g*|

**overloading** *finalGraph*  $\equiv$  *final* :: *graph*  $\Rightarrow$  *bool*  
**begin**  
**definition** *finalGraph* *g*  $\equiv$  (*nonFinals* *g* = [])  
**end**

**lemma** *finalGraph-faces[simp]*: *final* *g*  $\Longrightarrow$  *finals* *g* = *faces* *g*  
 <*proof*>

**lemma** *finalGraph-face*: *final* *g*  $\Longrightarrow$  *f*  $\in$  *set* (*faces* *g*)  $\Longrightarrow$  *final* *f*  
 <*proof*>

**definition** *finalVertex* :: *graph*  $\Rightarrow$  *vertex*  $\Rightarrow$  *bool* **where**  
*finalVertex* *g* *v*  $\equiv$   $\forall$  *f*  $\in$  *set*(*facesAt* *g* *v*). *final* *f*

**lemma** *finalVertex-final-face[dest]*:  
*finalVertex* *g* *v*  $\Longrightarrow$  *f*  $\in$  *set* (*facesAt* *g* *v*)  $\Longrightarrow$  *final* *f*  
 <*proof*>

counting faces

**definition** *degree* :: *graph*  $\Rightarrow$  *vertex*  $\Rightarrow$  *nat* **where**  
*degree* *g* *v*  $\equiv$  |*facesAt* *g* *v*|

**definition** *tri* :: *graph*  $\Rightarrow$  *vertex*  $\Rightarrow$  *nat* **where**  
*tri* *g* *v*  $\equiv$  |[*f*  $\leftarrow$  *facesAt* *g* *v*. *final* *f*  $\wedge$  |*vertices* *f*| = 3]|

**definition** *quad* :: *graph*  $\Rightarrow$  *vertex*  $\Rightarrow$  *nat* **where**  
*quad* *g* *v*  $\equiv$  |[*f*  $\leftarrow$  *facesAt* *g* *v*. *final* *f*  $\wedge$  |*vertices* *f*| = 4]|



**definition** *except* :: *graph*  $\Rightarrow$  *vertex*  $\Rightarrow$  *nat* **where**  
*except* *g v*  $\equiv$   $||f \leftarrow \text{facesAt } g \ v. \text{ final } f \wedge 5 \leq |\text{vertices } f| ||$

**definition** *vertextype* :: *graph*  $\Rightarrow$  *vertex*  $\Rightarrow$  *nat*  $\times$  *nat*  $\times$  *nat* **where**  
*vertextype* *g v*  $\equiv$  (*tri* *g v*, *quad* *g v*, *except* *g v*)

**lemma**[*simp*]:  $0 \leq \text{tri } g \ v$  *<proof>*

**lemma**[*simp*]:  $0 \leq \text{quad } g \ v$  *<proof>*

**lemma**[*simp*]:  $0 \leq \text{except } g \ v$  *<proof>*

**definition** *exceptionalVertex* :: *graph*  $\Rightarrow$  *vertex*  $\Rightarrow$  *bool* **where**  
*exceptionalVertex* *g v*  $\equiv$  *except* *g v*  $\neq$  0

**definition** *noExceptionals* :: *graph*  $\Rightarrow$  *vertex set*  $\Rightarrow$  *bool* **where**  
*noExceptionals* *g V*  $\equiv$   $(\forall v \in V. \neg \text{exceptionalVertex } g \ v)$

An edge (*a*, *b*) is contained in face *f*, *b* is the successor of *a* in *f*.

**overloading** *edges-graph*  $\equiv$  *edges* :: *graph*  $\Rightarrow$  (*vertex*  $\times$  *vertex*) *set*  
**begin**  
**definition**  $\mathcal{E}$  (*g*::*graph*)  $\equiv$   $\bigcup_{f \in \mathcal{F}_g} \text{edges } f$   
**end**

**definition** *neighbors* :: *graph*  $\Rightarrow$  *vertex*  $\Rightarrow$  *vertex list* **where**  
*neighbors* *g v*  $\equiv$   $[f \cdot v. f \leftarrow \text{facesAt } g \ v]$

## 4.5 Navigation in graphs

The function *s'* permutating the faces at a vertex, is implemented by the function *nextFace*

**definition** *nextFace* :: *graph*  $\times$  *vertex*  $\Rightarrow$  *face*  $\Rightarrow$  *face* **where**

**definition** *directedLength* :: *face*  $\Rightarrow$  *vertex*  $\Rightarrow$  *vertex*  $\Rightarrow$  *nat* **where**  
*directedLength* *f a b*  $\equiv$   
*if* *a = b* *then* 0 *else*  $|(between (\text{vertices } f) \ a \ b)| + 1$

## 4.6 Code generator setup

**definition** *final-face* :: *face*  $\Rightarrow$  *bool* **where**  
*final-face-code-def*: *final-face* = *final*  
**declare** *final-face-code-def* [*symmetric*, *code-unfold*]

**lemma** *final-face-code* [*code*]:  
*final-face* (*Face* vs *Final*)  $\longleftrightarrow$  *True*

*final-face* (*Face vs Nonfinal*)  $\longleftrightarrow$  *False*  
*<proof>*

**definition** *final-graph* :: *graph*  $\Rightarrow$  *bool* **where**  
*final-graph-code-def*: *final-graph* = *final*  
**declare** *final-graph-code-def* [*symmetric*, *code-unfold*]

**lemma** *final-graph-code* [*code*]: *final-graph* *g* = *List.null* (*nonFinals* *g*)  
*<proof>*

**definition** *vertices-face* :: *face*  $\Rightarrow$  *vertex list* **where**  
*vertices-face-code-def*: *vertices-face* = *vertices*  
**declare** *vertices-face-code-def* [*symmetric*, *code-unfold*]

**lemma** *vertices-face-code* [*code*]: *vertices-face* (*Face vs f*) = *vs*  
*<proof>*

**definition** *vertices-graph* :: *graph*  $\Rightarrow$  *vertex list* **where**  
*vertices-graph-code-def*: *vertices-graph* = *vertices*  
**declare** *vertices-graph-code-def* [*symmetric*, *code-unfold*]

**lemma** *vertices-graph-code* [*code*]:  
*vertices-graph* (*Graph fs n f h*) = [*0* ..< *n*]  
*<proof>*

**end**

## 5 Syntax for operations on immutable arrays

**theory** *IArray-Syntax*  
**imports** *Main HOL-Library.IArray*  
**begin**

### 5.1 Tabulation

**definition** *tabulate* :: *nat*  $\Rightarrow$  (*nat*  $\Rightarrow$  '*a*)  $\Rightarrow$  '*a* *iarray*  
**where**

*tabulate* *n f* = *IArray.of-fun* *f n*

**definition** *tabulate2* :: *nat*  $\Rightarrow$  *nat*  $\Rightarrow$  (*nat*  $\Rightarrow$  *nat*  $\Rightarrow$  '*a*)  $\Rightarrow$  '*a* *iarray iarray*  
**where**

*tabulate2* *m n f* = *IArray.of-fun* ( $\lambda i$ . *IArray.of-fun* (*f i*) *n*) *m*

**definition** *tabulate3* :: *nat*  $\Rightarrow$  *nat*  $\Rightarrow$  *nat*  $\Rightarrow$

(*nat*  $\Rightarrow$  *nat*  $\Rightarrow$  *nat*  $\Rightarrow$  '*a*)  $\Rightarrow$  '*a* *iarray iarray iarray* **where**

*tabulate3* *l m n f*  $\equiv$  *IArray.of-fun* ( $\lambda i$ . *IArray.of-fun* ( $\lambda j$ . *IArray.of-fun* ( $\lambda k$ . *f i j k*) *n*) *m*) *l*

**syntax**

```

-tabulate :: 'a => pptrn => nat => 'a iarray (([-. - < -]))
-tabulate2 :: 'a => pptrn => nat => pptrn => nat => 'a iarray
  (([-. - < -, - < -]))
-tabulate3 :: 'a => pptrn => nat => pptrn => nat => pptrn => nat => 'a iarray
  (([-. - < -, - < -, - < -]))

```

### translations

```

[[f. x < n]] == CONST tabulate n (\x. f)
[[f. x < m, y < n]] == CONST tabulate2 m n (\x y. f)
[[f. x < l, y < m, z < n]] == CONST tabulate3 l m n (\x y z. f)

```

## 5.2 Access

**abbreviation** *sub1-syntax* :: 'a iarray => nat => 'a (([-]) [1000] 999)

**where**

```
a[[n]] ≡ IArray.sub a n
```

**abbreviation** *sub2-syntax* :: 'a iarray iarray => nat => nat => 'a (([-,-]) [1000] 999)

**where**

```
as[[m, n]] ≡ IArray.sub (IArray.sub as m) n
```

**abbreviation** *sub3-syntax* :: 'a iarray iarray iarray => nat => nat => nat => 'a (([-,-,-]) [1000] 999)

**where**

```
as[[l, m, n]] ≡ IArray.sub (IArray.sub (IArray.sub as l) m) n
```

examples:  $[[0::'a. i < 5]]$ ,  $[[i. i < 5, j < 3]]$

**end**

## 6 Enumerating Patches

**theory** *Enumerator*

**imports** *Graph IArray-Syntax*

**begin**

Generates an Enumeration of lists. (See Kepler98, PartIII, section 8, p.11).

Used to construct all possible extensions of an unfinished outer face  $F$  with *outer* vertices by a new finished inner face with *inner* vertices, such a fixed edge  $e$  of the outer face is also contained in the inner face.

Label the vertices of  $F$  consecutively  $0, \dots, outer - 1$ , with  $0$  and  $outer - 1$  the endpoints of  $e$ .

Generate all lists

$$[a_0, \dots, a_{inner-1}]$$

of length *inner*, such that  $0 = a_0 \leq a_1 \dots a_{inner-2} < a_{inner-1}$ . Every list represents an inner face, with vertices  $v_0, \dots, v_{inner-1}$ .

Construct the vertices  $v_0, \dots, v_{inner-1}$  inductively: If  $i = 1$  or  $a_i \neq a_{i-1}$ , we set  $v_i$  to the vertex with index  $a_i$  of  $F$ . But if  $a_i = a_{i-1}$ , we add a new vertex  $v_i$  to the planar map. The new face is to be drawn along the edge  $e$  over the face  $F$ .

As we run over all *inner* and all lists  $[a_0, \dots, a_{inner_1}]$ , we run over all osibilites from the finishe face along the edge  $e$  inside  $F$ .

**definition** *enumBase* ::  $nat \Rightarrow nat \text{ list list}$  **where**  
*enumBase nmax*  $\equiv [[i]. i \leftarrow [0 ..< Suc nmax]]$

**definition** *enumAppend* ::  $nat \Rightarrow nat \text{ list list} \Rightarrow nat \text{ list list}$  **where**  
*enumAppend nmax iss*  $\equiv \bigsqcup_{is \in iss} [is @ [n]. n \leftarrow [last is ..< Suc nmax]]$

**definition** *enumerator* ::  $nat \Rightarrow nat \Rightarrow nat \text{ list list}$  **where**  
*enumerator inner outer*  $\equiv$   
 let  $nmax = outer - 2; k = inner - 3$  in  
 $[[0] @ is @ [outer - 1]. is \leftarrow (enumAppend nmax \hat{\wedge} k) (enumBase nmax)]$

**definition** *enumTab* ::  $nat \text{ list list iarray iarray}$  **where**  
*enumTab*  $\equiv \llbracket enumerator \text{ inner outer}. inner < 9, outer < 9 \rrbracket$

**definition** *enum* ::  $nat \Rightarrow nat \Rightarrow nat \text{ list list}$  **where**  
*enum inner outer*  $\equiv$  if  $inner < 9 \wedge outer < 9$  then *enumTab* $\llbracket inner, outer \rrbracket$   
 else *enumerator inner outer*

**primrec** *hideDupsRec* ::  $'a \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ option list}$  **where**  
*hideDupsRec*  $a [] = []$   
 $| \text{hideDupsRec } a (b \# bs) =$   
 (if  $a = b$  then  $None \# \text{hideDupsRec } b \text{ bs}$   
 else  $Some b \# \text{hideDupsRec } b \text{ bs}$ )

**primrec** *hideDups* ::  $'a \text{ list} \Rightarrow 'a \text{ option list}$  **where**  
*hideDups*  $[] = []$   
 $| \text{hideDups } (b \# bs) = Some b \# \text{hideDupsRec } b \text{ bs}$

**definition** *indexToVertexList* ::  $face \Rightarrow vertex \Rightarrow nat \text{ list} \Rightarrow vertex \text{ option list}$   
**where**  
*indexToVertexList f v is*  $\equiv \text{hideDups } [f^k.v. k \leftarrow is]$

**end**

## 7 Subdividing a Face

**theory** *FaceDivision*  
**imports** *Graph*  
**begin**

**definition** *split-face* :: *face*  $\Rightarrow$  *vertex*  $\Rightarrow$  *vertex*  $\Rightarrow$  *vertex list*  $\Rightarrow$  *face*  $\times$  *face* **where**  
*split-face* *f* *ram*<sub>1</sub> *ram*<sub>2</sub> *newVs*  $\equiv$  *let* *vs* = *vertices* *f*;  
*f*<sub>1</sub> = [*ram*<sub>1</sub>] @ *between* *vs* *ram*<sub>1</sub> *ram*<sub>2</sub> @ [*ram*<sub>2</sub>];  
*f*<sub>2</sub> = [*ram*<sub>2</sub>] @ *between* *vs* *ram*<sub>2</sub> *ram*<sub>1</sub> @ [*ram*<sub>1</sub>] *in*  
(*Face* (*rev* *newVs* @ *f*<sub>1</sub>) *Nonfinal*,  
*Face* (*f*<sub>2</sub> @ *newVs*) *Nonfinal*)

**definition** *replacefacesAt* :: *nat list*  $\Rightarrow$  *face*  $\Rightarrow$  *face list*  $\Rightarrow$  *face list list*  $\Rightarrow$  *face list list* **where**  
*replacefacesAt* *ns* *f* *fs* *F*  $\equiv$  *mapAt* *ns* (*replace* *f* *fs*) *F*

**definition** *makeFaceFinalFaceList* :: *face*  $\Rightarrow$  *face list*  $\Rightarrow$  *face list* **where**  
*makeFaceFinalFaceList* *f* *fs*  $\equiv$  *replace* *f* [*setFinal* *f*] *fs*

**definition** *makeFaceFinal* :: *face*  $\Rightarrow$  *graph*  $\Rightarrow$  *graph* **where**  
*makeFaceFinal* *f* *g*  $\equiv$   
*Graph* (*makeFaceFinalFaceList* *f* (*faces* *g*))  
(*countVertices* *g*)  
[*makeFaceFinalFaceList* *f* *fs*. *fs*  $\leftarrow$  *faceListAt* *g*]  
(*heights* *g*)

**definition** *heightsNewVertices* :: *nat*  $\Rightarrow$  *nat*  $\Rightarrow$  *nat*  $\Rightarrow$  *nat list* **where**  
*heightsNewVertices* *h*<sub>1</sub> *h*<sub>2</sub> *n*  $\equiv$  [*min* (*h*<sub>1</sub> + *i* + 1) (*h*<sub>2</sub> + *n* - *i*). *i*  $\leftarrow$  [0 ..< *n*]]

**definition** *splitFace*  
:: *graph*  $\Rightarrow$  *vertex*  $\Rightarrow$  *vertex*  $\Rightarrow$  *face*  $\Rightarrow$  *vertex list*  $\Rightarrow$  *face*  $\times$  *face*  $\times$  *graph* **where**  
*splitFace* *g* *ram*<sub>1</sub> *ram*<sub>2</sub> *oldF* *newVs*  $\equiv$   
*let* *fs* = *faces* *g*;  
*n* = *countVertices* *g*;  
*Fs* = *faceListAt* *g*;  
*h* = *heights* *g*;  
*vs*<sub>1</sub> = *between* (*vertices* *oldF*) *ram*<sub>1</sub> *ram*<sub>2</sub>;  
*vs*<sub>2</sub> = *between* (*vertices* *oldF*) *ram*<sub>2</sub> *ram*<sub>1</sub>;  
(*f*<sub>1</sub>, *f*<sub>2</sub>) = *split-face* *oldF* *ram*<sub>1</sub> *ram*<sub>2</sub> *newVs*;  
*Fs* = *replacefacesAt* *vs*<sub>1</sub> *oldF* [*f*<sub>1</sub>] *Fs*;  
*Fs* = *replacefacesAt* *vs*<sub>2</sub> *oldF* [*f*<sub>2</sub>] *Fs*;  
*Fs* = *replacefacesAt* [*ram*<sub>1</sub>] *oldF* [*f*<sub>2</sub>, *f*<sub>1</sub>] *Fs*;  
*Fs* = *replacefacesAt* [*ram*<sub>2</sub>] *oldF* [*f*<sub>1</sub>, *f*<sub>2</sub>] *Fs*;  
*Fs* = *Fs* @ *replicate* |*newVs*| [*f*<sub>1</sub>, *f*<sub>2</sub>] *in*  
(*f*<sub>1</sub>, *f*<sub>2</sub>, *Graph* ((*replace* *oldF* [*f*<sub>2</sub>] *fs*)@ [*f*<sub>1</sub>]))

$$(n + |newVs|)$$

$$Fs$$

$$(h @ heightsNewVertices (h!ram_1)(h!ram_2) |newVs|))$$

**primrec** *subdivFace'* :: *graph*  $\Rightarrow$  *face*  $\Rightarrow$  *vertex*  $\Rightarrow$  *nat*  $\Rightarrow$  *vertex option list*  $\Rightarrow$  *graph* **where**

*subdivFace' g f u n []* = *makeFaceFinal f g*

| *subdivFace' g f u n (vo#vos)* =

  (*case vo of None*  $\Rightarrow$  *subdivFace' g f u (Suc n) vos*

  | (*Some v*)  $\Rightarrow$

*if f.u = v*  $\wedge$  *n = 0*

*then subdivFace' g f v 0 vos*

*else let ws = [countVertices g ..< countVertices g + n];*

    (*f*<sub>1</sub>, *f*<sub>2</sub>, *g'*) = *splitFace g u v f ws in*

*subdivFace' g' f<sub>2</sub> v 0 vos*)

**definition** *subdivFace* :: *graph*  $\Rightarrow$  *face*  $\Rightarrow$  *vertex option list*  $\Rightarrow$  *graph* **where**

*subdivFace g f vos*  $\equiv$  *subdivFace' g f (the(hd vos)) 0 (tl vos)*

**end**

## 8 Transitive Closure of Successor List Function

**theory** *RTranCl*

**imports** *Main*

**begin**

The reflexive transitive closure of a relation induced by a function of type *'a*  $\Rightarrow$  *'a list*. Instead of defining the closure again it would have been simpler to take  $\{(x, y). y \in \text{set } (f x)\}^*$ .

**abbreviation** (*input*)

*in-set* :: *'a*  $\Rightarrow$  (*'a*  $\Rightarrow$  *'b list*)  $\Rightarrow$  *'b*  $\Rightarrow$  *bool* (*- [-]*  $\rightarrow$  *- [55,0,55] 50*) **where**

*g [succs]*  $\rightarrow$  *g'*  $\equiv$  *g' \in set (succs g)*

**inductive-set**

*RTranCl* :: (*'a*  $\Rightarrow$  *'a list*)  $\Rightarrow$  (*'a* \* *'a*) *set*

**and** *in-RTranCl* :: *'a*  $\Rightarrow$  (*'a*  $\Rightarrow$  *'a list*)  $\Rightarrow$  *'a*  $\Rightarrow$  *bool*

  (*- [-]*  $\rightarrow$  \* *- [55,0,55] 50*)

**for** *succs* :: *'a*  $\Rightarrow$  *'a list*

**where**

*g [succs]*  $\rightarrow$  \* *g'*  $\equiv$  (*g, g'*)  $\in$  *RTranCl succs*

| *refl*: *g [succs]*  $\rightarrow$  \* *g*

| *succs*: *g [succs]*  $\rightarrow$  *g'*  $\implies$  *g' [succs]*  $\rightarrow$  \* *g''*  $\implies$  *g [succs]*  $\rightarrow$  \* *g''*

**inductive-cases** *RTranCl-elim*: (*h, h'*) : *RTranCl succs*

**lemma** *RTranCl-induct*:  
 $(h, h') \in RTranCl\ succs \implies$   
 $P\ h \implies$   
 $(\bigwedge g\ g'.\ g' \in set\ (succs\ g) \implies P\ g \implies P\ g') \implies$   
 $P\ h'$   
 $\langle proof \rangle$

**definition** *invariant* ::  $('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a\ list) \Rightarrow bool$  **where**  
 $invariant\ P\ succs \equiv \forall g\ g'.\ g' \in set(succs\ g) \longrightarrow P\ g \longrightarrow P\ g'$

**lemma** *invariantE*:  
 $invariant\ P\ succs \implies g\ [succs] \rightarrow g' \implies P\ g \implies P\ g'$   
 $\langle proof \rangle$

**lemma** *inv-subset*:  
 $invariant\ P\ f \implies (\bigwedge g.\ P\ g \implies set(f'\ g) \subseteq set(f\ g)) \implies invariant\ P\ f'$   
 $\langle proof \rangle$

**lemma** *RTranCl-inv*:  
 $invariant\ P\ succs \implies (g, g') \in RTranCl\ succs \implies P\ g \implies P\ g'$   
 $\langle proof \rangle$

**lemma** *RTranCl-subset2*:  
**assumes**  $a: (s, g) : RTranCl\ f$   
**shows**  $(\bigwedge g.\ (s, g) \in RTranCl\ f \implies set(f\ g) \subseteq set(h\ g)) \implies (s, g) : RTranCl\ h$   
 $\langle proof \rangle$

**end**

## 9 Plane Graph Enumeration

**theory** *Plane*  
**imports** *Enumerator FaceDivision RTranCl*  
**begin**

**definition** *maxGon* ::  $nat \Rightarrow nat$  **where**  
 $maxGon\ p \equiv p+3$

**declare** *maxGon-def* [*simp*]

**definition** *duplicateEdge* ::  $graph \Rightarrow face \Rightarrow vertex \Rightarrow vertex \Rightarrow bool$  **where**  
 $duplicateEdge\ g\ f\ a\ b \equiv$   
 $2 \leq directedLength\ f\ a\ b \wedge 2 \leq directedLength\ f\ b\ a \wedge b \in set\ (neighbors\ g\ a)$

**primrec** *containsUnacceptableEdgeSnd* ::

$(\text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool}) \Rightarrow \text{nat} \Rightarrow \text{nat list} \Rightarrow \text{bool}$  **where**  
 $\text{containsUnacceptableEdgeSnd } N \ v \ [] = \text{False} \mid$   
 $\text{containsUnacceptableEdgeSnd } N \ v \ (w\#ws) =$   
 $(\text{case } ws \text{ of } [] \Rightarrow \text{False}$   
 $\mid (w'\#ws') \Rightarrow \text{if } v < w \wedge w < w' \wedge N \ w \ w' \text{ then True}$   
 $\text{else containsUnacceptableEdgeSnd } N \ w \ ws)$

**primrec**  $\text{containsUnacceptableEdge} :: (\text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool}) \Rightarrow \text{nat list} \Rightarrow \text{bool}$   
**where**  
 $\text{containsUnacceptableEdge } N \ [] = \text{False} \mid$   
 $\text{containsUnacceptableEdge } N \ (v\#vs) =$   
 $(\text{case } vs \text{ of } [] \Rightarrow \text{False}$   
 $\mid (w\#ws) \Rightarrow \text{if } v < w \wedge N \ v \ w \text{ then True}$   
 $\text{else containsUnacceptableEdgeSnd } N \ v \ vs)$

**definition**  $\text{containsDuplicateEdge} :: \text{graph} \Rightarrow \text{face} \Rightarrow \text{vertex} \Rightarrow \text{nat list} \Rightarrow \text{bool}$   
**where**  
 $\text{containsDuplicateEdge } g \ f \ v \ is \equiv$   
 $\text{containsUnacceptableEdge } (\lambda i \ j. \text{duplicateEdge } g \ f \ (f^i \cdot v) \ (f^j \cdot v)) \ is$

**definition**  $\text{containsDuplicateEdge}' :: \text{graph} \Rightarrow \text{face} \Rightarrow \text{vertex} \Rightarrow \text{nat list} \Rightarrow \text{bool}$   
**where**  
 $\text{containsDuplicateEdge}' \ g \ f \ v \ is \equiv$   
 $2 \leq |is| \wedge$   
 $((\exists k < |is| - 2. \text{let } i0 = is!k; i1 = is!(k+1); i2 = is!(k+2) \text{ in}$   
 $(\text{duplicateEdge } g \ f \ (f^{i1} \cdot v) \ (f^{i2} \cdot v)) \wedge (i0 < i1) \wedge (i1 < i2))$   
 $\vee (\text{let } i0 = is!0; i1 = is!1 \text{ in}$   
 $(\text{duplicateEdge } g \ f \ (f^{i0} \cdot v) \ (f^{i1} \cdot v)) \wedge (i0 < i1)))$

**definition**  $\text{generatePolygon} :: \text{nat} \Rightarrow \text{vertex} \Rightarrow \text{face} \Rightarrow \text{graph} \Rightarrow \text{graph list}$  **where**  
 $\text{generatePolygon } n \ v \ f \ g \equiv$   
 $\text{let enumeration} = \text{enumerator } n \ | \text{vertices } f|;$   
 $\text{enumeration} = [is \leftarrow \text{enumeration}. \neg \text{containsDuplicateEdge } g \ f \ v \ is];$   
 $\text{vertexLists} = [\text{indexToVertexList } f \ v \ is. \ is \leftarrow \text{enumeration}] \text{ in}$   
 $[\text{subdivFace } g \ f \ vs. \ vs \leftarrow \text{vertexLists}]$

**definition**  $\text{next-plane0} :: \text{nat} \Rightarrow \text{graph} \Rightarrow \text{graph list}$  ( $\text{next'-plane0}_-$ ) **where**  
 $\text{next-plane0}_p \ g \equiv$   
 $\text{if final } g \text{ then } []$   
 $\text{else } \bigsqcup_{f \in \text{nonFinals } g} \bigsqcup_{v \in \text{vertices } f} \bigsqcup_{i \in [3..< \text{Suc}(\text{maxGon } p)]} \text{generatePolygon } i$   
 $v \ f \ g$

**definition**  $\text{Seed} :: \text{nat} \Rightarrow \text{graph}$  ( $\text{Seed}_-$ ) **where**  
 $\text{Seed}_p \equiv \text{graph}(\text{maxGon } p)$

**lemma**  $\text{Seed-not-final[iff]}: \neg \text{final } (\text{Seed } p)$   
 $\langle \text{proof} \rangle$



**definition** *PlaneGraphs0* :: graph set **where**  
*PlaneGraphs0*  $\equiv \bigcup p. \{g. \text{Seed}_p [\text{next-plane0}_p] \rightarrow^* g \wedge \text{final } g\}$

**end**

**theory** *Plane1*  
**imports** *Plane*  
**begin**

This is an optimized definition of plane graphs and the one we adopt as our point of reference. In every step only one fixed nonfinal face (the smallest one) and one edge in that face are picked.

**definition** *minimalFace* :: face list  $\Rightarrow$  face **where**  
*minimalFace*  $\equiv \text{minimal } (\text{length} \circ \text{vertices})$

**definition** *minimalVertex* :: graph  $\Rightarrow$  face  $\Rightarrow$  vertex **where**  
*minimalVertex*  $g f \equiv \text{minimal } (\text{height } g) (\text{vertices } f)$

**definition** *next-plane* :: nat  $\Rightarrow$  graph  $\Rightarrow$  graph list (*next'-plane*.) **where**  
*next-plane*<sub>*p*</sub>  $g \equiv$   
  let *fs* = *nonFinals* *g* in  
  if *fs* = [] then []  
  else let *f* = *minimalFace* *fs*; *v* = *minimalVertex* *g f* in  
   $\bigsqcup_{i \in [3..< \text{Suc}(\text{maxGon } p)]} \text{generatePolygon } i \ v \ f \ g$

**definition** *PlaneGraphsP* :: nat  $\Rightarrow$  graph set (*PlaneGraphs*.) **where**  
*PlaneGraphs*<sub>*p*</sub>  $\equiv \{g. \text{Seed}_p [\text{next-plane}_p] \rightarrow^* g \wedge \text{final } g\}$

**definition** *PlaneGraphs* :: graph set **where**  
*PlaneGraphs*  $\equiv \bigcup p. \text{PlaneGraphs}_p$

**end**

## 10 Properties of Graph Utilities

**theory** *GraphProps*  
**imports** *Graph*  
**begin**

**declare** [[*linarith-neq-limit* = 3]]

**lemma** *final-setFinal*[*iff*]: *final*(*setFinal* *f*)  
 $\langle \text{proof} \rangle$

**lemma** *eq-setFinal-iff*[*iff*]:  $(f = \text{setFinal } f) = \text{final } f$   
(*proof*)

**lemma** *setFinal-eq-iff*[*iff*]:  $(\text{setFinal } f = f) = \text{final } f$   
(*proof*)

**lemma** *distinct-vertices*[*iff*]:  $\text{distinct}(\text{vertices}(g::\text{graph}))$   
(*proof*)

## 10.1 *nextElem*

**lemma** *nextElem-append*[*simp*]:  
 $y \notin \text{set } xs \implies \text{nextElem } (xs @ ys) d y = \text{nextElem } ys d y$   
(*proof*)

**lemma** *nextElem-cases*:  
 $\text{nextElem } xs d x = y \implies$   
 $x \notin \text{set } xs \wedge y = d \vee$   
 $xs \neq [] \wedge x = \text{last } xs \wedge y = d \wedge x \notin \text{set}(\text{butlast } xs) \vee$   
 $(\exists us \text{ vs. } xs = us @ [x,y] @ vs \wedge x \notin \text{set } us)$   
(*proof*)

**lemma** *nextElem-notin-butlast*[*rule-format,simp*]:  
 $y \notin \text{set}(\text{butlast } xs) \longrightarrow \text{nextElem } xs x y = x$   
(*proof*)

**lemma** *nextElem-in*:  $\text{nextElem } xs x y : \text{set}(x\#xs)$   
(*proof*)

**lemma** *nextElem-notin*[*simp*]:  $a \notin \text{set } as \implies \text{nextElem } as c a = c$   
(*proof*)

**lemma** *nextElem-last*[*simp*]: **assumes** *dist*:  $\text{distinct } xs$   
**shows**  $\text{nextElem } xs c (\text{last } xs) = c$   
(*proof*)

**lemma** *prevElem-nextElem*:  
**assumes** *dist*:  $\text{distinct } xs$  **and** *xs*:  $x : \text{set } xs$   
**shows**  $\text{nextElem } (\text{rev } xs) (\text{last } xs) (\text{nextElem } xs (\text{hd } xs) x) = x$   
(*proof*)

**lemma** *nextElem-prevElem*:  
 $[\text{distinct } xs; x : \text{set } xs] \implies$   
 $\text{nextElem } xs (\text{hd } xs) (\text{nextElem } (\text{rev } xs) (\text{last } xs) x) = x$   
(*proof*)

**lemma** *nextElem-nth*:

$\wedge i. \llbracket \text{distinct } xs; i < \text{length } xs \rrbracket$   
 $\implies \text{nextElem } xs \ z \ (xs!i) = (\text{if } \text{length } xs = i+1 \ \text{then } z \ \text{else } xs!(i+1))$   
*<proof>*

## 10.2 *nextVertex*

**lemma** *nextVertex-in-face'*[simp]:

$\text{vertices } f \neq [] \implies f \cdot v \in \mathcal{V} f$   
*<proof>*

**lemma** *nextVertex-in-face*[simp]:

$v \in \text{set } (\text{vertices } f) \implies f \cdot v \in \mathcal{V} f$   
*<proof>*

**lemma** *nextVertex-prevVertex*[simp]:

$\llbracket \text{distinct}(\text{vertices } f); v \in \mathcal{V} f \rrbracket$   
 $\implies f \cdot (f^{-1} \cdot v) = v$   
*<proof>*

**lemma** *prevVertex-nextVertex*[simp]:

$\llbracket \text{distinct}(\text{vertices } f); v \in \mathcal{V} f \rrbracket$   
 $\implies f^{-1} \cdot (f \cdot v) = v$   
*<proof>*

**lemma** *prevVertex-in-face*[simp]:

$v \in \mathcal{V} f \implies f^{-1} \cdot v \in \mathcal{V} f$   
*<proof>*

**lemma** *nextVertex-nth*:

$\llbracket \text{distinct}(\text{vertices } f); i < |\text{vertices } f| \rrbracket \implies$   
 $f \cdot (\text{vertices } f ! i) = \text{vertices } f ! ((i+1) \bmod |\text{vertices } f|)$   
*<proof>*

## 10.3 $\mathcal{E}$

**lemma** *edges-face-eq*:

$((a,b) \in \mathcal{E} (f::\text{face})) = ((f \cdot a = b) \wedge a \in \mathcal{V} f)$   
*<proof>*

**lemma** *edges-setFinal*[simp]:  $\mathcal{E}(\text{setFinal } f) = \mathcal{E} f$

*<proof>*

**lemma** *in-edges-in-vertices*:

$(x,y) \in \mathcal{E}(f::\text{face}) \implies x \in \mathcal{V} f \wedge y \in \mathcal{V} f$   
*<proof>*

**lemma** *vertices-conv-Union-edges*:

$$\mathcal{V}(f::\text{face}) = \bigcup_{(a,b) \in \mathcal{E} f} \{a\}$$

*<proof>*

**lemma** *nextVertex-in-edges*:  $v \in \mathcal{V} f \implies (v, f \cdot v) \in \text{edges } f$

*<proof>*

**lemma** *prevVertex-in-edges*:

$$\llbracket \text{distinct}(\text{vertices } f); v \in \mathcal{V} f \rrbracket \implies (f^{-1} \cdot v, v) \in \text{edges } f$$

*<proof>*

## 10.4 Triangles

**lemma** *vertices-triangle*:

$$\begin{aligned} |\text{vertices } f| = 3 &\implies a \in \mathcal{V} f \implies \\ \text{distinct } (\text{vertices } f) &\implies \\ \mathcal{V} f &= \{a, f \cdot a, f \cdot (f \cdot a)\} \end{aligned}$$

*<proof>*

**lemma** *tri-next3-id*:

$$\begin{aligned} |\text{vertices } f| = 3 &\implies \text{distinct}(\text{vertices } f) \implies v \in \mathcal{V} f \\ &\implies f \cdot (f \cdot (f \cdot v)) = v \end{aligned}$$

*<proof>*

**lemma** *triangle-nextVertex-prevVertex*:

$$\begin{aligned} |\text{vertices } f| = 3 &\implies a \in \mathcal{V} f \implies \\ \text{distinct } (\text{vertices } f) &\implies \\ f \cdot (f \cdot a) &= f^{-1} \cdot a \end{aligned}$$

*<proof>*

## 10.5 Quadrilaterals

**lemma** *vertices-quad*:

$$\begin{aligned} |\text{vertices } f| = 4 &\implies a \in \mathcal{V} f \implies \\ \text{distinct } (\text{vertices } f) &\implies \\ \mathcal{V} f &= \{a, f \cdot a, f \cdot (f \cdot a), f \cdot (f \cdot (f \cdot a))\} \end{aligned}$$

*<proof>*

**lemma** *quad-next4-id*:

$$\llbracket |\text{vertices } f| = 4; \text{distinct}(\text{vertices } f); v \in \mathcal{V} f \rrbracket \implies f \cdot (f \cdot (f \cdot (f \cdot v))) = v$$

*<proof>*

**lemma** *quad-nextVertex-prevVertex*:

$$|\text{vertices } f| = 4 \implies a \in \mathcal{V} f \implies \text{distinct } (\text{vertices } f) \implies$$

$$f \cdot (f \cdot (f \cdot a)) = f^{-1} \cdot a$$

*<proof>*

**lemma** *len-faces-sum*:  $|faces\ g| = |finals\ g| + |nonFinals\ g|$   
*<proof>*

**lemma** *graph-max-final-ex*:  
 $\exists f \in set\ (finals\ (graph\ n)).\ |vertices\ f| = n$   
*<proof>*

## 10.6 No loops

**lemma** *distinct-no-loop2*:  
 $\llbracket distinct(vertices\ f); v \in \mathcal{V}\ f; u \in \mathcal{V}\ f; u \neq v \rrbracket \implies f \cdot v \neq v$   
*<proof>*

**lemma** *distinct-no-loop1*:  
 $\llbracket distinct(vertices\ f); v \in \mathcal{V}\ f; |vertices\ f| > 1 \rrbracket \implies f \cdot v \neq v$   
*<proof>*

## 10.7 between

**lemma** *between-front[simp]*:  
 $v \notin set\ us \implies between\ (u \# us\ @\ v \# vs)\ u\ v = us$   
*<proof>*

**lemma** *between-back*:  
 $\llbracket v \notin set\ us; u \notin set\ vs; v \neq u \rrbracket \implies between\ (v \# vs\ @\ u \# us)\ u\ v = us$   
*<proof>*

**lemma** *next-between*:  
 $\llbracket distinct(vertices\ f); v \in \mathcal{V}\ f; u \in \mathcal{V}\ f; f \cdot v \neq u \rrbracket$   
 $\implies f \cdot v \in set(between\ (vertices\ f)\ v\ u)$   
*<proof>*

**lemma** *next-between2*:  
 $\llbracket distinct(vertices\ f); v \in \mathcal{V}\ f; u \in \mathcal{V}\ f; u \neq v \rrbracket \implies$   
 $v \in set(between\ (vertices\ f)\ u\ (f \cdot v))$   
*<proof>*

**lemma** *between-next-empty*:  
 $distinct(vertices\ f) \implies between\ (vertices\ f)\ v\ (f \cdot v) = []$   
*<proof>*

**lemma** *unroll-between-next2*:  
 $\llbracket \text{distinct}(\text{vertices } f); u \in \mathcal{V} f; v \in \mathcal{V} f; u \neq v \rrbracket \implies$   
 $\text{between}(\text{vertices } f) u (f \cdot v) = \text{between}(\text{vertices } f) u v @ [v]$   
 $\langle \text{proof} \rangle$

**lemma** *nextVertex-eq-lemma*:  
 $\llbracket \text{distinct}(\text{vertices } f); x \in \mathcal{V} f; y \in \mathcal{V} f; x \neq y;$   
 $v \in \text{set}(x \# \text{between}(\text{vertices } f) x y) \rrbracket \implies$   
 $f \cdot v = \text{nextElem}(x \# \text{between}(\text{vertices } f) x y @ [y]) z v$   
 $\langle \text{proof} \rangle$

**end**

## 11 Properties of Patch Enumeration

**theory** *EnumeratorProps*  
**imports** *Enumerator GraphProps*  
**begin**

**lemma** *length-hideDupsRec[simp]*:  $\bigwedge x. \text{length}(\text{hideDupsRec } x \text{ } xs) = \text{length } xs$   
 $\langle \text{proof} \rangle$

**lemma** *length-hideDups[simp]*:  $\text{length}(\text{hideDups } xs) = \text{length } xs$   
 $\langle \text{proof} \rangle$

**lemma** *length-indexToVertexList[simp]*:  
 $\text{length}(\text{indexToVertexList } x \text{ } y \text{ } xs) = \text{length } xs$   
 $\langle \text{proof} \rangle$

**definition** *increasing* ::  $(\alpha :: \text{linorder}) \text{ list} \Rightarrow \text{bool}$  **where**  
 $\text{increasing } ls \equiv \forall x \ y \ as \ bs. ls = as @ x \# y \# bs \longrightarrow x \leq y$

**lemma** *increasing1*:  $\bigwedge as \ x. \text{increasing } ls \implies ls = as @ x \# cs @ y \# bs \implies x \leq y$   
 $\langle \text{proof} \rangle$

**lemma** *increasing2*:  $\text{increasing}(as @ bs) \implies x \in \text{set } as \implies y \in \text{set } bs \implies x \leq y$   
 $\langle \text{proof} \rangle$

**lemma** *increasing3*:  $\forall as \ bs. (ls = as @ bs \longrightarrow (\forall x \in \text{set } as. \forall y \in \text{set } bs. x \leq y)) \implies \text{increasing}(ls)$   
 $\langle \text{proof} \rangle$

**lemma** *increasing4*:  $\text{increasing } (as@bs) \implies \text{increasing } as$   
*<proof>*

**lemma** *increasing5*:  $\text{increasing } (as@bs) \implies \text{increasing } bs$   
*<proof>*

**lemma** *enumBase-length*:  $ls \in \text{set } (enumBase \ nmax) \implies \text{length } ls = 1$   
*<proof>*

**lemma** *enumBase-bound*:  $\forall y \in \text{set } (enumBase \ nmax). \forall z \in \text{set } y. z \leq nmax$   
*<proof>*

**lemmas** *enumBase-simps* = *enumBase-length enumBase-bound*

**lemma** *enumAppend-bound*:  $ls \in \text{set } ((enumAppend \ nmax) \ lss) \implies$   
 $\forall y \in \text{set } lss. \forall z \in \text{set } y. z \leq nmax \implies x \in \text{set } ls \implies x \leq nmax$   
*<proof>*

**lemma** *enumAppend-bound-rec*:  $ls \in \text{set } (((enumAppend \ nmax) \ \wedge \ n) \ lss) \implies$   
 $\forall y \in \text{set } lss. \forall z \in \text{set } y. z \leq nmax \implies x \in \text{set } ls \implies x \leq nmax$   
*<proof>*

**lemma** *enumAppend-increase-rec*:  
 $\bigwedge m \ as \ bs. ls \in \text{set } (((enumAppend \ nmax) \ \wedge \ m) \ (enumBase \ nmax)) \implies$   
 $as \ @ \ bs = ls \implies \forall x \in \text{set } as. \forall y \in \text{set } bs. x \leq y$   
*<proof>*

**lemma** *enumAppend-length1*:  $\bigwedge ls. ls \in \text{set } ((enumAppend \ nmax \ \wedge \ n) \ lss) \implies$   
 $(\forall l \in \text{set } lss. |l| = k) \implies |ls| = k + n$   
*<proof>*

**lemma** *enumAppend-length2*:  $\bigwedge ls. ls \in \text{set } ((enumAppend \ nmax \ \wedge \ n) \ lss) \implies$   
 $(\bigwedge l. l \in \text{set } lss \implies |l| = k) \implies K = k + n \implies |ls| = K$   
*<proof>*

**lemma** *enum-enumerator*:  
 $enum \ i \ j = \text{enumerator } i \ j$

*<proof>*

**lemma** *enumerator-hd*:  $ls \in \text{set } (\text{enumerator } m \ n) \implies \text{hd } ls = 0$   
*<proof>*

**lemma** *enumerator-last*:  $ls \in \text{set } (\text{enumerator } m \ n) \implies \text{last } ls = (n - 1)$   
*<proof>*

**lemma** *enumerator-length*:  $ls \in \text{set } (\text{enumerator } m \ n) \implies 2 \leq \text{length } ls$   
*<proof>*

**lemmas** *set-enumerator-simps* = *enumerator-hd enumerator-last enumerator-length*

**lemma** *enumerator-not-empty[dest]*:  $ls \in \text{set } (\text{enumerator } m \ n) \implies ls \neq []$   
*<proof>*

**lemma** *enumerator-length2*:  $ls \in \text{set } (\text{enumerator } m \ n) \implies 2 < m \implies \text{length } ls = m$   
*<proof>*

**lemma** *enumerator-bound*:  $ls \in \text{set } (\text{enumerator } m \ nmax) \implies 0 < nmax \implies x \in \text{set } ls \implies x < nmax$   
*<proof>*

**lemma** *enumerator-bound2*:  $ls \in \text{set } (\text{enumerator } m \ nmax) \implies 1 < nmax \implies x \in \text{set } (\text{butlast } ls) \implies x < nmax - \text{Suc } 0$   
*<proof>*

**lemma** *enumerator-bound3*:  $ls \in \text{set } (\text{enumerator } m \ nmax) \implies 1 < nmax \implies \text{last } (\text{butlast } ls) < nmax - \text{Suc } 0$   
*<proof>*

**lemma** *enumerator-increase*:  $\bigwedge as \ bs. ls \in \text{set } (\text{enumerator } m \ nmax) \implies as @ bs = ls \implies \forall x \in \text{set } as. \forall y \in \text{set } bs. x \leq y$   
*<proof>*

**lemma** *enumerator-increasing*:  $ls \in \text{set } (\text{enumerator } m \ nmax) \implies \text{increasing } ls$   
*<proof>*

**definition** *incrIndexList* ::  $\text{nat list} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool}$  **where**  
*incrIndexList*  $ls \ m \ nmax \equiv$   
 $1 < m \wedge 1 < nmax \wedge$   
 $\text{hd } ls = 0 \wedge \text{last } ls = (nmax - 1) \wedge \text{length } ls = m$   
 $\wedge \text{last } (\text{butlast } ls) < \text{last } ls \wedge \text{increasing } ls$



**lemma** *incrIndexList-1lem*[simp]: *incrIndexList ls m nmax*  $\implies$  *Suc 0 < m*  
<proof>

**lemma** *incrIndexList-1len*[simp]: *incrIndexList ls m nmax*  $\implies$  *Suc 0 < nmax*  
<proof>

**lemma** *incrIndexList-help2*[simp]: *incrIndexList ls m nmax*  $\implies$  *hd ls = 0*  
<proof>

**lemma** *incrIndexList-help21*[simp]: *incrIndexList (l # ls) m nmax*  $\implies$  *l = 0*  
<proof>

**lemma** *incrIndexList-help3*[simp]: *incrIndexList ls m nmax*  $\implies$  *last ls = (nmax*  
*- (Suc 0))*  
<proof>

**lemma** *incrIndexList-help4*[simp]: *incrIndexList ls m nmax*  $\implies$  *length ls = m*  
<proof>

**lemma** *incrIndexList-help5*[intro]: *incrIndexList ls m nmax*  $\implies$  *last (butlast ls)*  
*< nmax - Suc 0*  
<proof>

**lemma** *incrIndexList-help6*[simp]: *incrIndexList ls m nmax*  $\implies$  *increasing ls*  
<proof>

**lemma** *incrIndexList-help7*[simp]: *incrIndexList ls m nmax*  $\implies$  *ls  $\neq$  []*  
<proof>

**lemma** *incrIndexList-help71*[simp]:  $\neg$  *incrIndexList [] m nmax*  
<proof>

**lemma** *incrIndexList-help8*[simp]: *incrIndexList ls m nmax*  $\implies$  *butlast ls  $\neq$  []*  
<proof>

**lemma** *incrIndexList-help81*[simp]:  $\neg$  *incrIndexList [l] m nmax*  
<proof>

**lemma** *incrIndexList-help9*[intro]: (*incrIndexList ls m nmax*)  $\implies$   
*x  $\in$  set (butlast ls)  $\implies$  x  $\leq$  nmax - 2*  
<proof>

**lemma** *incrIndexList-help10*[intro]: (*incrIndexList ls m nmax*)  $\implies$   
*x  $\in$  set ls  $\implies$  x < nmax* <proof>

**lemma** *enumerator-correctness*: *2 < m  $\implies$  1 < nmax  $\implies$*   
*ls  $\in$  set (enumerator m nmax)  $\implies$*   
*incrIndexList ls m nmax*

*<proof>*

**lemma** *enumerator-completeness-help*:  $\bigwedge ls. \text{increasing } ls \implies ls \neq [] \implies \text{length } ls = \text{Suc } ks \implies \text{list-all } (\lambda x. x < \text{Suc } nmax) \text{ } ls \implies ls \in \text{set } ((\text{enumAppend } nmax \wedge ks) (\text{enumBase } nmax))$

*<proof>*

**lemma** *enumerator-completeness*:  $2 < m \implies \text{incrIndexList } ls \ m \ nmax \implies ls \in \text{set } (\text{enumerator } m \ nmax)$

*<proof>*

**lemma** *enumerator-equiv[simp]*:

$2 < n \implies 1 < m \implies is \in \text{set}(\text{enumerator } n \ m) = \text{incrIndexList } is \ n \ m$

*<proof>*

**end**

## 12 Properties of Face Division

**theory** *FaceDivisionProps*

**imports** *Plane EnumeratorProps*

**begin**

### 12.1 Finality

**lemma** *vertices-makeFaceFinal*:  $\text{vertices}(\text{makeFaceFinal } f \ g) = \text{vertices } g$

*<proof>*

**lemma** *edges-makeFaceFinal*:  $\mathcal{E} (\text{makeFaceFinal } f \ g) = \mathcal{E} \ g$

*<proof>*

**lemma** *in-set-repl-setFin*:

$f \in \text{set } fs \implies \text{final } f \implies f \in \text{set } (\text{replace } f' \ [\text{setFinal } f] \ fs)$

*<proof>*

**lemma** *in-set-repl*:  $f \in \text{set } fs \implies f \neq f' \implies f \in \text{set } (\text{replace } f' \ fs' \ fs)$

*<proof>*

**lemma** *makeFaceFinals-preserve-finals*:

$f \in \text{set } (\text{finals } g) \implies f \in \text{set } (\text{finals } (\text{makeFaceFinal } f' \ g))$

*<proof>*

**lemma** *len-faces-makeFaceFinal[simp]*:

$|\text{faces } (\text{makeFaceFinal } f \ g)| = |\text{faces } g|$

*<proof>*

**lemma** *len-finals-makeFaceFinal*:

$f \in \mathcal{F} g \implies \neg \text{final } f \implies |\text{finals } (\text{makeFaceFinal } f g)| = |\text{finals } g| + 1$   
 $\langle \text{proof} \rangle$

**lemma** *len-nonFinals-makeFaceFinal*:

$\llbracket \neg \text{final } f; f \in \mathcal{F} g \rrbracket$   
 $\implies |\text{nonFinals } (\text{makeFaceFinal } f g)| = |\text{nonFinals } g| - 1$   
 $\langle \text{proof} \rangle$

**lemma** *set-finals-makeFaceFinal[simp]*:  $\text{distinct}(\text{faces } g) \implies f \in \mathcal{F} g \implies$

$\text{set}(\text{finals } (\text{makeFaceFinal } f g)) = \text{insert } (\text{setFinal } f) (\text{set}(\text{finals } g))$   
 $\langle \text{proof} \rangle$

**lemma** *splitFace-preserve-final*:

$f \in \text{set } (\text{finals } g) \implies \neg \text{final } f' \implies$   
 $f \in \text{set } (\text{finals } (\text{snd } (\text{snd } (\text{splitFace } g i j f' ns))))$   
 $\langle \text{proof} \rangle$

**lemma** *splitFace-nonFinal-face*:

$\neg \text{final } (\text{fst } (\text{snd } (\text{splitFace } g i j f' ns)))$   
 $\langle \text{proof} \rangle$

**lemma** *subdivFace'-preserve-finals*:

$\bigwedge n i f' g. f \in \text{set } (\text{finals } g) \implies \neg \text{final } f' \implies$   
 $f \in \text{set } (\text{finals } (\text{subdivFace}' g f' i n is))$   
 $\langle \text{proof} \rangle$

**lemma** *subdivFace-pres-finals*:

$f \in \text{set } (\text{finals } g) \implies \neg \text{final } f' \implies$   
 $f \in \text{set } (\text{finals } (\text{subdivFace } g f' is))$   
 $\langle \text{proof} \rangle$

**declare** *Nat.diff-is-0-eq'* [simp del]

## 12.2 *is-prefix*

**definition** *is-prefix* :: 'a list  $\Rightarrow$  'a list  $\Rightarrow$  bool **where**

*is-prefix*  $ls\ vs \equiv (\exists bs. vs = ls @ bs)$

**lemma** *is-prefix-add*:

$\text{is-prefix } ls\ vs \implies \text{is-prefix } (as @ ls) (as @ vs) \langle \text{proof} \rangle$

**lemma** *is-prefix-hd[simp]*:

$\text{is-prefix } [l]\ vs = (l = \text{hd } vs \wedge vs \neq [])$

$\langle proof \rangle$

**lemma** *is-prefix-f[simp]*:

$is\_prefix (a\#as) (a\#vs) = is\_prefix as vs \langle proof \rangle$

**lemma** *splitAt-is-prefix*:  $ram \in set\ vs \implies is\_prefix (fst (splitAt\ ram\ vs) @ [ram])$

$vs$

$\langle proof \rangle$

### 12.3 *is-sublist*

**definition** *is-sublist* ::  $'a\ list \Rightarrow 'a\ list \Rightarrow bool$  **where**

$is\_sublist\ ls\ vs \equiv (\exists\ as\ bs.\ vs = as @ ls @ bs)$

**lemma** *is-prefix-sublist*:

$is\_prefix\ ls\ vs \implies is\_sublist\ ls\ vs \langle proof \rangle$

**lemma** *is-sublist-trans*:  $is\_sublist\ as\ bs \implies is\_sublist\ bs\ cs \implies is\_sublist\ as\ cs$

$\langle proof \rangle$

**lemma** *is-sublist-add*:  $is\_sublist\ as\ bs \implies is\_sublist\ as\ (xs @ bs @ ys)$

$\langle proof \rangle$

**lemma** *is-sublist-rec*:

$is\_sublist\ xs\ ys =$

$(if\ length\ xs > length\ ys\ then\ False\ else$

$if\ xs = take\ (length\ xs)\ ys\ then\ True\ else\ is\_sublist\ xs\ (tl\ ys))$

$\langle proof \rangle$

**lemma** *not-sublist-len[simp]*:

$|ys| < |xs| \implies \neg is\_sublist\ xs\ ys$

$\langle proof \rangle$

**lemma** *is-sublist-simp[simp]*:  $a \neq v \implies is\_sublist (a\#as) (v\#vs) = is\_sublist$

$(a\#as)\ vs$

$\langle proof \rangle$

**lemma** *is-sublist-id[simp]*:  $is\_sublist\ vs\ vs \langle proof \rangle$

**lemma** *is-sublist-in*:  $is\_sublist (a\#as)\ vs \implies a \in set\ vs \langle proof \rangle$

**lemma** *is-sublist-in1*:  $is\_sublist [x,y]\ vs \implies y \in set\ vs \langle proof \rangle$

**lemma** *is-sublist-notlast[simp]*:  $distinct\ vs \implies x = last\ vs \implies \neg is\_sublist [x,y]$

$vs$

$\langle proof \rangle$

**lemma** *is-sublist-nth1*:  $is\_sublist\ [x,y]\ ls \implies$   
 $\exists\ i\ j.\ i < length\ ls \wedge j < length\ ls \wedge ls!i = x \wedge ls!j = y \wedge Suc\ i = j$   
 $\langle proof \rangle$

**lemma** *is-sublist-nth2*:  $\exists\ i\ j.\ i < length\ ls \wedge j < length\ ls \wedge ls!i = x \wedge ls!j = y$   
 $\wedge\ Suc\ i = j \implies$   
 $is\_sublist\ [x,y]\ ls$   
 $\langle proof \rangle$

**lemma** *is-sublist-tl*:  $is\_sublist\ (a \# as)\ vs \implies is\_sublist\ as\ vs \langle proof \rangle$

**lemma** *is-sublist-hd*:  $is\_sublist\ (a \# as)\ vs \implies is\_sublist\ [a]\ vs \langle proof \rangle$

**lemma** *is-sublist-hd-eq[simp]*:  $(is\_sublist\ [a]\ vs) = (a \in set\ vs) \langle proof \rangle$

**lemma** *is-sublist-distinct-prefix*:  
 $is\_sublist\ (v \# as)\ (v \# vs) \implies distinct\ (v \# vs) \implies is\_prefix\ as\ vs$   
 $\langle proof \rangle$

**lemma** *is-sublist-distinct[intro]*:  
 $is\_sublist\ as\ vs \implies distinct\ vs \implies distinct\ as \langle proof \rangle$

**lemma** *is-sublist-y-hd*:  $distinct\ vs \implies y = hd\ vs \implies \neg is\_sublist\ [x,y]\ vs$   
 $\langle proof \rangle$

**lemma** *is-sublist-at1*:  $distinct\ (as\ @\ bs) \implies is\_sublist\ [x,y]\ (as\ @\ bs) \implies x \neq$   
 $(last\ as) \implies$   
 $is\_sublist\ [x,y]\ as \vee is\_sublist\ [x,y]\ bs$   
 $\langle proof \rangle$

**lemma** *is-sublist-at4*:  $distinct\ (as\ @\ bs) \implies is\_sublist\ [x,y]\ (as\ @\ bs) \implies$   
 $as \neq [] \implies x = last\ as \implies y = hd\ bs$   
 $\langle proof \rangle$

**lemma** *is-sublist-at5*:  $distinct\ (as\ @\ bs) \implies is\_sublist\ [x,y]\ (as\ @\ bs) \implies$   
 $is\_sublist\ [x,y]\ as \vee is\_sublist\ [x,y]\ bs \vee x = last\ as \wedge y = hd\ bs$   
 $\langle proof \rangle$

**lemma** *is-sublist-rev*:  $is\_sublist\ [a,b]\ (rev\ zs) = is\_sublist\ [b,a]\ zs$   
 $\langle proof \rangle$

**lemma** *is-sublist-at5'[simp]*:  
 $distinct\ as \implies distinct\ bs \implies set\ as \cap set\ bs = \{\} \implies is\_sublist\ [x,y]\ (as\ @\ bs)$   
 $\implies$   
 $is\_sublist\ [x,y]\ as \vee is\_sublist\ [x,y]\ bs \vee x = last\ as \wedge y = hd\ bs$   
 $\langle proof \rangle$

**lemma** *splitAt-is-sublist1R[simp]*:  $ram \in set\ vs \implies is\_sublist\ (fst\ (splitAt\ ram\ vs))$   
 $@\ [ram]\ vs$

*<proof>*

**lemma** *splitAt-is-sublist2R[simp]*:  $ram \in set\ vs \implies is\_sublist\ (ram\ \# \ snd\ (splitAt\ ram\ vs))\ vs$   
*<proof>*

## 12.4 *is-nextElem*

**definition** *is-nextElem* :: 'a list  $\Rightarrow$  'a  $\Rightarrow$  'a  $\Rightarrow$  bool **where**  
 $is\_nextElem\ xs\ x\ y \equiv is\_sublist\ [x,y]\ xs \vee xs \neq [] \wedge x = last\ xs \wedge y = hd\ xs$

**lemma** *is-nextElem-a[intro]*:  $is\_nextElem\ vs\ a\ b \implies a \in set\ vs$   
*<proof>*

**lemma** *is-nextElem-b[intro]*:  $is\_nextElem\ vs\ a\ b \implies b \in set\ vs$   
*<proof>*

**lemma** *is-nextElem-last-hd[intro]*:  $distinct\ vs \implies is\_nextElem\ vs\ x\ y \implies x = last\ vs \implies y = hd\ vs$   
*<proof>*

**lemma** *is-nextElem-last-ne[intro]*:  $distinct\ vs \implies is\_nextElem\ vs\ x\ y \implies x = last\ vs \implies vs \neq []$   
*<proof>*

**lemma** *is-nextElem-sublistI*:  $is\_sublist\ [x,y]\ vs \implies is\_nextElem\ vs\ x\ y$   
*<proof>*

**lemma** *is-nextElem-nth1*:  $is\_nextElem\ ls\ x\ y \implies \exists\ i\ j. i < length\ ls \wedge j < length\ ls \wedge ls!i = x \wedge ls!j = y \wedge (Suc\ i) \bmod (length\ ls) = j$   
*<proof>*

**lemma** *is-nextElem-nth2*:  $\exists\ i\ j. i < length\ ls \wedge j < length\ ls \wedge ls!i = x \wedge ls!j = y \wedge (Suc\ i) \bmod (length\ ls) = j \implies is\_nextElem\ ls\ x\ y$   
*<proof>*

**lemma** *is-nextElem-rotate1-aux*:  
 $is\_nextElem\ (rotate\ m\ ls)\ x\ y \implies is\_nextElem\ ls\ x\ y$   
*<proof>*

**lemma** *is-nextElem-rotate-eq[simp]*:  $is\_nextElem\ (rotate\ m\ ls)\ x\ y = is\_nextElem\ ls\ x\ y$   
*<proof>*

**lemma** *is-nextElem-congs-eq*:  $ls \cong ms \implies is\_nextElem\ ls\ x\ y = is\_nextElem\ ms\ x\ y$   
*<proof>*

**lemma** *is-nextElem-rev[simp]*:  $is\_nextElem\ (rev\ zs)\ a\ b = is\_nextElem\ zs\ b\ a$   
*<proof>*

**lemma** *is-nextElem-circ*:

$\llbracket \text{distinct } xs; \text{is-nextElem } xs \ a \ b; \text{is-nextElem } xs \ b \ a \rrbracket \implies |xs| \leq 2$   
 ⟨proof⟩

## 12.5 *nextElem, sublist, is-nextElem*

**lemma** *is-sublist-eq*:  $\text{distinct } vs \implies c \neq y \implies$

$(\text{nextElem } vs \ c \ x = y) = \text{is-sublist } [x,y] \ vs$   
 ⟨proof⟩

**lemma** *is-nextElem1*:  $\text{distinct } vs \implies x \in \text{set } vs \implies \text{nextElem } vs \ (\text{hd } vs) \ x = y$   
 $\implies \text{is-nextElem } vs \ x \ y$

⟨proof⟩

**lemma** *is-nextElem2*:  $\text{distinct } vs \implies x \in \text{set } vs \implies \text{is-nextElem } vs \ x \ y \implies \text{nextElem } vs \ (\text{hd } vs) \ x = y$

⟨proof⟩

**lemma** *nextElem-is-nextElem*:

$\text{distinct } xs \implies x \in \text{set } xs \implies$   
 $\text{is-nextElem } xs \ x \ y = (\text{nextElem } xs \ (\text{hd } xs) \ x = y)$   
 ⟨proof⟩

**lemma** *nextElem-congs-eq*:  $xs \cong ys \implies \text{distinct } xs \implies x \in \text{set } xs \implies$   
 $\text{nextElem } xs \ (\text{hd } xs) \ x = \text{nextElem } ys \ (\text{hd } ys) \ x$

⟨proof⟩

**lemma** *is-sublist-is-nextElem*:  $\text{distinct } vs \implies \text{is-nextElem } vs \ x \ y \implies \text{is-sublist } as$   
 $vs \implies x \in \text{set } as \implies x \neq \text{last } as \implies \text{is-sublist } [x,y] \ as$

⟨proof⟩

## 12.6 *before*

**definition** *before* :: 'a list  $\Rightarrow$  'a  $\Rightarrow$  'a  $\Rightarrow$  bool **where**

$\text{before } vs \ \text{ram1 } \text{ram2} \equiv \exists \ a \ b \ c. \ vs = a \ @ \ \text{ram1} \ \# \ b \ @ \ \text{ram2} \ \# \ c$

**lemma** *before-dist-fst-fst[simp]*:  $\text{before } vs \ \text{ram1 } \text{ram2} \implies \text{distinct } vs \implies \text{fst } (\text{splitAt } \text{ram2} \ (\text{fst } (\text{splitAt } \text{ram1} \ vs))) = \text{fst } (\text{splitAt } \text{ram1} \ (\text{fst } (\text{splitAt } \text{ram2} \ vs)))$

⟨proof⟩

**lemma** *before-dist-fst-snd[simp]*:  $\text{before } vs \ \text{ram1 } \text{ram2} \implies \text{distinct } vs \implies \text{fst } (\text{splitAt } \text{ram2} \ (\text{snd } (\text{splitAt } \text{ram1} \ vs))) = \text{snd } (\text{splitAt } \text{ram1} \ (\text{fst } (\text{splitAt } \text{ram2} \ vs)))$

⟨proof⟩

**lemma** *before-dist-snd-fst[simp]*:  $\text{before } vs \ \text{ram1 } \text{ram2} \implies \text{distinct } vs \implies \text{snd } (\text{splitAt } \text{ram2} \ (\text{fst } (\text{splitAt } \text{ram1} \ vs))) = \text{snd } (\text{splitAt } \text{ram1} \ (\text{snd } (\text{splitAt } \text{ram2} \ vs)))$

⟨proof⟩

⟨proof⟩

**lemma** *before-dist-snd-snd[simp]*:  $\text{before } vs \text{ ram1 ram2} \implies \text{distinct } vs \implies \text{snd} (\text{splitAt ram2} (\text{snd} (\text{splitAt ram1 } vs))) = \text{fst} (\text{splitAt ram1} (\text{snd} (\text{splitAt ram2 } vs)))$   
 ⟨proof⟩

**lemma** *before-dist-snd[simp]*:  $\text{before } vs \text{ ram1 ram2} \implies \text{distinct } vs \implies \text{fst} (\text{splitAt ram1} (\text{snd} (\text{splitAt ram2 } vs))) = \text{snd} (\text{splitAt ram2 } vs)$   
 ⟨proof⟩

**lemma** *before-dist-fst[simp]*:  $\text{before } vs \text{ ram1 ram2} \implies \text{distinct } vs \implies \text{fst} (\text{splitAt ram1} (\text{fst} (\text{splitAt ram2 } vs))) = \text{fst} (\text{splitAt ram1 } vs)$   
 ⟨proof⟩

**lemma** *before-or*:  $\text{ram1} \in \text{set } vs \implies \text{ram2} \in \text{set } vs \implies \text{ram1} \neq \text{ram2} \implies \text{before } vs \text{ ram1 ram2} \vee \text{before } vs \text{ ram2 ram1}$   
 ⟨proof⟩

**lemma** *before-r1*:  
 $\text{before } vs \text{ r1 r2} \implies \text{r1} \in \text{set } vs$  ⟨proof⟩

**lemma** *before-r2*:  
 $\text{before } vs \text{ r1 r2} \implies \text{r2} \in \text{set } vs$  ⟨proof⟩

**lemma** *before-dist-r2*:  
 $\text{distinct } vs \implies \text{before } vs \text{ r1 r2} \implies \text{r2} \in \text{set} (\text{snd} (\text{splitAt r1 } vs))$   
 ⟨proof⟩

**lemma** *before-dist-not-r2[intro]*:  
 $\text{distinct } vs \implies \text{before } vs \text{ r1 r2} \implies \text{r2} \notin \text{set} (\text{fst} (\text{splitAt r1 } vs))$  ⟨proof⟩

**lemma** *before-dist-r1*:  
 $\text{distinct } vs \implies \text{before } vs \text{ r1 r2} \implies \text{r1} \in \text{set} (\text{fst} (\text{splitAt r2 } vs))$   
 ⟨proof⟩

**lemma** *before-dist-not-r1[intro]*:  
 $\text{distinct } vs \implies \text{before } vs \text{ r1 r2} \implies \text{r1} \notin \text{set} (\text{snd} (\text{splitAt r2 } vs))$  ⟨proof⟩

**lemma** *before-snd*:  
 $\text{r2} \in \text{set} (\text{snd} (\text{splitAt r1 } vs)) \implies \text{before } vs \text{ r1 r2}$   
 ⟨proof⟩

**lemma** *before-fst*:  
 $\text{r2} \in \text{set } vs \implies \text{r1} \in \text{set} (\text{fst} (\text{splitAt r2 } vs)) \implies \text{before } vs \text{ r1 r2}$   
 ⟨proof⟩

**lemma** *before-dist-eq-fst*:  
 $\text{distinct } vs \implies \text{r2} \in \text{set } vs \implies \text{r1} \in \text{set} (\text{fst} (\text{splitAt r2 } vs)) = \text{before } vs \text{ r1 r2}$   
 ⟨proof⟩



**lemma** *before-dist-eq-snd*:

$distinct\ vs \implies r2 \in set\ (snd\ (splitAt\ r1\ vs)) = before\ vs\ r1\ r2$   
 $\langle proof \rangle$

**lemma** *before-dist-not1*:

$distinct\ vs \implies before\ vs\ ram1\ ram2 \implies \neg\ before\ vs\ ram2\ ram1$   
 $\langle proof \rangle$

**lemma** *before-dist-not2*:

$distinct\ vs \implies ram1 \in set\ vs \implies ram2 \in set\ vs \implies ram1 \neq ram2 \implies \neg\ (before\ vs\ ram1\ ram2) \implies before\ vs\ ram2\ ram1$   
 $\langle proof \rangle$

**lemma** *before-dist-eq*:

$distinct\ vs \implies ram1 \in set\ vs \implies ram2 \in set\ vs \implies ram1 \neq ram2 \implies (\neg\ (before\ vs\ ram1\ ram2)) = before\ vs\ ram2\ ram1$   
 $\langle proof \rangle$

**lemma** *before-vs*:

$distinct\ vs \implies before\ vs\ ram1\ ram2 \implies vs = fst\ (splitAt\ ram1\ vs) @ ram1 \# fst\ (splitAt\ ram2\ (snd\ (splitAt\ ram1\ vs))) @ ram2 \# snd\ (splitAt\ ram2\ vs)$   
 $\langle proof \rangle$

## 12.7 *between*

**definition** *pre-between* :: 'a list  $\Rightarrow$  'a  $\Rightarrow$  'a  $\Rightarrow$  bool **where**

$pre-between\ vs\ ram1\ ram2 \equiv$   
 $distinct\ vs \wedge ram1 \in set\ vs \wedge ram2 \in set\ vs \wedge ram1 \neq ram2$

**declare** *pre-between-def* [simp]

**lemma** *pre-between-dist*[intro]:

$pre-between\ vs\ ram1\ ram2 \implies distinct\ vs\ \langle proof \rangle$

**lemma** *pre-between-r1*[intro]:

$pre-between\ vs\ ram1\ ram2 \implies ram1 \in set\ vs\ \langle proof \rangle$

**lemma** *pre-between-r2*[intro]:

$pre-between\ vs\ ram1\ ram2 \implies ram2 \in set\ vs\ \langle proof \rangle$

**lemma** *pre-between-r12*[intro]:

$pre-between\ vs\ ram1\ ram2 \implies ram1 \neq ram2\ \langle proof \rangle$

**lemma** *pre-between-sym1*:

$pre-between\ vs\ ram1\ ram2 \implies pre-between\ vs\ ram2\ ram1\ \langle proof \rangle$

**lemma** *pre-between-before*[dest]:

$pre-between\ vs\ ram1\ ram2 \implies before\ vs\ ram1\ ram2 \vee before\ vs\ ram2\ ram1\ \langle proof \rangle$

**lemma** *pre-between-rotate1*[*intro*]:

*pre-between vs ram1 ram2*  $\implies$  *pre-between (rotate1 vs) ram1 ram2*  $\langle$ *proof* $\rangle$

**lemma** *pre-between-rotate*[*intro*]:

*pre-between vs ram1 ram2*  $\implies$  *pre-between (rotate n vs) ram1 ram2*  $\langle$ *proof* $\rangle$

**lemma** *pre-between vs ram1 ram2*  $\implies$   $(\neg$  *before vs ram1 ram2*) = *before vs ram2 ram1*

$\langle$ *proof* $\rangle$

**declare** *pre-between-def* [*simp del*]

**lemma** *between-simp1*[*simp*]:

*before vs ram1 ram2*  $\implies$  *pre-between vs ram1 ram2*  $\implies$   
*between vs ram1 ram2* = *fst (splitAt ram2 (snd (splitAt ram1 vs)))*  
 $\langle$ *proof* $\rangle$

**lemma** *between-simp2*[*simp*]:

*before vs ram1 ram2*  $\implies$  *pre-between vs ram1 ram2*  $\implies$   
*between vs ram2 ram1* = *snd (splitAt ram2 vs) @ fst (splitAt ram1 vs)*  
 $\langle$ *proof* $\rangle$

**lemma** *between-not-r1*[*intro*]:

*distinct vs*  $\implies$  *ram1*  $\notin$  *set (between vs ram1 ram2)*  
 $\langle$ *proof* $\rangle$

**lemma** *between-not-r2*[*intro*]:

*distinct vs*  $\implies$  *ram2*  $\notin$  *set (between vs ram1 ram2)*  
 $\langle$ *proof* $\rangle$

**lemma** *between-distinct*[*intro*]:

*distinct vs*  $\implies$  *distinct (between vs ram1 ram2)*  
 $\langle$ *proof* $\rangle$

**lemma** *between-distinct-r12*:

*distinct vs*  $\implies$  *ram1*  $\neq$  *ram2*  $\implies$  *distinct (ram1 # between vs ram1 ram2 @ [ram2])*  $\langle$ *proof* $\rangle$

**lemma** *between-vs*:

*before vs ram1 ram2*  $\implies$  *pre-between vs ram1 ram2*  $\implies$   
*vs* = *fst (splitAt ram1 vs) @ ram1 # (between vs ram1 ram2) @ ram2 # snd (splitAt ram2 vs)*  
 $\langle$ *proof* $\rangle$

**lemma** *between-in*:

*before vs ram1 ram2*  $\implies$  *pre-between vs ram1 ram2*  $\implies$   $x \in$  *set vs*  $\implies$   $x =$  *ram1*  
 $\vee x \in$  *set (between vs ram1 ram2)*  $\vee x =$  *ram2*  $\vee x \in$  *set (between vs ram2 ram1)*

*<proof>*

**lemma**

*before vs ram1 ram2  $\implies$  pre-between vs ram1 ram2  $\implies$   
hd vs  $\neq$  ram1  $\implies$  (a,b) = splitAt (hd vs) (between vs ram2 ram1)  $\implies$   
vs = [hd vs] @ b @ [ram1] @ (between vs ram1 ram2) @ [ram2] @ a*  
*<proof>*

**lemma** *between-congs: pre-between vs ram1 ram2  $\implies$  vs  $\cong$  vs'  $\implies$  between vs  
ram1 ram2 = between vs' ram1 ram2*  
*<proof>*

**lemma** *between-inter-empty:*

*pre-between vs ram1 ram2  $\implies$   
set (between vs ram1 ram2)  $\cap$  set (between vs ram2 ram1) = {}*  
*<proof>*

### 12.7.1 between is-nextElem

**lemma** *is-nextElem-or1: pre-between vs ram1 ram2  $\implies$*

*is-nextElem vs x y  $\implies$  before vs ram1 ram2  $\implies$   
is-sublist [x,y] (ram1 # between vs ram1 ram2 @ [ram2])  
 $\vee$  is-sublist [x,y] (ram2 # between vs ram2 ram1 @ [ram1])*  
*<proof>*

**lemma** *is-nextElem-or: pre-between vs ram1 ram2  $\implies$  is-nextElem vs x y  $\implies$*

*is-sublist [x,y] (ram1 # between vs ram1 ram2 @ [ram2])  $\vee$  is-sublist [x,y] (ram2  
# between vs ram2 ram1 @ [ram1])*  
*<proof>*

**lemma** *pre-between vs ram1 ram2  $\implies$*

*before vs ram2 ram1  $\implies$   
 $\exists$  as bs cs. between vs ram1 ram2 = cs @ as  $\wedge$  vs = as @ [ram2] @ bs @ [ram1]*  
*@ cs*  
*<proof>*

**lemma** *is-sublist-same-len[simp]:*

*length xs = length ys  $\implies$  is-sublist xs ys = (xs = ys)*  
*<proof>*

**lemma** *is-nextElem-between-empty[simp]:*

*distinct vs  $\implies$  is-nextElem vs a b  $\implies$  between vs a b = []*  
*<proof>*

**lemma** *is-nextElem-between-empty': between vs a b = []  $\implies$  distinct vs  $\implies$  a  $\in$*

$set\ vs \implies b \in set\ vs \implies$   
 $a \neq b \implies is\_nextElem\ vs\ a\ b$   
 <proof>

**lemma** *between-nextElem*:  $pre\_between\ vs\ u\ v \implies$   
 $between\ vs\ u\ (nextElem\ vs\ (hd\ vs)\ v) = between\ vs\ u\ v\ @\ [v]$   
 <proof>

**lemma** *nextVertices-in-face[simp]*:  $v \in \mathcal{V}\ f \implies f^n \cdot v \in \mathcal{V}\ f$   
 <proof>

### 12.7.2 *is-nextElem* edges equivalence

**lemma** *is-nextElem-edges1*:  $distinct\ (vertices\ f) \implies (a,b) \in edges\ (f::face) \implies$   
 $is\_nextElem\ (vertices\ f)\ a\ b$  <proof>

**lemma** *is-nextElem-edges2*:  
 $distinct\ (vertices\ f) \implies is\_nextElem\ (vertices\ f)\ a\ b \implies$   
 $(a,b) \in edges\ (f::face)$   
 <proof>

**lemma** *is-nextElem-edges-eq[simp]*:  
 $distinct\ (vertices\ (f::face)) \implies$   
 $(a,b) \in edges\ f = is\_nextElem\ (vertices\ f)\ a\ b$   
 <proof>

### 12.7.3 *nextVertex*

**lemma** *nextElem-suc2*:  $distinct\ (vertices\ f) \implies last\ (vertices\ f) = v \implies v \in set$   
 $(vertices\ f) \implies f \cdot v = hd\ (vertices\ f)$   
 <proof>

## 12.8 *split-face*

**definition** *pre-split-face* ::  $face \Rightarrow nat \Rightarrow nat \Rightarrow nat\ list \Rightarrow bool$  **where**  
 $pre\_split\_face\ oldF\ ram1\ ram2\ newVertexList \equiv$   
 $distinct\ (vertices\ oldF) \wedge distinct\ (newVertexList)$   
 $\wedge \mathcal{V}\ oldF \cap set\ newVertexList = \{\}$   
 $\wedge ram1 \in \mathcal{V}\ oldF \wedge ram2 \in \mathcal{V}\ oldF \wedge ram1 \neq ram2$

**declare** *pre-split-face-def* [simp]

**lemma** *pre-split-face-p-between[intro]*:

$pre-split-face\ oldF\ ram1\ ram2\ newVertexList \implies pre-between\ (vertices\ oldF)\ ram1\ ram2$   $\langle proof \rangle$

**lemma** *pre-split-face-sym1*:

$pre-split-face\ oldF\ ram1\ ram2\ newVertexList \implies pre-split-face\ oldF\ ram2\ ram1\ newVertexList$   $\langle proof \rangle$

**lemma** *pre-split-face-rev*[intro]:

$pre-split-face\ oldF\ ram1\ ram2\ newVertexList \implies pre-split-face\ oldF\ ram1\ ram2\ (rev\ newVertexList)$   $\langle proof \rangle$

**lemma** *split-face-distinct1*:

$(f12, f21) = split-face\ oldF\ ram1\ ram2\ newVertexList \implies pre-split-face\ oldF\ ram1\ ram2\ newVertexList \implies distinct\ (vertices\ f12)$   $\langle proof \rangle$

**lemma** *split-face-distinct1'*[intro]:

$pre-split-face\ oldF\ ram1\ ram2\ newVertexList \implies distinct\ (vertices\ (fst(split-face\ oldF\ ram1\ ram2\ newVertexList)))$   $\langle proof \rangle$

**lemma** *split-face-distinct2*:

$(f12, f21) = split-face\ oldF\ ram1\ ram2\ newVertexList \implies pre-split-face\ oldF\ ram1\ ram2\ newVertexList \implies distinct\ (vertices\ f21)$   $\langle proof \rangle$

**lemma** *split-face-distinct2'*[intro]:

$pre-split-face\ oldF\ ram1\ ram2\ newVertexList \implies distinct\ (vertices\ (snd(split-face\ oldF\ ram1\ ram2\ newVertexList)))$   $\langle proof \rangle$

**declare** *pre-split-face-def* [simp del]

**lemma** *split-face-edges-or*:  $(f12, f21) = split-face\ oldF\ ram1\ ram2\ newVertexList \implies pre-split-face\ oldF\ ram1\ ram2\ newVertexList \implies (a, b) \in edges\ oldF \implies (a, b) \in edges\ f12 \vee (a, b) \in edges\ f21$   $\langle proof \rangle$

## 12.9 verticesFrom

**definition** *verticesFrom* :: *face*  $\Rightarrow$  *vertex*  $\Rightarrow$  *vertex list* **where**  
*verticesFrom* *f*  $\equiv rotate-to\ (vertices\ f)$

**lemmas** *verticesFrom-Def* = *verticesFrom-def* *rotate-to-def*

**lemma** *len-vFrom*[simp]:

$v \in \mathcal{V} f \implies |\text{verticesFrom } f \ v| = |\text{vertices } f|$   
 $\langle \text{proof} \rangle$

**lemma** *verticesFrom-empty[simp]*:  
 $v \in \mathcal{V} f \implies (\text{verticesFrom } f \ v = []) = (\text{vertices } f = [])$   
 $\langle \text{proof} \rangle$

**lemma** *verticesFrom-congs*:  
 $v \in \mathcal{V} f \implies (\text{vertices } f) \cong (\text{verticesFrom } f \ v)$   
 $\langle \text{proof} \rangle$

**lemma** *verticesFrom-eq-if-vertices-cong*:  
 $\llbracket \text{distinct}(\text{vertices } f); \text{distinct}(\text{vertices } f'); x \in \mathcal{V} f \rrbracket \implies$   
 $\text{verticesFrom } f \ x = \text{verticesFrom } f' \ x$   
 $\langle \text{proof} \rangle$

**lemma** *verticesFrom-in[intro]*:  $v \in \mathcal{V} f \implies a \in \mathcal{V} f \implies a \in \text{set } (\text{verticesFrom } f \ v)$   
 $\langle \text{proof} \rangle$

**lemma** *verticesFrom-in'*:  $a \in \text{set } (\text{verticesFrom } f \ v) \implies a \neq v \implies a \in \mathcal{V} f$   
 $\langle \text{proof} \rangle$

**lemma** *set-verticesFrom*:  
 $v \in \mathcal{V} f \implies \text{set } (\text{verticesFrom } f \ v) = \mathcal{V} f$   
 $\langle \text{proof} \rangle$

**lemma** *verticesFrom-hd*:  $\text{hd } (\text{verticesFrom } f \ v) = v \ \langle \text{proof} \rangle$

**lemma** *verticesFrom-distinct[simp]*:  $\text{distinct } (\text{vertices } f) \implies v \in \mathcal{V} f \implies \text{distinct } (\text{verticesFrom } f \ v) \ \langle \text{proof} \rangle$

**lemma** *verticesFrom-nextElem-eq*:  
 $\text{distinct } (\text{vertices } f) \implies v \in \mathcal{V} f \implies u \in \mathcal{V} f \implies$   
 $\text{nextElem } (\text{verticesFrom } f \ v) (\text{hd } (\text{verticesFrom } f \ v)) \ u$   
 $= \text{nextElem } (\text{vertices } f) (\text{hd } (\text{vertices } f)) \ u \ \langle \text{proof} \rangle$

**lemma** *nextElem-vFrom-suc1*:  $\text{distinct } (\text{vertices } f) \implies v \in \mathcal{V} f \implies i < \text{length } (\text{vertices } f) \implies \text{last } (\text{verticesFrom } f \ v) \neq u \implies (\text{verticesFrom } f \ v) ! i = u \implies f \cdot u = (\text{verticesFrom } f \ v) ! (\text{Suc } i)$   
 $\langle \text{proof} \rangle$

**lemma** *verticesFrom-nth*:  $\text{distinct } (\text{vertices } f) \implies d < \text{length } (\text{vertices } f) \implies v \in \mathcal{V} f \implies (\text{verticesFrom } f \ v) ! d = f^d \cdot v$   
 $\langle \text{proof} \rangle$

**lemma** *verticesFrom-length*:  $\text{distinct}(\text{vertices } f) \implies v \in \text{set}(\text{vertices } f) \implies \text{length}(\text{verticesFrom } f v) = \text{length}(\text{vertices } f)$   
 <proof>

**lemma** *verticesFrom-between*:  $v' \in \mathcal{V} f \implies \text{pre-between}(\text{vertices } f) u v \implies \text{between}(\text{vertices } f) u v = \text{between}(\text{verticesFrom } f v') u v$   
 <proof>

**lemma** *verticesFrom-is-nextElem*:  $v \in \mathcal{V} f \implies \text{is-nextElem}(\text{vertices } f) a b = \text{is-nextElem}(\text{verticesFrom } f v) a b$   
 <proof>

**lemma** *verticesFrom-is-nextElem-last*:  $v' \in \mathcal{V} f \implies \text{distinct}(\text{vertices } f) \implies \text{is-nextElem}(\text{verticesFrom } f v') (\text{last}(\text{verticesFrom } f v')) v \implies v = v'$   
 <proof>

**lemma** *verticesFrom-is-nextElem-hd*:  $v' \in \mathcal{V} f \implies \text{distinct}(\text{vertices } f) \implies \text{is-nextElem}(\text{verticesFrom } f v') u v' \implies u = \text{last}(\text{verticesFrom } f v')$   
 <proof>

**lemma** *verticesFrom-pres-nodes1*:  $v \in \mathcal{V} f \implies \mathcal{V} f = \text{set}(\text{verticesFrom } f v)$   
 <proof>

**lemma** *verticesFrom-pres-nodes*:  $v \in \mathcal{V} f \implies w \in \mathcal{V} f \implies w \in \text{set}(\text{verticesFrom } f v)$   
 <proof>

**lemma** *before-verticesFrom*:  $\text{distinct}(\text{vertices } f) \implies v \in \mathcal{V} f \implies w \in \mathcal{V} f \implies v \neq w \implies \text{before}(\text{verticesFrom } f v) v w$   
 <proof>

**lemma** *last-vFrom*:  
 $\llbracket \text{distinct}(\text{vertices } f); x \in \mathcal{V} f \rrbracket \implies \text{last}(\text{verticesFrom } f x) = f^{-1} \cdot x$   
 <proof>

**lemma** *rotate-before-vFrom*:  
 $\llbracket \text{distinct}(\text{vertices } f); r \in \mathcal{V} f; r \neq u \rrbracket \implies \text{before}(\text{verticesFrom } f r) u v \implies \text{before}(\text{verticesFrom } f v) r u$   
 <proof>

**lemma** *before-between*:  
 $\llbracket \text{before}(\text{verticesFrom } f x) y z; \text{distinct}(\text{vertices } f); x \in \mathcal{V} f; x \neq y \rrbracket \implies y \in \text{set}(\text{between}(\text{vertices } f) x z)$   
 <proof>

**lemma** *before-between2*:

$\llbracket \text{before } (\text{verticesFrom } f \ u) \ v \ w; \text{distinct}(\text{vertices } f); \ u \in \mathcal{V} \ f \rrbracket$   
 $\implies u = v \vee u \in \text{set } (\text{between } (\text{vertices } f) \ w \ v)$   
 <proof>

## 12.10 splitFace

**definition** *pre-splitFace* :: *graph*  $\Rightarrow$  *vertex*  $\Rightarrow$  *vertex*  $\Rightarrow$  *face*  $\Rightarrow$  *vertex list*  $\Rightarrow$  *bool*  
**where**

*pre-splitFace* *g ram1 ram2 oldF nvs*  $\equiv$   
 $\text{oldF} \in \mathcal{F} \ g \wedge \neg \text{final } \text{oldF} \wedge \text{distinct } (\text{vertices } \text{oldF}) \wedge \text{distinct } \text{nvs}$   
 $\wedge \mathcal{V} \ g \cap \text{set } \text{nvs} = \{\}$   
 $\wedge \mathcal{V} \ \text{oldF} \cap \text{set } \text{nvs} = \{\}$   
 $\wedge \text{ram1} \in \mathcal{V} \ \text{oldF} \wedge \text{ram2} \in \mathcal{V} \ \text{oldF}$   
 $\wedge \text{ram1} \neq \text{ram2}$   
 $\wedge (((\text{ram1}, \text{ram2}) \notin \text{edges } \text{oldF} \wedge (\text{ram2}, \text{ram1}) \notin \text{edges } \text{oldF})$   
 $\wedge (\text{ram1}, \text{ram2}) \notin \text{edges } g \wedge (\text{ram2}, \text{ram1}) \notin \text{edges } g) \vee \text{nvs} \neq [])$

**declare** *pre-splitFace-def* [*simp*]

**lemma** *pre-splitFace-pre-split-face*[*simp*]:

*pre-splitFace* *g ram1 ram2 oldF nvs*  $\implies$  *pre-split-face* *oldF ram1 ram2 nvs*  
 <proof>

**lemma** *pre-splitFace-oldF*[*simp*]:

*pre-splitFace* *g ram1 ram2 oldF nvs*  $\implies$   $\text{oldF} \in \mathcal{F} \ g$   
 <proof>

**declare** *pre-splitFace-def* [*simp del*]

**lemma** *splitFace-split-face*:

$\text{oldF} \in \mathcal{F} \ g \implies$   
 $(f_1, f_2, \text{newGraph}) = \text{splitFace } g \ \text{ram}_1 \ \text{ram}_2 \ \text{oldF} \ \text{newVs} \implies$   
 $(f_1, f_2) = \text{split-face } \text{oldF} \ \text{ram}_1 \ \text{ram}_2 \ \text{newVs}$   
 <proof>

**lemma** *split-face-empty-ram2-ram1-in-f12*:

*pre-split-face* *oldF ram1 ram2* []  $\implies$   
 $(f12, f21) = \text{split-face } \text{oldF} \ \text{ram1} \ \text{ram2} \ [] \implies (\text{ram2}, \text{ram1}) \in \text{edges } f12$   
 <proof>

**lemma** *split-face-empty-ram2-ram1-in-f12'*:

*pre-split-face* *oldF ram1 ram2* []  $\implies$   
 $(\text{ram2}, \text{ram1}) \in \text{edges } (\text{fst } (\text{split-face } \text{oldF} \ \text{ram1} \ \text{ram2} \ []))$   
 <proof>

**lemma** *splitFace-empty-ram2-ram1-in-f12*:

*pre-splitFace* *g ram1 ram2 oldF* []  $\implies$



$(f12, f21, newGraph) = splitFace\ g\ ram1\ ram2\ oldF\ [] \implies$   
 $(ram2, ram1) \in edges\ f12$   
 <proof>

**lemma** *splitFace-f12-new-vertices*:  
 $(f12, f21, newGraph) = splitFace\ g\ ram1\ ram2\ oldF\ newVs \implies$   
 $v \in set\ newVs \implies v \in \mathcal{V}\ f12$   
 <proof>

**lemma** *splitFace-add-vertices-direct[simp]*:  
 $vertices\ (snd\ (snd\ (splitFace\ g\ ram1\ ram2\ oldF\ [countVertices\ g\ ..<\ countVertices\ g\ +\ n])))$   
 $=\ vertices\ g\ @\ [countVertices\ g\ ..<\ countVertices\ g\ +\ n]$   
 <proof>

**lemma** *splitFace-delete-oldF*:  
 $(f12, f21, newGraph) = splitFace\ g\ ram1\ ram2\ oldF\ newVertexList \implies$   
 $oldF \neq f12 \implies oldF \neq f21 \implies distinct\ (faces\ g) \implies$   
 $oldF \notin \mathcal{F}\ newGraph$   
 <proof>

**lemma** *splitFace-faces-1*:  
 $(f12, f21, newGraph) = splitFace\ g\ ram1\ ram2\ oldF\ newVertexList \implies$   
 $oldF \in \mathcal{F}\ g \implies$   
 $set\ (faces\ newGraph) \cup \{oldF\} = \{f12, f21\} \cup set\ (faces\ g)$   
 $(is\ ?oldF \implies ?C \implies ?A = ?B)$   
 <proof>

**lemma** *splitFace-distinct1[intro]:pre-splitFace*  $g\ ram1\ ram2\ oldF\ newVertexList$   
 $\implies$   
 $distinct\ (vertices\ (fst\ (snd\ (splitFace\ g\ ram1\ ram2\ oldF\ newVertexList))))$   
 <proof>

**lemma** *splitFace-distinct2[intro]:pre-splitFace*  $g\ ram1\ ram2\ oldF\ newVertexList$   
 $\implies$   
 $distinct\ (vertices\ (fst\ (splitFace\ g\ ram1\ ram2\ oldF\ newVertexList)))$   
 <proof>

**lemma** *splitFace-add-f21'*:  $f' \in \mathcal{F}\ g' \implies fst\ (snd\ (splitFace\ g'\ v\ a\ f'\ nvl))$   
 $\in \mathcal{F}\ (snd\ (snd\ (splitFace\ g'\ v\ a\ f'\ nvl)))$   
 <proof>

**lemma** *split-face-help[simp]*:  $Suc\ 0 < |vertices\ (fst\ (split-face\ f'\ v\ a\ nvl))|$   
 <proof>

**lemma** *split-face-help'[simp]*:  $Suc\ 0 < |vertices\ (snd\ (split-face\ f'\ v\ a\ nvl))|$

$\langle \text{proof} \rangle$

**lemma** *splitFace-split*:  $f \in \mathcal{F} (\text{snd} (\text{snd} (\text{splitFace } g \ v \ a \ f' \ \text{nv}))) \implies$   
 $f \in \mathcal{F} \ g$   
 $\vee f = \text{fst} (\text{splitFace } g \ v \ a \ f' \ \text{nv})$   
 $\vee f = (\text{fst} (\text{snd} (\text{splitFace } g \ v \ a \ f' \ \text{nv})))$   
 $\langle \text{proof} \rangle$

**lemma** *pre-FaceDiv-between1*:  $\text{pre-splitFace } g' \ \text{ram1} \ \text{ram2} \ f \ [] \implies$   
 $\neg \text{between} (\text{vertices } f) \ \text{ram1} \ \text{ram2} = []$   
 $\langle \text{proof} \rangle$

**lemma** *pre-FaceDiv-between2*:  $\text{pre-splitFace } g' \ \text{ram1} \ \text{ram2} \ f \ [] \implies$   
 $\neg \text{between} (\text{vertices } f) \ \text{ram2} \ \text{ram1} = []$   
 $\langle \text{proof} \rangle$

**definition** *Edges* ::  $\text{vertex list} \Rightarrow (\text{vertex} \times \text{vertex}) \text{ set}$  **where**  
 $\text{Edges } \text{vs} \equiv \{(a,b). \text{is-sublist } [a,b] \ \text{vs}\}$

**lemma** *Edges-Nil[simp]*:  $\text{Edges } [] = \{\}$   
 $\langle \text{proof} \rangle$

**lemma** *Edges-rev*:  
 $\text{Edges} (\text{rev } (\text{zs}::\text{vertex list})) = \{(b,a). (a,b) \in \text{Edges } \text{zs}\}$   
 $\langle \text{proof} \rangle$

**lemma** *in-Edges-rev[simp]*:  
 $((a,b) : \text{Edges} (\text{rev } (\text{zs}::\text{vertex list}))) = ((b,a) \in \text{Edges } \text{zs})$   
 $\langle \text{proof} \rangle$

**lemma** *notinset-notinEdge1*:  $x \notin \text{set } \text{xs} \implies (x,y) \notin \text{Edges } \text{xs}$   
 $\langle \text{proof} \rangle$

**lemma** *notinset-notinEdge2*:  $y \notin \text{set } \text{xs} \implies (x,y) \notin \text{Edges } \text{xs}$   
 $\langle \text{proof} \rangle$

**lemma** *in-Edges-in-set*:  $(x,y) : \text{Edges } \text{vs} \implies x \in \text{set } \text{vs} \wedge y \in \text{set } \text{vs}$   
 $\langle \text{proof} \rangle$

**lemma** *edges-conv-Edges*:  
 $\text{distinct}(\text{vertices}(f::\text{face})) \implies \mathcal{E} \ f =$   
 $\text{Edges} (\text{vertices } f) \cup$   
 $(\text{if } \text{vertices } f = [] \text{ then } \{\} \text{ else } \{(last(\text{vertices } f), hd(\text{vertices } f))\})$   
 $\langle \text{proof} \rangle$

**lemma** *Edges-Cons*:  $Edges(x\#xs) =$   
 (if  $xs = []$  then  $\{\}$  else  $Edges\ xs \cup \{(x,hd\ xs)\}$ )  
 <proof>

**lemma** *Edges-append*:  $Edges(xs\ @\ ys) =$   
 (if  $xs = []$  then  $Edges\ ys$  else  
 if  $ys = []$  then  $Edges\ xs$  else  
 $Edges\ xs \cup Edges\ ys \cup \{(last\ xs, hd\ ys)\}$ )  
 <proof>

**lemma** *Edges-rev-disj*:  $distinct\ xs \implies Edges(rev\ xs) \cap Edges(xs) = \{\}$   
 <proof>

**lemma** *disj-sets-disj-Edges*:  
 $set\ xs \cap set\ ys = \{\} \implies Edges\ xs \cap Edges\ ys = \{\}$   
 <proof>

**lemma** *disj-sets-disj-Edges2*:  
 $set\ ys \cap set\ xs = \{\} \implies Edges\ xs \cap Edges\ ys = \{\}$   
 <proof>

**lemma** *finite-Edges[iff]*:  $finite(Edges\ xs)$   
 <proof>

**lemma** *Edges-compl*:  
 $\llbracket distinct\ vs; x \in set\ vs; y \in set\ vs; x \neq y \rrbracket \implies$   
 $Edges(x\ \# \text{between}\ vs\ x\ y\ @\ [y]) \cap Edges(y\ \# \text{between}\ vs\ y\ x\ @\ [x]) = \{\}$   
 <proof>

**lemma** *Edges-disj*:  
 $\llbracket distinct\ vs; x \in set\ vs; z \in set\ vs; x \neq y; y \neq z;$   
 $y \in set(\text{between}\ vs\ x\ z) \rrbracket \implies$   
 $Edges(x\ \# \text{between}\ vs\ x\ y\ @\ [y]) \cap Edges(y\ \# \text{between}\ vs\ y\ z\ @\ [z]) = \{\}$   
 <proof>

**lemma** *edges-conv-Un-Edges*:  
 $\llbracket distinct(vertices(f::face)); x \in \mathcal{V}\ f; y \in \mathcal{V}\ f; x \neq y \rrbracket \implies$   
 $\mathcal{E}\ f = Edges(x\ \# \text{between}\ (vertices\ f)\ x\ y\ @\ [y]) \cup$   
 $Edges(y\ \# \text{between}\ (vertices\ f)\ y\ x\ @\ [x])$   
 <proof>

**lemma** *Edges-between-edges*:  
 $\llbracket (a,b) \in Edges\ (u\ \# \text{between}\ (vertices(f::face))\ u\ v\ @\ [v]);$   
 $pre-split-face\ f\ u\ v\ vs \rrbracket \implies (a,b) \in \mathcal{E}\ f$

$\langle \text{proof} \rangle$

**lemma** *edges-split-face1: pre-split-face f u v vs  $\implies$*   
 $\mathcal{E}(\text{fst}(\text{split-face } f \ u \ v \ vs)) =$   
 $\text{Edges}(v \ \# \ \text{rev } vs \ @ \ [u]) \cup \text{Edges}(u \ \# \ \text{between } (\text{vertices } f) \ u \ v \ @ \ [v])$   
 $\langle \text{proof} \rangle$

**lemma** *edges-split-face2: pre-split-face f u v vs  $\implies$*   
 $\mathcal{E}(\text{snd}(\text{split-face } f \ u \ v \ vs)) =$   
 $\text{Edges}(u \ \# \ vs \ @ \ [v]) \cup \text{Edges}(v \ \# \ \text{between } (\text{vertices } f) \ v \ u \ @ \ [u])$   
 $\langle \text{proof} \rangle$

**lemma** *split-face-empty-ram1-ram2-in-f21:*  
 $\text{pre-split-face } \text{oldF } \text{ram1 } \text{ram2 } [] \implies$   
 $(f12, f21) = \text{split-face } \text{oldF } \text{ram1 } \text{ram2 } [] \implies (\text{ram1}, \text{ram2}) \in \text{edges } f21$   
 $\langle \text{proof} \rangle$

**lemma** *split-face-empty-ram1-ram2-in-f21':*  
 $\text{pre-split-face } \text{oldF } \text{ram1 } \text{ram2 } [] \implies$   
 $(\text{ram1}, \text{ram2}) \in \text{edges } (\text{snd } (\text{split-face } \text{oldF } \text{ram1 } \text{ram2 } []))$   
 $\langle \text{proof} \rangle$

**lemma** *splitFace-empty-ram1-ram2-in-f21:*  
 $\text{pre-splitFace } g \ \text{ram1 } \text{ram2 } \text{oldF } [] \implies$   
 $(f12, f21, \text{newGraph}) = \text{splitFace } g \ \text{ram1 } \text{ram2 } \text{oldF } [] \implies$   
 $(\text{ram1}, \text{ram2}) \in \text{edges } f21$   
 $\langle \text{proof} \rangle$

**lemma** *splitFace-f21-new-vertices:*  
 $(f12, f21, \text{newGraph}) = \text{splitFace } g \ \text{ram1 } \text{ram2 } \text{oldF } \text{newVs} \implies$   
 $v \in \text{set } \text{newVs} \implies v \in \mathcal{V} \ f21$   
 $\langle \text{proof} \rangle$

**lemma** *split-face-edges-f12:*

**assumes** *vors: pre-split-face f ram1 ram2 vs*  
 $(f12, f21) = \text{split-face } f \ \text{ram1 } \text{ram2 } vs$   
 $vs \neq [] \ \text{vs1} = \text{between } (\text{vertices } f) \ \text{ram1 } \text{ram2 } vs1 \neq []$

**shows**  $\text{edges } f12 =$   
 $\{(hd \ vs, \ \text{ram1}), (\text{ram1}, \ hd \ vs1), (\text{last } \ vs1, \ \text{ram2}), (\text{ram2}, \ \text{last } \ vs)\} \cup$   
 $\text{Edges}(\text{rev } vs) \cup \text{Edges } vs1 \ (\text{is } ?lhs = ?rhs)$   
 $\langle \text{proof} \rangle$

**lemma** *split-face-edges-f12-vs:*

**assumes**  $vors$ : *pre-split-face*  $f$   $ram1$   $ram2$   $\square$   
 $(f12, f21) = \textit{split-face } f \textit{ } ram1 \textit{ } ram2 \square$   
 $vs1 = \textit{between (vertices } f \textit{ ) } ram1 \textit{ } ram2 \textit{ } vs1 \neq \square$   
**shows**  $\textit{edges } f12 = \{(ram2, ram1), (ram1, hd \textit{ } vs1), (last \textit{ } vs1, ram2)\} \cup$   
 $\textit{Edges } vs1 \textit{ (is } ?lhs = ?rhs)$   
 $\langle \textit{proof} \rangle$

**lemma** *split-face-edges-f12-bet*:  
**assumes**  $vors$ : *pre-split-face*  $f$   $ram1$   $ram2$   $vs$   
 $(f12, f21) = \textit{split-face } f \textit{ } ram1 \textit{ } ram2 \textit{ } vs$   
 $vs \neq \square \textit{between (vertices } f \textit{ ) } ram1 \textit{ } ram2 = \square$   
**shows**  $\textit{edges } f12 = \{(hd \textit{ } vs, ram1), (ram1, ram2), (ram2, last \textit{ } vs)\} \cup$   
 $\textit{Edges}(rev \textit{ } vs) \textit{ (is } ?lhs = ?rhs)$   
 $\langle \textit{proof} \rangle$

**lemma** *split-face-edges-f12-bet-vs*:  
**assumes**  $vors$ : *pre-split-face*  $f$   $ram1$   $ram2$   $\square$   
 $(f12, f21) = \textit{split-face } f \textit{ } ram1 \textit{ } ram2 \square$   
 $\textit{between (vertices } f \textit{ ) } ram1 \textit{ } ram2 = \square$   
**shows**  $\textit{edges } f12 = \{(ram2, ram1), (ram1, ram2)\} \textit{ (is } ?lhs = ?rhs)$   
 $\langle \textit{proof} \rangle$

**lemma** *split-face-edges-f12-subset*: *pre-split-face*  $f$   $ram1$   $ram2$   $vs \implies$   
 $(f12, f21) = \textit{split-face } f \textit{ } ram1 \textit{ } ram2 \textit{ } vs \implies vs \neq \square \implies$   
 $\{(hd \textit{ } vs, ram1), (ram2, last \textit{ } vs)\} \cup \textit{Edges}(rev \textit{ } vs) \subseteq \textit{edges } f12$   
 $\langle \textit{proof} \rangle$

**lemma** *split-face-edges-f21*:  
**assumes**  $vors$ : *pre-split-face*  $f$   $ram1$   $ram2$   $vs$   
 $(f12, f21) = \textit{split-face } f \textit{ } ram1 \textit{ } ram2 \textit{ } vs$   
 $vs \neq \square \textit{ } vs2 = \textit{between (vertices } f \textit{ ) } ram2 \textit{ } ram1 \textit{ } vs2 \neq \square$   
**shows**  $\textit{edges } f21 = \{(last \textit{ } vs2, ram1), (ram1, hd \textit{ } vs), (last \textit{ } vs, ram2), (ram2, hd$   
 $\textit{ } vs2)\} \cup$   
 $\textit{Edges } vs \cup \textit{Edges } vs2 \textit{ (is } ?lhs = ?rhs)$   
 $\langle \textit{proof} \rangle$

**lemma** *split-face-edges-f21-vs*:  
**assumes**  $vors$ : *pre-split-face*  $f$   $ram1$   $ram2$   $\square$   
 $(f12, f21) = \textit{split-face } f \textit{ } ram1 \textit{ } ram2 \square$   
 $vs2 = \textit{between (vertices } f \textit{ ) } ram2 \textit{ } ram1 \textit{ } vs2 \neq \square$   
**shows**  $\textit{edges } f21 = \{(last \textit{ } vs2, ram1), (ram1, ram2), (ram2, hd \textit{ } vs2)\} \cup$   
 $\textit{Edges } vs2 \textit{ (is } ?lhs = ?rhs)$   
 $\langle \textit{proof} \rangle$

**lemma** *split-face-edges-f21-bet*:

**assumes** *vors*: *pre-split-face f ram1 ram2 vs*

$(f12, f21) = \text{split-face } f \text{ ram1 ram2 vs}$

$vs \neq [] \text{ between (vertices } f) \text{ ram2 ram1} = []$

**shows**  $\text{edges } f21 = \{(ram1, hd \text{ vs}), (last \text{ vs}, ram2), (ram2, ram1)\} \cup$

$\text{Edges vs (is ?lhs = ?rhs)}$

*<proof>*

**lemma** *split-face-edges-f21-bet-vs*:

**assumes** *vors*: *pre-split-face f ram1 ram2 []*

$(f12, f21) = \text{split-face } f \text{ ram1 ram2 []}$

$\text{between (vertices } f) \text{ ram2 ram1} = []$

**shows**  $\text{edges } f21 = \{(ram1, ram2), (ram2, ram1)\} \text{ (is ?lhs = ?rhs)}$

*<proof>*

**lemma** *split-face-edges-f21-subset*: *pre-split-face f ram1 ram2 vs*  $\implies$

$(f12, f21) = \text{split-face } f \text{ ram1 ram2 vs} \implies vs \neq [] \implies$

$\{(last \text{ vs}, ram2), (ram1, hd \text{ vs})\} \cup \text{Edges vs} \subseteq \text{edges } f21$

*<proof>*

**lemma** *verticesFrom-ram1*: *pre-split-face f ram1 ram2 vs*  $\implies$

$\text{verticesFrom } f \text{ ram1} = ram1 \# \text{ between (vertices } f) \text{ ram1 ram2} @ ram2 \#$

$\text{between (vertices } f) \text{ ram2 ram1}$

*<proof>*

**lemma** *split-face-edges-f-vs1-vs2*:

**assumes** *vors*: *pre-split-face f ram1 ram2 vs*

$\text{between (vertices } f) \text{ ram1 ram2} = []$

$\text{between (vertices } f) \text{ ram2 ram1} = []$

**shows**  $\text{edges } f = \{(ram2, ram1), (ram1, ram2)\} \text{ (is ?lhs = ?rhs)}$

*<proof>*

**lemma** *split-face-edges-f-vs1*:

**assumes** *vors*: *pre-split-face f ram1 ram2 vs*

$\text{between (vertices } f) \text{ ram1 ram2} = []$

$vs2 = \text{between (vertices } f) \text{ ram2 ram1 vs2} \neq []$

**shows**  $\text{edges } f = \{(last \text{ vs2}, ram1), (ram1, ram2), (ram2, hd \text{ vs2})\} \cup$

$\text{Edges vs2 (is ?lhs = ?rhs)}$

*<proof>*

**lemma** *split-face-edges-f-vs2*:

**assumes** *vors*: *pre-split-face f ram1 ram2 vs*

$vs1 = \text{between (vertices } f) \text{ ram1 ram2 vs1} \neq []$

$\text{between (vertices } f) \text{ ram2 ram1} = []$

**shows**  $\text{edges } f = \{(ram2, ram1), (ram1, hd \text{ vs1}), (last \text{ vs1}, ram2)\} \cup$

$\text{Edges vs1 (is ?lhs = ?rhs)}$

*<proof>*

**lemma** *split-face-edges-f*:

**assumes** *vors*: *pre-split-face f ram1 ram2 vs*

*vs1 = between (vertices f) ram1 ram2 vs1 ≠ []*

*vs2 = between (vertices f) ram2 ram1 vs2 ≠ []*

**shows** *edges f = {(last vs2, ram1), (ram1, hd vs1), (last vs1, ram2), (ram2, hd vs2)}* ∪

*Edges vs1 ∪ Edges vs2 (is ?lhs = ?rhs)*

*<proof>*

**lemma** *split-face-edges-f12-f21*:

*pre-split-face f ram1 ram2 vs ⇒ (f12, f21) = split-face f ram1 ram2 vs ⇒*

*vs ≠ []*

*⇒ edges f12 ∪ edges f21 = edges f ∪*

*{(hd vs, ram1), (ram1, hd vs), (last vs, ram2), (ram2, last vs)}* ∪

*Edges vs ∪*

*Edges (rev vs)*

*<proof>*

**lemma** *split-face-edges-f12-f21-vs*:

*pre-split-face f ram1 ram2 [] ⇒ (f12, f21) = split-face f ram1 ram2 []*

*⇒ edges f12 ∪ edges f21 = edges f ∪*

*{(ram2, ram1), (ram1, ram2)}*

*<proof>*

**lemma** *split-face-edges-f12-f21-sym*:

*f ∈ ℱ g ⇒*

*pre-split-face f ram1 ram2 vs ⇒ (f12, f21) = split-face f ram1 ram2 vs*

*⇒ ((a,b) ∈ edges f12 ∨ (a,b) ∈ edges f21) =*

*((a,b) ∈ edges f ∨*

*((b,a) ∈ edges f12 ∨ (b,a) ∈ edges f21) ∧*

*((a,b) ∈ edges f12 ∨ (a,b) ∈ edges f21)))*

*<proof>*

**lemma** *splitFace-edges-g'-help*: *pre-splitFace g ram1 ram2 f vs ⇒*

*(f12, f21, g') = splitFace g ram1 ram2 f vs ⇒ vs ≠ [] ⇒*

*edges g' = edges g ∪ edges f ∪ Edges vs ∪ Edges(rev vs) ∪*

*{(ram2, last vs), (hd vs, ram1), (ram1, hd vs), (last vs, ram2)}*

*<proof>*

**lemma** *pre-splitFace-edges-f-in-g*: *pre-splitFace g ram1 ram2 f vs ⇒ edges f ⊆*

*edges g*

*<proof>*

**lemma** *pre-splitFace-edges-f-in-g2*: *pre-splitFace g ram1 ram2 f vs ⇒ x ∈ edges*

$f \implies x \in \text{edges } g$   
 ⟨proof⟩

**lemma** *splitFace-edges-g'*:  $\text{pre-splitFace } g \text{ ram1 ram2 } f \text{ vs} \implies$   
 $(f12, f21, g') = \text{splitFace } g \text{ ram1 ram2 } f \text{ vs} \implies \text{vs} \neq [] \implies$   
 $\text{edges } g' = \text{edges } g \cup \text{Edges } \text{vs} \cup \text{Edges}(\text{rev } \text{vs}) \cup$   
 $\{(ram2, \text{last } \text{vs}), (\text{hd } \text{vs}, ram1), (ram1, \text{hd } \text{vs}), (\text{last } \text{vs}, ram2)\}$   
 ⟨proof⟩

**lemma** *splitFace-edges-g'-vs*:  $\text{pre-splitFace } g \text{ ram1 ram2 } f [] \implies$   
 $(f12, f21, g') = \text{splitFace } g \text{ ram1 ram2 } f [] \implies$   
 $\text{edges } g' = \text{edges } g \cup \{(ram1, ram2), (ram2, ram1)\}$   
 ⟨proof⟩

**lemma** *splitFace-edges-incr*:  
 $\text{pre-splitFace } g \text{ ram1 ram2 } f \text{ vs} \implies$   
 $(f_1, f_2, g') = \text{splitFace } g \text{ ram1 ram2 } f \text{ vs} \implies$   
 $\text{edges } g \subseteq \text{edges } g'$   
 ⟨proof⟩

**lemma** *snd-snd-splitFace-edges-incr*:  
 $\text{pre-splitFace } g \text{ v}_1 \text{ v}_2 \text{ f vs} \implies$   
 $\text{edges } g \subseteq \text{edges}(\text{snd}(\text{snd}(\text{splitFace } g \text{ v}_1 \text{ v}_2 \text{ f vs})))$   
 ⟨proof⟩

## 12.11 removeNones

**definition** *removeNones* :: 'a option list  $\Rightarrow$  'a list **where**  
 $\text{removeNones } v\text{OptionList} \equiv [\text{the } x. x \leftarrow v\text{OptionList}, x \neq \text{None}]$

**declare** *removeNones-def* [simp]

**lemma** *removeNones-inI*[intro]:  $\text{Some } a \in \text{set } ls \implies a \in \text{set } (\text{removeNones } ls)$   
 ⟨proof⟩

**lemma** *removeNones-hd*[simp]:  $\text{removeNones } (\text{Some } a \# ls) = a \# \text{removeNones } ls$   
 ⟨proof⟩

**lemma** *removeNones-last*[simp]:  $\text{removeNones } (ls @ [\text{Some } a]) = \text{removeNones } ls @ [a]$   
 ⟨proof⟩

**lemma** *removeNones-in*[simp]:  $\text{removeNones } (as @ \text{Some } a \# bs) = \text{removeNones } as @ a \# \text{removeNones } bs$   
 ⟨proof⟩

**lemma** *removeNones-none-hd*[simp]:  $\text{removeNones } (\text{None} \# ls) = \text{removeNones } ls$   
 ⟨proof⟩

**lemma** *removeNones-none-last*[simp]:  $\text{removeNones } (ls @ [\text{None}]) = \text{removeNones } ls$   
 ⟨proof⟩

**lemma** *removeNones-none-in*[simp]:  $\text{removeNones } (as @ \text{None} \# bs) = \text{removeNones } (as @ bs)$   
 ⟨proof⟩



**lemma** *removeNones-empty[simp]*:  $\text{removeNones } [] = []$   $\langle \text{proof} \rangle$   
**declare** *removeNones-def* [simp del]

### 12.12 *natToVertexList*

**primrec** *natToVertexListRec* ::  
 $\text{nat} \Rightarrow \text{vertex} \Rightarrow \text{face} \Rightarrow \text{nat list} \Rightarrow \text{vertex option list}$   
**where**  
 $\text{natToVertexListRec } \text{old } v f [] = []$  |  
 $\text{natToVertexListRec } \text{old } v f (i\#is) =$   
 $(\text{if } i = \text{old} \text{ then } \text{None}\#\text{natToVertexListRec } i v f \text{ is}$   
 $\text{else } \text{Some } (f^i \cdot v)$   
 $\#\text{natToVertexListRec } i v f \text{ is})$

**primrec** *natToVertexList* ::  
 $\text{vertex} \Rightarrow \text{face} \Rightarrow \text{nat list} \Rightarrow \text{vertex option list}$   
**where**  
 $\text{natToVertexList } v f [] = []$  |  
 $\text{natToVertexList } v f (i\#is) =$   
 $(\text{if } i = 0 \text{ then } (\text{Some } v)\#(\text{natToVertexListRec } i v f \text{ is}) \text{ else } [])$

### 12.13 *indexToVertexList*

**lemma** *nextVertex-inj*:  
 $\text{distinct } (\text{vertices } f) \Longrightarrow v \in \mathcal{V} f \Longrightarrow$   
 $i < \text{length } (\text{vertices } (f::\text{face})) \Longrightarrow a < \text{length } (\text{vertices } f) \Longrightarrow$   
 $f^a \cdot v = f^i \cdot v \Longrightarrow i = a$   
 $\langle \text{proof} \rangle$

**lemma** *a: distinct (vertices f)  $\Longrightarrow v \in \mathcal{V} f \Longrightarrow (\forall i \in \text{set } is. i < \text{length } (\text{vertices } f)) \Longrightarrow$*   
 $(\bigwedge a. a < \text{length } (\text{vertices } f) \Longrightarrow \text{hideDupsRec } ((f \cdot \hat{\ } a) v) [(f \cdot \hat{\ } k) v. k \leftarrow is] = \text{natToVertexListRec } a v f \text{ is})$   
 $\langle \text{proof} \rangle$

**lemma** *indexToVertexList-natToVertexList-eq*:  $\text{distinct } (\text{vertices } f) \Longrightarrow v \in \mathcal{V} f \Longrightarrow$   
 $(\forall i \in \text{set } is. i < \text{length } (\text{vertices } f)) \Longrightarrow is \neq [] \Longrightarrow$   
 $\text{hd } is = 0 \Longrightarrow \text{indexToVertexList } f v is = \text{natToVertexList } v f \text{ is}$   
 $\langle \text{proof} \rangle$

**lemma** *nvlr-length*:  $\bigwedge \text{old}. (\text{length } (\text{natToVertexListRec } \text{old } v f \text{ ls})) = \text{length } ls$

$\langle \text{proof} \rangle$

**lemma** *nvl-length[simp]*:  $hd\ e = 0 \implies length\ (natToVertexList\ v\ f\ e) = length\ e$   
 $\langle \text{proof} \rangle$

**lemma** *natToVertexListRec-length[simp]*:  $\bigwedge\ e\ f.\ length\ (natToVertexListRec\ e\ v\ f\ es) = length\ es$   
 $\langle \text{proof} \rangle$

**lemma** *natToVertexList-length[simp]*:  $incrIndexList\ es\ (length\ es)\ (length\ (vertices\ f)) \implies$   
 $length\ (natToVertexList\ v\ f\ es) = length\ es\ \langle \text{proof} \rangle$

**lemma** *natToVertexList-nth-Suc*:  $incrIndexList\ es\ (length\ es)\ (length\ (vertices\ f))$   
 $\implies\ Suc\ n < length\ es \implies$   
 $(natToVertexList\ v\ f\ es)!(Suc\ n) = (if\ (es!n = es!(Suc\ n))\ then\ None\ else\ Some\ (f^{(es!Suc\ n)}\ .\ v))$   
 $\langle \text{proof} \rangle$

**lemma** *natToVertexList-nth-0*:  $incrIndexList\ es\ (length\ es)\ (length\ (vertices\ f))$   
 $\implies\ 0 < length\ es \implies$   
 $(natToVertexList\ v\ f\ es)!0 = Some\ (f^{(es!0)}\ .\ v)$   
 $\langle \text{proof} \rangle$

**lemma** *natToVertexList-hd[simp]*:  
 $incrIndexList\ es\ (length\ es)\ (length\ (vertices\ f)) \implies hd\ (natToVertexList\ v\ f\ es)$   
 $=\ Some\ v$   
 $\langle \text{proof} \rangle$

**lemma** *nth-last[intro]*:  $Suc\ i = length\ xs \implies xs!i = last\ xs$   
 $\langle \text{proof} \rangle$

**declare** *incrIndexList-help4* [simp del]

**lemma** *natToVertexList-last[simp]*:  
 $distinct\ (vertices\ f) \implies v \in \mathcal{V}\ f \implies incrIndexList\ es\ (length\ es)\ (length\ (vertices\ f)) \implies last\ (natToVertexList\ v\ f\ es) = Some\ (last\ (verticesFrom\ f\ v))$   
 $\langle \text{proof} \rangle$

**lemma** *indexToVertexList-last[simp]*:  
 $distinct\ (vertices\ f) \implies v \in \mathcal{V}\ f \implies incrIndexList\ es\ (length\ es)\ (length\ (vertices\ f)) \implies last\ (indexToVertexList\ f\ v\ es) = Some\ (last\ (verticesFrom\ f\ v))$   
 $\langle \text{proof} \rangle$

**lemma** *nths-take*:  $\bigwedge\ n\ iset.\ \forall\ i \in iset.\ i < n \implies nths\ (take\ n\ xs)\ iset = nths\ xs$   
*iset*

$\langle \text{proof} \rangle$

**lemma** *nths-reduceIndices*:  $\bigwedge \text{iset}. \text{nths } xs \text{ iset} = \text{nths } xs \{i. i < \text{length } xs \wedge i \in \text{iset}\}$

$\langle \text{proof} \rangle$

**lemma** *natToVertexList-nths1*:  $\text{distinct } (\text{vertices } f) \implies$   
 $v \in \mathcal{V} f \implies vs = \text{verticesFrom } f v \implies$   
 $\text{incrIndexList } es \ (\text{length } es) \ (\text{length } vs) \implies n \leq \text{length } es \implies$   
 $\text{nths } (\text{take } (\text{Suc } (es!(n - 1))) \ vs) \ (\text{set } (\text{take } n \ es))$   
 $= \text{removeNones } (\text{take } n \ (\text{natToVertexList } v \ f \ es))$

$\langle \text{proof} \rangle$

**lemma** *natToVertexList-nths*:  $\text{distinct } (\text{vertices } f) \implies v \in \mathcal{V} f \implies$   
 $\text{incrIndexList } es \ (\text{length } es) \ (\text{length } (\text{vertices } f)) \implies$   
 $\text{nths } (\text{verticesFrom } f v) \ (\text{set } es) = \text{removeNones } (\text{natToVertexList } v \ f \ es)$

$\langle \text{proof} \rangle$

**lemma** *filter-Cons2*:

$x \notin \text{set } ys \implies [y \leftarrow ys. y = x \vee P y] = [y \leftarrow ys. P y]$

$\langle \text{proof} \rangle$

**lemma** *natToVertexList-removeNones*:

$\text{distinct } (\text{vertices } f) \implies v \in \mathcal{V} f \implies$   
 $\text{incrIndexList } es \ (\text{length } es) \ (\text{length } (\text{vertices } f)) \implies$   
 $[x \leftarrow \text{verticesFrom } f v. x \in \text{set } (\text{removeNones } (\text{natToVertexList } v \ f \ es))]$   
 $= \text{removeNones } (\text{natToVertexList } v \ f \ es)$

$\langle \text{proof} \rangle$

**definition** *is-duplicateEdge* ::  $\text{graph} \Rightarrow \text{face} \Rightarrow \text{vertex} \Rightarrow \text{vertex} \Rightarrow \text{bool}$  **where**

$\text{is-duplicateEdge } g \ f \ a \ b \equiv$   
 $((a, b) \in \text{edges } g \wedge (a, b) \notin \text{edges } f \wedge (b, a) \notin \text{edges } f)$   
 $\vee ((b, a) \in \text{edges } g \wedge (b, a) \notin \text{edges } f \wedge (a, b) \notin \text{edges } f)$

**definition** *invalidVertexList* ::  $\text{graph} \Rightarrow \text{face} \Rightarrow \text{vertex option list} \Rightarrow \text{bool}$  **where**

$\text{invalidVertexList } g \ f \ vs \equiv$   
 $\exists i < |vs| - 1.$   
 $\text{case } vs!i \text{ of } \text{None} \Rightarrow \text{False}$   
 $\quad | \text{Some } a \Rightarrow \text{case } vs!(i+1) \text{ of } \text{None} \Rightarrow \text{False}$   
 $\quad | \text{Some } b \Rightarrow \text{is-duplicateEdge } g \ f \ a \ b$

## 12.14 *pre-subdivFace*(<sup>1</sup>)

**definition** *pre-subdivFace-face* ::  $\text{face} \Rightarrow \text{vertex} \Rightarrow \text{vertex option list} \Rightarrow \text{bool}$  **where**  
 $\text{pre-subdivFace-face } f \ v' \ v\text{OptionList} \equiv$

$$\begin{aligned}
& [v \leftarrow \text{verticesFrom } f \ v'. \ v \in \text{set } (\text{removeNones } v\text{OptionList})] \\
& = (\text{removeNones } v\text{OptionList}) \\
& \wedge \neg \text{final } f \wedge \text{distinct } (\text{vertices } f) \\
& \wedge \text{hd } (v\text{OptionList}) = \text{Some } v' \\
& \wedge v' \in \mathcal{V} f \\
& \wedge \text{last } (v\text{OptionList}) = \text{Some } (\text{last } (\text{verticesFrom } f \ v')) \\
& \wedge \text{hd } (\text{tl } (v\text{OptionList})) \neq \text{last } (v\text{OptionList}) \\
& \wedge 2 < | v\text{OptionList} | \\
& \wedge v\text{OptionList} \neq [] \\
& \wedge \text{tl } (v\text{OptionList}) \neq []
\end{aligned}$$

**definition** *pre-subdivFace* :: *graph*  $\Rightarrow$  *face*  $\Rightarrow$  *vertex*  $\Rightarrow$  *vertex option list*  $\Rightarrow$  *bool*  
**where**

$$\begin{aligned}
& \text{pre-subdivFace } g \ f \ v' \ v\text{OptionList} \equiv \\
& \text{pre-subdivFace-face } f \ v' \ v\text{OptionList} \wedge \neg \text{invalidVertexList } g \ f \ v\text{OptionList}
\end{aligned}$$

**definition** *pre-subdivFace'* :: *graph*  $\Rightarrow$  *face*  $\Rightarrow$  *vertex*  $\Rightarrow$  *vertex*  $\Rightarrow$  *nat*  $\Rightarrow$  *vertex option list*  $\Rightarrow$  *bool* **where**

$$\begin{aligned}
& \text{pre-subdivFace}' \ g \ f \ v' \ \text{ram1} \ n \ v\text{OptionList} \equiv \\
& \neg \text{final } f \wedge v' \in \mathcal{V} f \wedge \text{ram1} \in \mathcal{V} f \\
& \wedge v' \notin \text{set } (\text{removeNones } v\text{OptionList}) \\
& \wedge \text{distinct } (\text{vertices } f) \\
& \wedge ( \\
& \quad [v \leftarrow \text{verticesFrom } f \ v'. \ v \in \text{set } (\text{removeNones } v\text{OptionList})] \\
& \quad = (\text{removeNones } v\text{OptionList}) \\
& \wedge \text{before } (\text{verticesFrom } f \ v') \ \text{ram1} \ (\text{hd } (\text{removeNones } v\text{OptionList})) \\
& \wedge \text{last } (v\text{OptionList}) = \text{Some } (\text{last } (\text{verticesFrom } f \ v')) \\
& \wedge v\text{OptionList} \neq [] \\
& \wedge ((v' = \text{ram1} \wedge (0 < n)) \vee ((v' = \text{ram1} \wedge (\text{hd } (v\text{OptionList}) \neq \text{Some } (\text{last } (\text{verticesFrom } f \ v')))) \vee (v' \neq \text{ram1}))) \\
& \wedge \neg \text{invalidVertexList } g \ f \ v\text{OptionList} \\
& \wedge (n = 0 \wedge \text{hd } (v\text{OptionList}) \neq \text{None} \longrightarrow \neg \text{is-duplicateEdge } g \ f \ \text{ram1} \ (\text{the } (\text{hd } (v\text{OptionList})))) \\
& \quad \vee (v\text{OptionList} = [] \wedge v' \neq \text{ram1}) \\
& )
\end{aligned}$$

**lemma** *pre-subdivFace-face-in-f[intro]*: *pre-subdivFace-face* *f v ls*  $\Longrightarrow$  *Some a*  $\in$  *set ls*  $\Longrightarrow$  *a*  $\in$  *set (verticesFrom f v)*  
 $\langle \text{proof} \rangle$

**lemma** *pre-subdivFace-in-f[intro]*: *pre-subdivFace* *g f v ls*  $\Longrightarrow$  *Some a*  $\in$  *set ls*  $\Longrightarrow$  *a*  $\in$  *set (verticesFrom f v)*  
 $\langle \text{proof} \rangle$

**lemma** *pre-subdivFace-face-in-f'[intro]*: *pre-subdivFace-face* *f v ls*  $\Longrightarrow$  *Some a*  $\in$  *set ls*  $\Longrightarrow$  *a*  $\in$   $\mathcal{V} f$

*<proof>*

**lemma** *filter-congs-shorten1*:  $\text{distinct } (\text{verticesFrom } f \ v) \implies [v \leftarrow \text{verticesFrom } f \ v$   
 $. \ v = a \ \vee \ v \in \text{set } vs] = (a \ \# \ vs)$   
 $\implies [v \leftarrow \text{verticesFrom } f \ v . \ v \in \text{set } vs] = vs$   
*<proof>*

**lemma** *ovl-shorten*:  $\text{distinct } (\text{verticesFrom } f \ v) \implies$   
 $[v \leftarrow \text{verticesFrom } f \ v . \ v \in \text{set } (\text{removeNones } (va \ \# \ vol))] = (\text{removeNones } (va$   
 $\ \# \ vol))$   
 $\implies [v \leftarrow \text{verticesFrom } f \ v . \ v \in \text{set } (\text{removeNones } (vol))] = (\text{removeNones}$   
 $(vol))$   
*<proof>*

**lemma** *pre-subdivFace-face-distinct*:  $\text{pre-subdivFace-face } f \ v \ \text{vol} \implies \text{distinct } (\text{removeNones}$   
 $vol)$   
*<proof>*

**lemma** *invalidVertexList-shorten*:  $\text{invalidVertexList } g \ f \ vol \implies \text{invalidVertexList}$   
 $g \ f \ (v \ \# \ vol)$   
*<proof>*

**lemma** *pre-subdivFace-pre-subdivFace'*:  $v \in \mathcal{V} \ f \implies \text{pre-subdivFace } g \ f \ v \ (vo \ \#$   
 $vol) \implies$   
 $\text{pre-subdivFace}' \ g \ f \ v \ v \ 0 \ (vol)$   
*<proof>*

**lemma** *pre-subdivFace'-distinct*:  $\text{pre-subdivFace}' \ g \ f \ v' \ v \ n \ vol \implies \text{distinct } (\text{removeNones}$   
 $vol)$   
*<proof>*

**lemma** *natToVertexList-pre-subdivFace-face*:  
 $\neg \text{final } f \implies \text{distinct } (\text{vertices } f) \implies v \in \mathcal{V} \ f \implies 2 < |es| \implies$   
 $\text{incrIndexList } es \ (\text{length } es) \ (\text{length } (\text{vertices } f)) \implies$   
 $\text{pre-subdivFace-face } f \ v \ (\text{natToVertexList } v \ f \ es)$   
*<proof>*

**lemma** *indexToVertexList-pre-subdivFace-face*:  
 $\neg \text{final } f \implies \text{distinct } (\text{vertices } f) \implies v \in \mathcal{V} \ f \implies 2 < |es| \implies$   
 $\text{incrIndexList } es \ (\text{length } es) \ (\text{length } (\text{vertices } f)) \implies$   
 $\text{pre-subdivFace-face } f \ v \ (\text{indexToVertexList } f \ v \ es)$   
*<proof>*

**lemma** *subdivFace-subdivFace'-eq*:  $\text{pre-subdivFace } g \ f \ v \ vol \implies \text{subdivFace } g \ f \ vol$   
 $= \text{subdivFace}' \ g \ f \ v \ 0 \ (tl \ vol)$

$\langle \text{proof} \rangle$

**lemma** *pre-subdivFace'-None*:

$\text{pre-subdivFace}' g f v' v n (\text{None} \# \text{vol}) \implies$   
 $\text{pre-subdivFace}' g f v' v (\text{Suc } n) \text{vol}$

$\langle \text{proof} \rangle$

**declare** *verticesFrom-between* [*simp del*]

**lemma** *verticesFrom-split*:  $v \# \text{tl} (\text{verticesFrom } f v) = \text{verticesFrom } f v \langle \text{proof} \rangle$

**lemma** *verticesFrom-v*:  $\text{distinct} (\text{vertices } f) \implies \text{vertices } f = a @ v \# b \implies$   
 $\text{verticesFrom } f v = v \# b @ a$

$\langle \text{proof} \rangle$

**lemma** *splitAt-fst[*simp*]*:  $\text{distinct } xs \implies xs = a @ v \# b \implies \text{fst} (\text{splitAt } v xs) =$   
 $a$

$\langle \text{proof} \rangle$

**lemma** *splitAt-snd[*simp*]*:  $\text{distinct } xs \implies xs = a @ v \# b \implies \text{snd} (\text{splitAt } v xs)$   
 $= b$

$\langle \text{proof} \rangle$

**lemma** *verticesFrom-splitAt-v-fst[*simp*]*:

$\text{distinct} (\text{verticesFrom } f v) \implies \text{fst} (\text{splitAt } v (\text{verticesFrom } f v)) = []$

$\langle \text{proof} \rangle$

**lemma** *verticesFrom-splitAt-v-snd[*simp*]*:

$\text{distinct} (\text{verticesFrom } f v) \implies \text{snd} (\text{splitAt } v (\text{verticesFrom } f v)) = \text{tl} (\text{verticesFrom}$   
 $f v)$

$\langle \text{proof} \rangle$

**lemma** *filter-distinct-at*:

$\text{distinct } xs \implies xs = (as @ u \# bs) \implies [v \leftarrow xs. v = u \vee P v] = u \# us \implies$   
 $[v \leftarrow bs. P v] = us \wedge [v \leftarrow as. P v] = []$

$\langle \text{proof} \rangle$

**lemma** *filter-distinct-at3*:  $\text{distinct } xs \implies xs = (as @ u \# bs) \implies$

$[v \leftarrow xs. v = u \vee P v] = u \# us \implies \forall z \in \text{set } zs. z \in \text{set } as \vee \neg (P z) \implies$   
 $[v \leftarrow zs @ bs. P v] = us$

$\langle \text{proof} \rangle$

**lemma** *filter-distinct-at4*:  $\text{distinct } xs \implies xs = (as @ u \# bs)$

$\implies [v \leftarrow xs. v = u \vee v \in \text{set } us] = u \# us$

$\implies \text{set } zs \cap \text{set } us \subseteq \{u\} \cup \text{set } as$

$\implies [v \leftarrow zs@bs. v \in \text{set } us] = us$   
 <proof>

**lemma filter-distinct-at5:**  $\text{distinct } xs \implies xs = (as @ u \# bs)$   
 $\implies [v \leftarrow xs. v = u \vee v \in \text{set } us] = u \# us$   
 $\implies \text{set } zs \cap \text{set } xs \subseteq \{u\} \cup \text{set } as$   
 $\implies [v \leftarrow zs@bs. v \in \text{set } us] = us$   
 <proof>

**lemma filter-distinct-at6:**  $\text{distinct } xs \implies xs = (as @ u \# bs)$   
 $\implies [v \leftarrow xs. v = u \vee v \in \text{set } us] = u \# us$   
 $\implies \text{set } zs \cap \text{set } xs \subseteq \{u\} \cup \text{set } as$   
 $\implies [v \leftarrow zs@bs. v \in \text{set } us] = us \wedge [v \leftarrow bs. v \in \text{set } us] = us$   
 <proof>

**lemma filter-distinct-at-special:**  
 $\text{distinct } xs \implies xs = (as @ u \# bs)$   
 $\implies [v \leftarrow xs. v = u \vee v \in \text{set } us] = u \# us$   
 $\implies \text{set } zs \cap \text{set } xs \subseteq \{u\} \cup \text{set } as$   
 $\implies us = \text{hd-us} \# \text{tl-us}$   
 $\implies [v \leftarrow zs@bs. v \in \text{set } us] = us \wedge \text{hd-us} \in \text{set } bs$   
 <proof>

**lemma pre-subdivFace'-Some1':**  
**assumes**  $\text{pre-add: pre-subdivFace}' g f v' v n ((\text{Some } u) \# \text{vol})$   
**and**  $\text{pre-fdg: pre-splitFace } g v u f ws$   
**and**  $\text{fdg: } f21 = \text{fst } (\text{snd } (\text{splitFace } g v u f ws))$   
**and**  $g': g' = \text{snd } (\text{snd } (\text{splitFace } g v u f ws))$   
**shows**  $\text{pre-subdivFace}' g' f21 v' u 0 \text{vol}$   
 <proof>

**lemma before-filter:**  $\bigwedge ys. \text{filter } P xs = ys \implies \text{distinct } xs \implies \text{before } ys \ u \ v \implies$   
 $\text{before } xs \ u \ v$   
 <proof>

**lemma pre-subdivFace'-Some2:**  $\text{pre-subdivFace}' g f v' v 0 ((\text{Some } u) \# \text{vol}) \implies$   
 $\text{pre-subdivFace}' g f v' u 0 \text{vol}$   
 <proof>

**lemma pre-subdivFace'-preFaceDiv:**  $\text{pre-subdivFace}' g f v' v n ((\text{Some } u) \# \text{vol})$   
 $\implies f \in \mathcal{F} g \implies (f \cdot v = u \implies n \neq 0) \implies \mathcal{V} f \subseteq \mathcal{V} g$   
 $\implies \text{pre-splitFace } g v u f [\text{countVertices } g ..< \text{countVertices } g + n]$

*<proof>*

**lemma** *pre-subdivFace'-Some1*:

*pre-subdivFace' g f v' v n ((Some u) # vol)*  
 $\implies f \in \mathcal{F} g \implies (f \cdot v = u \implies n \neq 0) \implies \mathcal{V} f \subseteq \mathcal{V} g$   
 $\implies f21 = fst (snd (splitFace g v u f [countVertices g ..< countVertices g + n]))$   
 $\implies g' = snd (snd (splitFace g v u f [countVertices g ..< countVertices g + n]))$   
 $\implies pre-subdivFace' g' f21 v' u 0 vol$   
*<proof>*

**end**

## 13 Invariants of (Plane) Graphs

**theory** *Invariants*

**imports** *FaceDivisionProps*

**begin**

### 13.1 Rotation of face into normal form

**definition** *minVertex* :: *face*  $\Rightarrow$  *vertex* **where**  
*minVertex f*  $\equiv$  *min-list (vertices f)*

**definition** *normFace* :: *face*  $\Rightarrow$  *vertex list* **where**  
*normFace*  $\equiv$   $\lambda f. verticesFrom f (minVertex f)$

**definition** *normFaces* :: *face list*  $\Rightarrow$  *vertex list list* **where**  
*normFaces fl*  $\equiv$  *map normFace fl*

**lemma** *normFaces-distinct*: *distinct (normFaces fl)*  $\implies$  *distinct fl*  
*<proof>*

### 13.2 Minimal (plane) graph properties

**definition** *minGraphProps'* :: *graph*  $\Rightarrow$  *bool* **where**  
*minGraphProps' g*  $\equiv$   $\forall f \in \mathcal{F} g. 2 < |vertices f| \wedge distinct (vertices f)$

**definition** *edges-sym* :: *graph*  $\Rightarrow$  *bool* **where**  
*edges-sym g*  $\equiv$   $\forall a b. (a,b) \in edges g \implies (b,a) \in edges g$

**definition** *faceListAt-len* :: *graph*  $\Rightarrow$  *bool* **where**  
*faceListAt-len g*  $\equiv$   $(length (faceListAt g) = countVertices g)$

**definition** *facesAt-eq* :: *graph*  $\Rightarrow$  *bool* **where**  
*facesAt-eq g*  $\equiv$   $\forall v \in \mathcal{V} g. set(facesAt g v) = \{f. f \in \mathcal{F} g \wedge v \in \mathcal{V} f\}$



**definition** *facesAt-distinct* :: *graph*  $\Rightarrow$  *bool* **where**  
*facesAt-distinct*  $g \equiv \forall v \in \mathcal{V} g. \text{distinct} (\text{normFaces} (\text{facesAt } g \ v))$

**definition** *faces-distinct* :: *graph*  $\Rightarrow$  *bool* **where**  
*faces-distinct*  $g \equiv \text{distinct} (\text{normFaces} (\text{faces } g))$

**definition** *faces-subset* :: *graph*  $\Rightarrow$  *bool* **where**  
*faces-subset*  $g \equiv \forall f \in \mathcal{F} g. \mathcal{V} f \subseteq \mathcal{V} g$

**definition** *edges-disj* :: *graph*  $\Rightarrow$  *bool* **where**  
*edges-disj*  $g \equiv$   
 $\forall f \in \mathcal{F} g. \forall f' \in \mathcal{F} g. f \neq f' \longrightarrow \mathcal{E} f \cap \mathcal{E} f' = \{\}$

**definition** *face-face-op* :: *graph*  $\Rightarrow$  *bool* **where**  
*face-face-op*  $g \equiv |\text{faces } g| \neq 2 \longrightarrow$   
 $(\forall f \in \mathcal{F} g. \forall f' \in \mathcal{F} g. f \neq f' \longrightarrow \mathcal{E} f \neq (\mathcal{E} f')^{-1})$

**definition** *one-final-but* :: *graph*  $\Rightarrow$  (*vertex*  $\times$  *vertex*)*set*  $\Rightarrow$  *bool* **where**  
*one-final-but*  $g \ E \equiv$   
 $\forall f \in \mathcal{F} g. \neg \text{final } f \longrightarrow$   
 $(\forall (a,b) \in \mathcal{E} f - E. (b,a) : E \vee (\exists f' \in \mathcal{F} g. \text{final } f' \wedge (b,a) \in \mathcal{E} f'))$

**definition** *one-final* :: *graph*  $\Rightarrow$  *bool* **where**  
*one-final*  $g \equiv \text{one-final-but } g \ \{\}$

**definition** *minGraphProps* :: *graph*  $\Rightarrow$  *bool* **where**  
*minGraphProps*  $g \equiv \text{minGraphProps}' g \wedge \text{facesAt-eq } g \wedge \text{faceListAt-len } g \wedge \text{facesAt-distinct } g$   
 $\wedge \text{faces-distinct } g \wedge \text{faces-subset } g \wedge \text{edges-sym } g \wedge \text{edges-disj } g \wedge \text{face-face-op } g$

**definition** *inv* :: *graph*  $\Rightarrow$  *bool* **where**  
*inv*  $g \equiv \text{minGraphProps } g \wedge \text{one-final } g \wedge |\text{faces } g| \geq 2$

**lemma** *facesAt-distinctI*:  
 $(\bigwedge v. v \in \mathcal{V} g \implies \text{distinct} (\text{normFaces} (\text{facesAt } g \ v))) \implies \text{facesAt-distinct } g$   
 $\langle \text{proof} \rangle$

**lemma** *minGraphProps2*:  
 $\text{minGraphProps } g \implies f \in \mathcal{F} g \implies 2 < |\text{vertices } f|$   
 $\langle \text{proof} \rangle$

**lemma** *mgp-vertices3*:  
 $\text{minGraphProps } g \implies f \in \mathcal{F} g \implies |\text{vertices } f| \geq 3$   
 $\langle \text{proof} \rangle$

**lemma** *mgp-vertices-nonempty*:  
 $\text{minGraphProps } g \implies f \in \mathcal{F} g \implies \text{vertices } f \neq []$

$\langle \text{proof} \rangle$

**lemma** *minGraphProps3*:

$\text{minGraphProps } g \implies f \in \mathcal{F} g \implies \text{distinct } (\text{vertices } f)$

$\langle \text{proof} \rangle$

**lemma** *minGraphProps4*:

$\text{minGraphProps } g \implies (\text{length } (\text{faceListAt } g) = \text{countVertices } g)$

$\langle \text{proof} \rangle$

**lemma** *minGraphProps5*:

$\llbracket \text{minGraphProps } g; v : \mathcal{V} g; f \in \text{set } (\text{facesAt } g v) \rrbracket \implies f \in \mathcal{F} g$

$\langle \text{proof} \rangle$

**lemma** *minGraphProps6*:

$\text{minGraphProps } g \implies v : \mathcal{V} g \implies f \in \text{set } (\text{facesAt } g v) \implies v \in \mathcal{V} f$

$\langle \text{proof} \rangle$

**lemma** *minGraphProps9*:

$\text{minGraphProps } g \implies f \in \mathcal{F} g \implies v \in \mathcal{V} f \implies v \in \mathcal{V} g$

$\langle \text{proof} \rangle$

**lemma** *minGraphProps7*:

$\text{minGraphProps } g \implies f \in \mathcal{F} g \implies v \in \mathcal{V} f \implies f \in \text{set } (\text{facesAt } g v)$

$\langle \text{proof} \rangle$

**lemma** *minGraphProps-facesAt-eq*:  $\text{minGraphProps } g \implies$

$v \in \mathcal{V} g \implies \text{set } (\text{facesAt } g v) = \{f \in \mathcal{F} g. v \in \mathcal{V} f\}$

$\langle \text{proof} \rangle$

**lemma** *mgp-dist-facesAt[simp]*:

$\text{minGraphProps } g \implies v : \mathcal{V} g \implies \text{distinct } (\text{facesAt } g v)$

$\langle \text{proof} \rangle$

**lemma** *minGraphProps8*:

$\text{minGraphProps } g \implies v : \mathcal{V} g \implies \text{distinct } (\text{normFaces } (\text{facesAt } g v))$

$\langle \text{proof} \rangle$

**lemma** *minGraphProps8a*:

$\text{minGraphProps } g \implies v \in \mathcal{V} g \implies \text{distinct } (\text{normFaces } (\text{faceListAt } g ! v))$

$\langle \text{proof} \rangle$

**lemma** *minGraphProps8a'*:  $\text{minGraphProps } g \implies$

$v < \text{countVertices } g \implies \text{distinct } (\text{normFaces } (\text{faceListAt } g ! v))$

$\langle \text{proof} \rangle$

**lemma** *minGraphProps9'*:

$\text{minGraphProps } g \implies f \in \mathcal{F} \ g \implies v \in \mathcal{V} \ f \implies v < \text{countVertices } g$   
(proof)

**lemma** *minGraphProps10*:

$\text{minGraphProps } g \implies (a, b) \in \text{edges } g \implies (b, a) \in \text{edges } g$   
(proof)

**lemma** *minGraphProps11*:

$\text{minGraphProps } g \implies \text{distinct } (\text{normFaces } (\text{faces } g))$   
(proof)

**lemma** *minGraphProps11'*:

$\text{minGraphProps } g \implies \text{distinct } (\text{faces } g)$   
(proof)

**lemma** *minGraphProps12*:

$\text{minGraphProps } g \implies f \in \mathcal{F} \ g \implies (a, b) \in \mathcal{E} \ f \implies (b, a) \notin \mathcal{E} \ f$   
(proof)

**lemma** *minGraphProps7'*:  $\text{minGraphProps } g \implies$

$f \in \mathcal{F} \ g \implies v \in \mathcal{V} \ f \implies f \in \text{set } (\text{faceListAt } g \ ! \ v)$   
(proof)

**lemma** *mgp-edges-disj*:

$\llbracket \text{minGraphProps } g; f \neq f'; f \in \mathcal{F} \ g; f' \in \mathcal{F} \ g \rrbracket \implies$   
 $uv \in \mathcal{E} \ f \implies uv \notin \mathcal{E} \ f'$   
(proof)

**lemma** *one-final-but-antimono*:

$\text{one-final-but } g \ E \implies E \subseteq E' \implies \text{one-final-but } g \ E'$   
(proof)

**lemma** *one-final-antimono*:  $\text{one-final } g \implies \text{one-final-but } g \ E$

(proof)

**lemma** *inv-two-faces*:  $\text{inv } g \implies |\text{faces } g| \geq 2$

(proof)

**lemma** *inv-mgp[simp]*:  $\text{inv } g \implies \text{minGraphProps } g$

(proof)

**lemma** *makeFaceFinal-id[simp]*:  $\text{final } f \implies \text{makeFaceFinal } f \ g = g$

(proof)

**lemma** *inv-one-finalD'*:

$\llbracket \text{inv } g; f \in \mathcal{F} \ g; \neg \text{final } f; (a,b) \in \mathcal{E} \ f \rrbracket \implies$   
 $\exists f' \in \mathcal{F} \ g. \text{final } f' \wedge f' \neq f \wedge (b,a) \in \mathcal{E} \ f'$   
 $\langle \text{proof} \rangle$

**lemmas** *minGraphProps* =

*minGraphProps2 minGraphProps3 minGraphProps4*  
*minGraphProps5 minGraphProps6 minGraphProps7 minGraphProps8*  
*minGraphProps9*

**lemma** *mgp-no-loop[simp]*:

$\text{minGraphProps } g \implies f \in \mathcal{F} \ g \implies v \in \mathcal{V} \ f \implies f \cdot v \neq v$   
 $\langle \text{proof} \rangle$

**lemma** *mgp-facesAt-no-loop*:

$\text{minGraphProps } g \implies v : \mathcal{V} \ g \implies f \in \text{set}(\text{facesAt } g \ v) \implies f \cdot v \neq v$   
 $\langle \text{proof} \rangle$

**lemma** *edge-pres-faceAt*:

$\llbracket \text{minGraphProps } g; u : \mathcal{V} \ g; f \in \text{set}(\text{facesAt } g \ u); (u,v) \in \mathcal{E} \ f \rrbracket \implies$   
 $f \in \text{set}(\text{facesAt } g \ v)$   
 $\langle \text{proof} \rangle$

**lemma** *in-facesAt-nextVertex*:

$\text{minGraphProps } g \implies v : \mathcal{V} \ g \implies f \in \text{set}(\text{facesAt } g \ v) \implies f \in \text{set}(\text{facesAt } g \ (f \cdot v))$   
 $\langle \text{proof} \rangle$

**lemma** *mgp-edge-face-ex*:

**assumes** *[intro]*:  $\text{minGraphProps } g \ v : \mathcal{V} \ g$   
**and** *fv*:  $f \in \text{set}(\text{facesAt } g \ v)$  **and** *uv*:  $(u,v) \in \mathcal{E} \ f$   
**shows**  $\exists f' \in \text{set}(\text{facesAt } g \ v). (v,u) \in \mathcal{E} \ f'$   
 $\langle \text{proof} \rangle$

**lemma** *nextVertex-in-graph*:

$\text{minGraphProps } g \implies v : \mathcal{V} \ g \implies f \in \text{set}(\text{facesAt } g \ v) \implies f \cdot v : \mathcal{V} \ g$   
 $\langle \text{proof} \rangle$

**lemma** *mgp-nextVertex-face-ex2*:

**assumes** *mgp[intro]*:  $\text{minGraphProps } g \ v : \mathcal{V} \ g$  **and** *f*:  $f \in \text{set}(\text{facesAt } g \ v)$   
**shows**  $\exists f' \in \text{set}(\text{facesAt } g \ (f \cdot v)). f' \cdot (f \cdot v) = v$   
 $\langle \text{proof} \rangle$

**lemma** *inv-finals-nonempty*:  $\text{inv } g \implies \text{finals } g \neq []$

$\langle \text{proof} \rangle$

### 13.3 containsDuplicateEdge

**definition**

containsUnacceptableEdgeSnd' :: (nat ⇒ nat ⇒ bool) ⇒ nat list ⇒ bool **where**  
containsUnacceptableEdgeSnd' N is ≡  
(∃ k < |is| - 2. let i0 = is!k; i1 = is!(k+1); i2 = is!(k+2) in  
N i1 i2 ∧ (i0 < i1) ∧ (i1 < i2))

**lemma** containsUnacceptableEdgeSnd-eq:

containsUnacceptableEdgeSnd N v is = containsUnacceptableEdgeSnd' N (v#is)  
⟨proof⟩

**lemma** containsDuplicateEdge-eq1:

containsDuplicateEdge g f v is = containsDuplicateEdge' g f v is  
⟨proof⟩

**lemma** containsDuplicateEdge-eq:

containsDuplicateEdge = containsDuplicateEdge'  
⟨proof⟩

**declare** Nat.diff-is-0-eq' [simp del]

### 13.4 replacefacesAt

**primrec** replacefacesAt2 ::

nat list ⇒ face ⇒ face list ⇒ face list list ⇒ face list list **where**  
replacefacesAt2 [] f fs F = F |  
replacefacesAt2 (n#ns) f fs F =  
(if n < |F|  
then replacefacesAt2 ns f fs (F [n:=replace f fs (F!n)])  
else replacefacesAt2 ns f fs F)

**lemma** replacefacesAt-eq[THEN eq-reflection]:

replacefacesAt ns oldf newfs F = replacefacesAt2 ns oldf newfs F  
⟨proof⟩

**lemma** replacefacesAt2-notin:

i ∉ set is ⇒ (replacefacesAt2 is oldf newFs Fss)!i = Fss!i  
⟨proof⟩

**lemma** replacefacesAt2-in:

i ∈ set is ⇒ distinct is ⇒ i < |Fss| ⇒  
(replacefacesAt2 is oldf newFs Fss)!i = replace oldf newFs (Fss !i)  
⟨proof⟩

**lemma** *distinct-replacefacesAt21:*

$i < |Fss| \implies i \in \text{set } is \implies \text{distinct } is \implies \text{distinct } (Fss!i) \implies \text{distinct } \text{newFs}$   
 $\implies$   
 $\text{set } (Fss ! i) \cap \text{set } \text{newFs} \subseteq \{\text{olfF}\} \implies$   
 $\text{distinct } ((\text{replacefacesAt2 } is \text{ olfF } \text{newFs } Fss)! i)$   
 $\langle \text{proof} \rangle$

**lemma** *distinct-replacefacesAt22:*

$i < |Fss| \implies i \notin \text{set } is \implies \text{distinct } is \implies \text{distinct } (Fss!i) \implies \text{distinct } \text{newFs}$   
 $\implies$   
 $\text{set } (Fss ! i) \cap \text{set } \text{newFs} \subseteq \{\text{olfF}\} \implies$   
 $\text{distinct } ((\text{replacefacesAt2 } is \text{ olfF } \text{newFs } Fss)! i)$   
 $\langle \text{proof} \rangle$

**lemma** *distinct-replacefacesAt2-2:*

$i < |Fss| \implies \text{distinct } is \implies \text{distinct } (Fss!i) \implies \text{distinct } \text{newFs} \implies$   
 $\text{set } (Fss ! i) \cap \text{set } \text{newFs} \subseteq \{\text{olfF}\} \implies$   
 $\text{distinct } ((\text{replacefacesAt2 } is \text{ olfF } \text{newFs } Fss)! i)$   
 $\langle \text{proof} \rangle$

**lemma** *replacefacesAt2-nth1:*

$k \notin \text{set } ns \implies (\text{replacefacesAt2 } ns \text{ oldf } \text{newfs } F) ! k = F ! k$   
 $\langle \text{proof} \rangle$

**lemma** *replacefacesAt2-nth1':*  $k \in \text{set } ns \implies k < |F| \implies \text{distinct } ns \implies$   
 $(\text{replacefacesAt2 } ns \text{ oldf } \text{newfs } F) ! k = (\text{replace oldf } \text{newfs } (F!k))$   
 $\langle \text{proof} \rangle$

**lemma** *replacefacesAt2-nth2:*  $k < |F| \implies$

$(\text{replacefacesAt2 } [k] \text{ oldf } \text{newfs } F) ! k = \text{replace oldf } \text{newfs } (F!k)$   
 $\langle \text{proof} \rangle$

**lemma** *replacefacesAt2-length[simp]:*

$|\text{replacefacesAt2 } nvs \text{ f' f'' } vs| = |vs|$   
 $\langle \text{proof} \rangle$

**lemma** *replacefacesAt2-nth:*  $k \in \text{set } ns \implies k < |F| \implies \text{oldf} \notin \text{set } \text{newfs} \implies$   
 $\text{distinct } (F!k) \implies \text{distinct } \text{newfs} \implies \text{oldf} \in \text{set } (F!k) \longrightarrow \text{set } \text{newfs} \cap \text{set } (F!k)$   
 $\subseteq \{\text{oldf}\} \implies$   
 $(\text{replacefacesAt2 } ns \text{ oldf } \text{newfs } F) ! k = (\text{replace oldf } \text{newfs } (F!k))$   
 $\langle \text{proof} \rangle$

**lemma** *replacefacesAt-notin:*

$i \notin \text{set } is \implies (\text{replacefacesAt } is \text{ olfF } \text{newFs } Fss)!i = Fss!i$   
 $\langle \text{proof} \rangle$

**lemma** *replacefacesAt-in:*

$i \in \text{set } is \implies \text{distinct } is \implies i < |Fss| \implies$   
 $(\text{replacefacesAt } is \text{ oldf newFs } Fss)!i = \text{replace oldf newFs } (Fss !i)$   
 <proof>

**lemma** *replacefacesAt-length[simp]*:  $|\text{replacefacesAt } nvs f' [f''] vs| = |vs|$   
 <proof>

**lemma** *replacefacesAt-nth2*:  $k < |F| \implies$   
 $(\text{replacefacesAt } [k] \text{ oldf newFs } F) !k = \text{replace oldf newFs } (F!k)$   
 <proof>

**lemma** *replacefacesAt-nth*:  $k \in \text{set } ns \implies k < |F| \implies \text{oldf} \notin \text{set } \text{newFs} \implies$   
 $\text{distinct } (F!k) \implies \text{distinct } \text{newFs} \implies \text{oldf} \in \text{set } (F!k) \longrightarrow \text{set } \text{newFs} \cap \text{set } (F!k)$   
 $\subseteq \{\text{oldf}\} \implies$   
 $(\text{replacefacesAt } ns \text{ oldf newFs } F) !k = (\text{replace oldf newFs } (F!k))$   
 <proof>

**lemma** *replacefacesAt2-5*:  $x \in \text{set } (\text{replacefacesAt2 } ns \text{ oldf newFs } F !k) \implies x \in$   
 $\text{set } (F!k) \vee x \in \text{set } \text{newFs}$   
 <proof>

**lemma** *replacefacesAt-Nil[simp]*:  $\text{replacefacesAt } [] f fs F = F$   
 <proof>

**lemma** *replacefacesAt-Cons[simp]*:  
 $\text{replacefacesAt } (n \# ns) f fs F =$   
 $(\text{if } n < |F| \text{ then } \text{replacefacesAt } ns f fs (F[n := \text{replace } f fs (F!n)])$   
 $\text{else } \text{replacefacesAt } ns f fs F)$   
 <proof>

**lemmas** *replacefacesAt-simps* = *replacefacesAt-Nil replacefacesAt-Cons*

**lemma** *len-nth-repAt[simp]*:  
 $\bigwedge xs. i < |xs| \implies |\text{replacefacesAt } is x [y] xs ! i| = |xs!i|$   
 <proof>

### 13.5 normFace

**lemma** *minVertex-in*:  $\text{vertices } f \neq [] \implies \text{minVertex } f \in \mathcal{V} f$   
 <proof>

**lemma** *minVertex-eq-if-vertices-eq*:  
 $\mathcal{V} f = \mathcal{V} f' \implies \text{minVertex } f = \text{minVertex } f'$   
 <proof>

**lemma** *normFace-replace-in*:

$normFace\ a \in set\ (normFaces\ (replace\ oldF\ newFs\ fs)) \implies$   
 $normFace\ a \in set\ (normFaces\ newFs) \vee normFace\ a \in set\ (normFaces\ fs)$   
(proof)

**lemma** *distinct-replace-norm*:

$distinct\ (normFaces\ fs) \implies distinct\ (normFaces\ newFs) \implies$   
 $set\ (normFaces\ fs) \cap set\ (normFaces\ newFs) \subseteq \{\}$   $\implies distinct\ (normFaces$   
 $(replace\ oldF\ newFs\ fs))$   
(proof)

**lemma** *distinct-replacefacesAt1-norm*:

$i < |Fss| \implies i \in set\ is \implies distinct\ is \implies distinct\ (normFaces\ (Fss!i)) \implies$   
 $distinct\ (normFaces\ newFs) \implies$   
 $set\ (normFaces\ (Fss\ !\ i)) \cap set\ (normFaces\ newFs) \subseteq \{\} \implies$   
 $distinct\ (normFaces\ ((replacefacesAt\ is\ oldF\ newFs\ Fss)!i))$   
(proof)

**lemma** *distinct-replacefacesAt2-norm*:

$i < |Fss| \implies i \notin set\ is \implies distinct\ is \implies distinct\ (normFaces\ (Fss!i)) \implies$   
 $distinct\ (normFaces\ newFs) \implies$   
 $set\ (normFaces\ (Fss\ !\ i)) \cap set\ (normFaces\ newFs) \subseteq \{\} \implies$   
 $distinct\ (normFaces\ ((replacefacesAt\ is\ oldF\ newFs\ Fss)!i))$   
(proof)

**lemma** *distinct-replacefacesAt-norm*:

$i < |Fss| \implies distinct\ is \implies distinct\ (normFaces\ (Fss!i)) \implies distinct\ (normFaces$   
 $newFs) \implies$   
 $set\ (normFaces\ (Fss\ !\ i)) \cap set\ (normFaces\ newFs) \subseteq \{\} \implies$   
 $distinct\ (normFaces\ ((replacefacesAt\ is\ oldF\ newFs\ Fss)!i))$   
(proof)

**lemma** *normFace-in-cong*:

$vertices\ f \neq [] \implies minGraphProps\ g \implies normFace\ f \in set\ (normFaces\ (faces$   
 $g))$   
 $\implies \exists f' \in set\ (faces\ g). f \cong f'$   
(proof)

**lemma** *normFace-neq*:

$a \in \mathcal{V}\ f \implies a \notin \mathcal{V}\ f' \implies vertices\ f' \neq [] \implies normFace\ f \neq normFace\ f'$   
(proof)

**lemma** *split-face-f12-f21-neq-norm*:

$pre-split-face\ oldF\ ram1\ ram2\ vs \implies$   
 $2 < |vertices\ oldF| \implies 2 < |vertices\ f12| \implies 2 < |vertices\ f21| \implies$



$(f12, f21) = \text{split-face } \text{oldF } \text{ram1 } \text{ram2 } \text{vs} \implies \text{normFace } f12 \neq \text{normFace } f21$   
 ⟨proof⟩

**lemma** *normFace-in*:  $f \in \text{set } \text{fs} \implies \text{normFace } f \in \text{set } (\text{normFaces } \text{fs})$   
 ⟨proof⟩

### 13.6 Invariants of *splitFace*

**lemma** *splitFace-holds-minGraphProps'*:  
 $\text{pre-splitFace } g' \ v \ a \ f' \ \text{vs} \implies \text{minGraphProps}' \ g' \implies$   
 $\text{minGraphProps}' \ (\text{snd } (\text{snd } (\text{splitFace } g' \ v \ a \ f' \ \text{vs})))$   
 ⟨proof⟩

**lemma** *splitFace-holds-faceListAt-len*:  
 $\text{pre-splitFace } g' \ v \ a \ f' \ \text{vs} \implies \text{minGraphProps } g' \implies$   
 $\text{faceListAt-len } (\text{snd } (\text{snd } (\text{splitFace } g' \ v \ a \ f' \ \text{vs})))$   
 ⟨proof⟩

**lemma** *splitFace-new-f12*:  
**assumes** *pre*:  $\text{pre-splitFace } g \ \text{ram1 } \ \text{ram2 } \ \text{oldF } \ \text{newVs}$   
**and** *props*:  $\text{minGraphProps } g$   
**and** *spl*:  $(f12, f21, \text{newGraph}) = \text{splitFace } g \ \text{ram1 } \ \text{ram2 } \ \text{oldF } \ \text{newVs}$   
**shows**  $f12 \notin \mathcal{F} \ g$   
 ⟨proof⟩

**lemma** *splitFace-new-f12-norm*:  
**assumes** *pre*:  $\text{pre-splitFace } g \ \text{ram1 } \ \text{ram2 } \ \text{oldF } \ \text{newVs}$   
**and** *props*:  $\text{minGraphProps } g$   
**and** *spl*:  $(f12, f21, \text{newGraph}) = \text{splitFace } g \ \text{ram1 } \ \text{ram2 } \ \text{oldF } \ \text{newVs}$   
**shows**  $\text{normFace } f12 \notin \text{set } (\text{normFaces } (\text{faces } g))$   
 ⟨proof⟩

**lemma** *splitFace-new-f21*:  
**assumes** *pre*:  $\text{pre-splitFace } g \ \text{ram1 } \ \text{ram2 } \ \text{oldF } \ \text{newVs}$   
**and** *props*:  $\text{minGraphProps } g$   
**and** *spl*:  $(f12, f21, \text{newGraph}) = \text{splitFace } g \ \text{ram1 } \ \text{ram2 } \ \text{oldF } \ \text{newVs}$   
**shows**  $f21 \notin \mathcal{F} \ g$   
 ⟨proof⟩

**lemma** *splitFace-new-f21-norm*:  
**assumes** *pre*:  $\text{pre-splitFace } g \ \text{ram1 } \ \text{ram2 } \ \text{oldF } \ \text{newVs}$   
**and** *props*:  $\text{minGraphProps } g$   
**and** *spl*:  $(f12, f21, \text{newGraph}) = \text{splitFace } g \ \text{ram1 } \ \text{ram2 } \ \text{oldF } \ \text{newVs}$   
**shows**  $\text{normFace } f21 \notin \text{set } (\text{normFaces } (\text{faces } g))$   
 ⟨proof⟩

**lemma** *splitFace-f21-oldF-neq*:  
*pre-splitFace g ram1 ram2 oldF newVs*  $\implies$   
*minGraphProps g*  $\implies$   
*(f12, f21, newGraph) = splitFace g ram1 ram2 oldF newVs*  $\implies$   
*oldF*  $\neq$  *f21*  
<proof>

**lemma** *splitFace-f12-oldF-neq*:  
*pre-splitFace g ram1 ram2 oldF newVs*  $\implies$   
*minGraphProps g*  $\implies$   
*(f12, f21, newGraph) = splitFace g ram1 ram2 oldF newVs*  $\implies$   
*oldF*  $\neq$  *f12*  
<proof>

**lemma** *splitFace-f12-f21-neq-norm*:  
*pre-splitFace g ram1 ram2 oldF vs*  $\implies$  *minGraphProps g*  $\implies$   
*(f12, f21, newGraph) = splitFace g ram1 ram2 oldF vs*  $\implies$   
*normFace f12*  $\neq$  *normFace f21*  
<proof>

**lemma** *set-faces-splitFace*:  
 $\llbracket$  *minGraphProps g*;  $f \in \mathcal{F} g$ ; *pre-splitFace g v1 v2 f vs*;  
*(f1, f2, g') = splitFace g v1 v2 f vs*  $\rrbracket$   
 $\implies \mathcal{F} g' = \{f1, f2\} \cup (\mathcal{F} g - \{f\})$   
<proof>

**declare** *minGraphProps8 minGraphProps8a minGraphProps8a'* [intro]

**lemma** *splitFace-holds-facesAt-distinct*:  
**assumes** *pre*: *pre-splitFace g v w f* [*countVertices g*..*countVertices g + n*]  
**and** *mgp*: *minGraphProps g*  
**shows** *facesAt-distinct* (*snd* (*snd* (*splitFace g v w f* [*countVertices g*..*countVertices g + n*]))))  
<proof>

**lemma** *splitFace-holds-facesAt-eq*:  
**assumes** *pre-F*: *pre-splitFace g' v a f'* [*countVertices g'*..*countVertices g' + n*]  
**and** *mgp*: *minGraphProps g'*  
**and** *g''*: *g'' = (snd (snd (splitFace g' v a f' [*countVertices g'*..*countVertices g' + n*]))))  
**shows** *facesAt-eq g''*  
<proof>*

**lemma** *splitFace-holds-faces-subset*:  
**assumes** *pre-F*: *pre-splitFace g' v a f'* [*countVertices g'*..*countVertices g' + n*]

**and** *mgp*: *minGraphProps g'*  
**shows** *faces-subset* (*snd* (*snd* (*splitFace g' v a f' [countVertices g'..<countVertices g' + n]*)))  
 ⟨*proof*⟩

**lemma** *splitFace-holds-edges-sym*:  
**assumes** *pre-F*: *pre-splitFace g' v a f' ws*  
**and** *mgp*: *minGraphProps g'*  
**shows** *edges-sym* (*snd* (*snd* (*splitFace g' v a f' ws*)))  
 ⟨*proof*⟩

**lemma** *splitFace-holds-faces-distinct*:  
**assumes** *pre-F*: *pre-splitFace g' v a f' [countVertices g'..<countVertices g' + n]*  
**and** *mgp*: *minGraphProps g'*  
**shows** *faces-distinct* (*snd* (*snd* (*splitFace g' v a f' [countVertices g'..<countVertices g' + n]*)))  
 ⟨*proof*⟩

**lemma** *help*:  
**shows**  $xs \neq [] \implies x \notin \text{set } xs \implies x \neq \text{hd } xs$  **and**  
 $xs \neq [] \implies x \notin \text{set } xs \implies x \neq \text{last } xs$  **and**  
 $xs \neq [] \implies x \notin \text{set } xs \implies \text{hd } xs \neq x$  **and**  
 $xs \neq [] \implies x \notin \text{set } xs \implies \text{last } xs \neq x$   
 ⟨*proof*⟩

**lemma** *split-face-edge-disj*:  
 $\llbracket \text{pre-split-face } f \text{ a b vs; } (f_1, f_2) = \text{split-face } f \text{ a b vs; } |\text{vertices } f| \geq 3; \text{ vs} = [] \longrightarrow (a,b) \notin \text{edges } f \wedge (b,a) \notin \text{edges } f \rrbracket$   
 $\implies \mathcal{E} f_1 \cap \mathcal{E} f_2 = \{\}$   
 ⟨*proof*⟩

**lemma** *splitFace-edge-disj*:  
**assumes** *mgp*: *minGraphProps g* **and** *pre*: *pre-splitFace g u v f vs*  
**and** *FDG*:  $(f_1, f_2, g') = \text{splitFace } g \text{ u v f vs}$   
**shows** *edges-disj g'*  
 ⟨*proof*⟩

**lemma** *splitFace-edges-disj2*:  
 $\text{minGraphProps } g \implies \text{pre-splitFace } g \text{ u v f vs}$   
 $\implies \text{edges-disj}(\text{snd}(\text{snd}(\text{splitFace } g \text{ u v f vs})))$   
 ⟨*proof*⟩

**lemma** *vertices-conv-Union-edges2*:  
 $\text{distinct}(\text{vertices } f) \implies \mathcal{V}(f::\text{face}) = (\bigcup (a,b) \in \mathcal{E} f. \{b\})$

$\langle proof \rangle$

**lemma** *splitFace-face-face-op*:

**assumes** *mgp*: *minGraphProps* *g* **and** *pre*: *pre-splitFace* *g* *u* *v* *f* *vs*

**and** *fdg*:  $(f_1, f_2, g') = \text{splitFace } g \ u \ v \ f \ vs$

**shows** *face-face-op* *g'*

$\langle proof \rangle$

**lemma** *splitFace-face-face-op2*:

*minGraphProps* *g*  $\implies$  *pre-splitFace* *g* *u* *v* *f* *vs*

$\implies$  *face-face-op*(*snd*(*snd*(*splitFace* *g* *u* *v* *f* *vs*)))

$\langle proof \rangle$

**lemma** *splitFace-holds-minGraphProps*:

**assumes** *precond*: *pre-splitFace* *g'* *v* *a* *f'* [*countVertices* *g'*..*countVertices* *g'* + *n*]

**and** *min*: *minGraphProps* *g'*

**shows** *minGraphProps* (*snd* (*snd* (*splitFace* *g'* *v* *a* *f'* [*countVertices* *g'*..*countVertices* *g'* + *n*])))

$\langle proof \rangle$

### 13.7 Invariants of *makeFaceFinal*

**lemma** *MakeFaceFinal-minGraphProps'*:

$f \in \mathcal{F} \ g \implies \text{minGraphProps } g \implies \text{minGraphProps}' (\text{makeFaceFinal } f \ g)$

$\langle proof \rangle$

**lemma** *MakeFaceFinal-facesAt-eq*:

$f \in \mathcal{F} \ g \implies \text{minGraphProps } g \implies \text{facesAt-eq } (\text{makeFaceFinal } f \ g)$

$\langle proof \rangle$

**lemma** *MakeFaceFinal-faceListAt-len*:

$f \in \mathcal{F} \ g \implies \text{minGraphProps } g \implies \text{faceListAt-len } (\text{makeFaceFinal } f \ g)$

$\langle proof \rangle$

**lemma** *normFaces-makeFaceFinalFaceList*: (*normFaces* (*makeFaceFinalFaceList* *f* *fs*)) = (*normFaces* *fs*)

$\langle proof \rangle$

**lemma** *MakeFaceFinal-facesAt-distinct*:

$f \in \mathcal{F} \ g \implies \text{minGraphProps } g \implies \text{facesAt-distinct } (\text{makeFaceFinal } f \ g)$

$\langle proof \rangle$

**lemma** *MakeFaceFinal-faces-subset*:

$f \in \mathcal{F} \ g \implies \text{minGraphProps } g \implies \text{faces-subset } (\text{makeFaceFinal } f \ g)$

$\langle proof \rangle$

**lemma** *MakeFaceFinal-edges-sym*:

$f \in \mathcal{F} \ g \implies \text{minGraphProps } g \implies \text{edges-sym } (\text{makeFaceFinal } f \ g)$

$\langle \text{proof} \rangle$

**lemma** *MakeFaceFinal-faces-distinct:*

$f \in \mathcal{F} \ g \implies \text{minGraphProps } g \implies \text{faces-distinct } (\text{makeFaceFinal } f \ g)$   
 $\langle \text{proof} \rangle$

**lemma** *MakeFaceFinal-edges-disj:*

$f \in \mathcal{F} \ g \implies \text{minGraphProps } g \implies \text{edges-disj } (\text{makeFaceFinal } f \ g)$   
 $\langle \text{proof} \rangle$

**lemma** *MakeFaceFinal-face-face-op:*

$f \in \mathcal{F} \ g \implies \text{minGraphProps } g \implies \text{face-face-op } (\text{makeFaceFinal } f \ g)$   
 $\langle \text{proof} \rangle$

**lemma** *MakeFaceFinal-minGraphProps:*

$f \in \mathcal{F} \ g \implies \text{minGraphProps } g \implies \text{minGraphProps } (\text{makeFaceFinal } f \ g)$   
 $\langle \text{proof} \rangle$

### 13.8 Invariants of *subdivFace'*

**lemma** *subdivFace'-holds-minGraphProps:*  $\bigwedge f \ v' \ v \ n \ g.$

$\text{pre-subdivFace}' \ g \ f \ v' \ v \ n \ \text{ovl} \implies f \in \mathcal{F} \ g \implies$   
 $\text{minGraphProps } g \implies \text{minGraphProps } (\text{subdivFace}' \ g \ f \ v \ n \ \text{ovl})$   
 $\langle \text{proof} \rangle$

**abbreviation** (*input*)

$\text{Edges-if} :: \text{face} \Rightarrow \text{vertex} \Rightarrow \text{vertex} \Rightarrow (\text{vertex} \times \text{vertex})\text{set}$  **where**  
 $\text{Edges-if } f \ u \ v ==$   
 $\text{if } u=v \text{ then } \{\} \text{ else } \text{Edges}(u \ \# \ \text{between } (\text{vertices } f) \ u \ v \ @ \ [v])$

**lemma** *FaceDivisionGraph-one-final-but:*

**assumes** *mgp:*  $\text{minGraphProps } g$  **and** *pre:*  $\text{pre-splitFace } g \ u \ v \ f \ vs$

**and** *fdg:*  $(f_1, f_2, g') = \text{splitFace } g \ u \ v \ f \ vs$

**and** *nrv:*  $r \neq v$

**and** *ruv:*  $\text{before } (\text{verticesFrom } f \ r) \ u \ v$  **and** *rf:*  $r \in \mathcal{V} \ f$

**and** *1:*  $\text{one-final-but } g \ (\text{Edges-if } f \ r \ u)$

**shows**  $\text{one-final-but } g' \ (\text{Edges}(r \ \# \ \text{between } (\text{vertices } f_2) \ r \ v \ @ \ [v]))$

$\langle \text{proof} \rangle$

**lemma** *one-final-but-makeFaceFinal:*

$\llbracket \text{minGraphProps } g; \text{one-final-but } g \ E; E \subseteq \mathcal{E} \ f; f \in \mathcal{F} \ g; \neg \text{final } f \rrbracket \implies$   
 $\text{one-final } (\text{makeFaceFinal } f \ g)$   
 $\langle \text{proof} \rangle$

**lemma** *one-final-subdivFace'*:

$\bigwedge f v n g.$   
 $pre-subdivFace' g f u v n ovs \implies minGraphProps g \implies f \in \mathcal{F} g \implies$   
 $one-final-but g (Edges-if f u v) \implies$   
 $one-final(subdivFace' g f v n ovs)$   
(proof)

**lemma** *neighbors-edges*:

$minGraphProps g \implies a : \mathcal{V} g \implies b \in set (neighbors g a) = ((a, b) \in edges g)$   
(proof)

**lemma** *no-self-edges*:  $minGraphProps' g \implies (a, a) \notin edges g$  (proof)

Requires only *distinct* (vertices  $f$ ) and that  $g$  has no self-loops.

**lemma** *duplicateEdge-is-duplicateEdge-eq*:

$minGraphProps g \implies f \in \mathcal{F} g \implies a \in \mathcal{V} f \implies b \in \mathcal{V} f \implies$   
 $duplicateEdge g f a b = is-duplicateEdge g f a b$   
(proof)

**lemma** *incrIndexList-less-eq*:

$incrIndexList ls m nmax \implies Suc n < |ls| \implies ls!n \leq ls!Suc n$   
(proof)

**lemma** *incrIndexList-less*:

$incrIndexList ls m nmax \implies Suc n < |ls| \implies ls!n \neq ls!Suc n \implies ls!n < ls!Suc n$   
(proof)

**lemma** *Seed-holds-minGraphProps'*:  $minGraphProps' (Seed p)$

(proof)

**lemma** *Seed-holds-facesAt-eq*:  $facesAt-eq (Seed p)$

(proof)

**lemma** *minVertex-zero1*:  $minVertex (Face [0..<Suc z] Final) = 0$

(proof)

**lemma** *minVertex-zero2*:  $minVertex (Face (rev [0..<Suc z]) Nonfinal) = 0$

(proof)

### 13.9 Invariants of *Seed*

**lemma** *Seed-holds-facesAt-distinct*:  $facesAt-distinct (Seed p)$

(proof)

**lemma** *Seed-holds-faces-subset*:  $faces-subset (Seed p)$

*<proof>*

**lemma** *Seed-holds-edges-sym: edges-sym (Seed p)*  
*<proof>*

**lemma** *Seed-holds-edges-disj: edges-disj (Seed p)*  
*<proof>*

**lemma** *Seed-holds-faces-distinct: faces-distinct (Seed p)*  
*<proof>*

**lemma** *Seed-holds-faceListAt-len: faceListAt-len (Seed p)*  
*<proof>*

**lemma** *face-face-op-Seed: face-face-op(Seed p)*  
*<proof>*

**lemma** *one-final-Seed: one-final Seed<sub>p</sub>*  
*<proof>*

**lemma** *two-face-Seed: |faces Seed<sub>p</sub>| ≥ 2*  
*<proof>*

**lemma** *inv-Seed: inv (Seed p)*  
*<proof>*

**lemma** *pre-subdivFace-indexToVertexList:*  
**assumes** *mgp: minGraphProps g and f: f ∈ set (nonFinals g)*  
**and** *v: v ∈ V f and e: e ∈ set (enumerator i |vertices f|)*  
**and** *containsNot: ¬ containsDuplicateEdge g f v e and i: 2 < i*  
**shows** *pre-subdivFace g f v (indexToVertexList f v e)*  
*<proof>*

### 13.10 Increasing properties of *subdivFace'*

**lemma** *subdivFace'-incr:*  
**assumes** *Ptrans: ∧x y z. Q x y ⇒ P y z ⇒ P x z*  
**and** *mkFin: ∧f g. f ∈ F g ⇒ ¬ final f ⇒ P g (makeFaceFinal f g)*  
**and** *fdg-incr: ∧g u v f vs.*  
*pre-splitFace g u v f vs ⇒*  
*Q g (snd(snd(splitFace g u v f vs)))*  
**shows**  
*∧f' v n g. pre-subdivFace' g f' v' v n ovl ⇒*  
*minGraphProps g ⇒ f' ∈ F g ⇒ P g (subdivFace' g f' v n ovl)*  
*<proof>*

**lemma** *next-plane0-via-subdivFace'*:

**assumes** *mgp*: *minGraphProps g* **and** *gg'*:  $g \text{ [next-plane0}_p] \rightarrow g'$

**and** *P*:  $\bigwedge f v' v n g \text{ ovs. } \text{minGraphProps } g \implies \text{pre-subdivFace' } g f v' v n \text{ ovs} \implies f \in \mathcal{F} g \implies P g (\text{subdivFace' } g f v n \text{ ovs})$

**shows**  $P g g'$

*<proof>*

**lemma** *next-plane0-incr*:

**assumes** *Ptrans*:  $\bigwedge x y z. Q x y \implies P y z \implies P x z$

**and** *mkFin*:  $\bigwedge f g. f \in \mathcal{F} g \implies \neg \text{final } f \implies P g (\text{makeFaceFinal } f g)$

**and** *fdg-incr*:  $\bigwedge g u v f \text{ vs. } \text{pre-splitFace } g u v f \text{ vs} \implies$

$Q g (\text{snd}(\text{snd}(\text{splitFace } g u v f \text{ vs})))$

**and** *mgp*: *minGraphProps g* **and** *gg'*:  $g \text{ [next-plane0}_p] \rightarrow g'$

**shows**  $P g g'$

*<proof>*

### 13.10.1 Increasing number of faces

**lemma** *splitFace-incr-faces*:

*pre-splitFace g u v f vs*  $\implies$

$|\text{finals}(\text{snd}(\text{snd}(\text{splitFace } g u v f \text{ vs})))| = |\text{finals } g| \wedge$

$|\text{nonFinals}(\text{snd}(\text{snd}(\text{splitFace } g u v f \text{ vs})))| = \text{Suc } |\text{nonFinals } g|$

*<proof>*

**lemma** *subdivFace'-incr-faces*:

*pre-subdivFace' g f u v n ovs*  $\implies$

*minGraphProps g*  $\implies f \in \mathcal{F} g \implies$

$|\text{finals}(\text{subdivFace' } g f u v n \text{ ovs})| = \text{Suc } |\text{finals } g| \wedge$

$|\text{nonFinals}(\text{subdivFace' } g f u v n \text{ ovs})| \geq |\text{nonFinals } g| - \text{Suc } 0$

*<proof>*

**lemma** *next-plane0-incr-faces*:

*minGraphProps g*  $\implies g \text{ [next-plane0}_p] \rightarrow g' \implies$

$|\text{finals } g'| = |\text{finals } g| + 1 \wedge |\text{nonFinals } g'| \geq |\text{nonFinals } g| - 1$

*<proof>*

**lemma** *two-faces-subdivFace'*:

*pre-subdivFace' g f u v n ovs*  $\implies \text{minGraphProps } g \implies f \in \mathcal{F} g \implies$

$|\text{faces } g| \geq 2 \implies |\text{faces}(\text{subdivFace' } g f u v n \text{ ovs})| \geq 2$

*<proof>*

### 13.11 Main invariant theorems

**lemma** *inv-genPoly*:

**assumes** *inv*: *inv g* **and** *polygen*:  $g' \in \text{set}(\text{generatePolygon } i v f g)$

**and** *f*:  $f \in \text{set}(\text{nonFinals } g)$  **and** *i*:  $2 < i$  **and** *v*:  $v \in \mathcal{V} f$

**shows** *inv g'*



*<proof>*

**lemma** *inv-inv-next-plane0*: *invariant inv next-plane0<sub>p</sub>*

*<proof>*

**end**

## 14 Further Plane Graph Properties

**theory** *PlaneProps*

**imports** *Invariants*

**begin**

### 14.1 *final*

**lemma** *plane-final-facesAt*:

**assumes** *inv g final g v :  $\mathcal{V} g f \in \text{set } (\text{facesAt } g v)$*  **shows** *final f*

*<proof>*

**lemma** *finalVertexI*:

$\llbracket \text{inv } g; \text{final } g; v \in \mathcal{V} g \rrbracket \implies \text{finalVertex } g v$

*<proof>*

**lemma** *setFinal-notin-finals*:

$\llbracket f \in \mathcal{F} g; \neg \text{final } f; \text{minGraphProps } g \rrbracket \implies \text{setFinal } f \notin \text{set } (\text{finals } g)$

*<proof>*

### 14.2 *degree*

**lemma** *planeN4*: *inv g  $\implies f \in \mathcal{F} g \implies 3 \leq |\text{vertices } f|$*

*<proof>*

**lemma** *degree-eq*:

**assumes** *pl: inv g and fin: final g and v: v :  $\mathcal{V} g$*

**shows** *degree g v = tri g v + quad g v + except g v*

*<proof>*

**lemma** *plane-fin-exceptionalVertex-def*:

**assumes** *pl: inv g and fin: final g and v: v :  $\mathcal{V} g$*

**shows** *exceptionalVertex g v =*

$( |\{ f \leftarrow \text{facesAt } g v . 5 \leq |\text{vertices } f| \} | \neq 0 )$

*<proof>*

**lemma** *not-exceptional*:

*inv g  $\implies$  final g  $\implies v : \mathcal{V} g \implies f \in \text{set } (\text{facesAt } g v) \implies$*

$\neg \text{exceptionalVertex } g \ v \implies |\text{vertices } f| \leq 4$   
 ⟨proof⟩

### 14.3 Misc

**lemma** *in-next-plane0I*:

**assumes**  $g' \in \text{set } (\text{generatePolygon } n \ v \ f \ g) \ f \in \text{set } (\text{nonFinals } g)$

$v \in \mathcal{V} \ f \ 3 \leq n < 4+p$

**shows**  $g' \in \text{set } (\text{next-plane0}_p \ g)$

⟨proof⟩

**lemma** *next-plane0-nonfinals*:  $g \ [\text{next-plane0}_p] \rightarrow g' \implies \text{nonFinals } g \neq []$

⟨proof⟩

**lemma** *next-plane0-ex*:

**assumes**  $a: g \ [\text{next-plane0}_p] \rightarrow g'$

**shows**  $\exists f \in \text{set}(\text{nonFinals } g). \exists v \in \mathcal{V} \ f. \exists i \in \text{set}([\mathcal{I}..<\text{Suc}(\text{maxGon } p)])$ .

$g' \in \text{set } (\text{generatePolygon } i \ v \ f \ g)$

⟨proof⟩

**lemma** *step-outside2*:

$\text{inv } g \implies g \ [\text{next-plane0}_p] \rightarrow g' \implies \neg \text{final } g' \implies |\text{faces } g'| \neq 2$

⟨proof⟩

### 14.4 Increasing final faces

**lemma** *set-finals-splitFace[simp]*:

$[[f \in \mathcal{F} \ g; \neg \text{final } f] \implies$

$\text{set}(\text{finals}(\text{snd}(\text{snd}(\text{splitFace } g \ u \ v \ f \ vs)))) = \text{set}(\text{finals } g)$

⟨proof⟩

**lemma** *next-plane0-finals-incr*:

$g \ [\text{next-plane0}_p] \rightarrow g' \implies f \in \text{set}(\text{finals } g) \implies f \in \text{set}(\text{finals } g')$

⟨proof⟩

**lemma** *next-plane0-finals-subset*:

$g' \in \text{set } (\text{next-plane0}_p \ g) \implies$

$\text{set } (\text{finals } g) \subseteq \text{set } (\text{finals } g')$

⟨proof⟩

**lemma** *next-plane0-final-mono*:

$[[g' \in \text{set } (\text{next-plane0}_p \ g); f \in \mathcal{F} \ g; \text{final } f] \implies f \in \mathcal{F} \ g'$

⟨proof⟩

## 14.5 Increasing vertices

**lemma** *next-plane0-vertices-subset*:

$\llbracket g' \in \text{set } (\text{next-plane0}_p g); \text{minGraphProps } g \rrbracket \implies \mathcal{V} g \subseteq \mathcal{V} g'$   
 $\langle \text{proof} \rangle$

## 14.6 Increasing vertex degrees

**lemma** *next-plane0-incr-faceListAt*:

$\llbracket g' \in \text{set } (\text{next-plane0}_p g); \text{minGraphProps } g \rrbracket$   
 $\implies |\text{faceListAt } g| \leq |\text{faceListAt } g'| \wedge$   
 $(\forall v < |\text{faceListAt } g|. |\text{faceListAt } g ! v| \leq |\text{faceListAt } g' ! v|)$   
 $(\text{is } - \implies - \implies ?Q g g')$   
 $\langle \text{proof} \rangle$

**lemma** *next-plane0-incr-degree*:

$\llbracket g' \in \text{set } (\text{next-plane0}_p g); \text{minGraphProps } g; v \in \mathcal{V} g \rrbracket$   
 $\implies \text{degree } g v \leq \text{degree } g' v$   
 $\langle \text{proof} \rangle$

## 14.7 Increasing *except*

**lemma** *next-plane0-incr-except*:

**assumes**  $g' \in \text{set } (\text{next-plane0}_p g) \text{ inv } g v \in \mathcal{V} g$   
**shows**  $\text{except } g v \leq \text{except } g' v$   
 $\langle \text{proof} \rangle$

## 14.8 Increasing edges

**lemma** *next-plane0-set-edges-subset*:

$\llbracket \text{minGraphProps } g; g [\text{next-plane0}_p] \rightarrow g' \rrbracket \implies \text{edges } g \subseteq \text{edges } g'$   
 $\langle \text{proof} \rangle$

## 14.9 Increasing final vertices

**declare** *atLeastLessThan-iff* [iff]

**lemma** *next-plane0-incr-finV*:

$\llbracket g' \in \text{set } (\text{next-plane0}_p g); \text{minGraphProps } g \rrbracket$   
 $\implies \forall v \in \mathcal{V} g. v \in \mathcal{V} g' \wedge$   
 $(\forall f \in \mathcal{F} g. v \in \mathcal{V} f \longrightarrow \text{final } f) \longrightarrow$   
 $(\forall f \in \mathcal{F} g'. v \in \mathcal{V} f \longrightarrow f \in \mathcal{F} g)$  (is -  $\implies$  -  $\implies$  ?Q g g')  
 $\langle \text{proof} \rangle$

**lemma** *next-plane0-finalVertex-mono*:

$\llbracket g' \in \text{set } (\text{next-plane0}_p g); \text{inv } g; u \in \mathcal{V} g; \text{finalVertex } g u \rrbracket$   
 $\implies \text{finalVertex } g' u$   
 $\langle \text{proof} \rangle$

## 14.10 Preservation of *facesAt* at final vertices

**lemma** *next-plane0-finalVertex-facesAt-eq*:

$\llbracket g' \in \text{set}(\text{next-plane0}_p g); \text{inv } g; v \in \mathcal{V} g; \text{finalVertex } g v \rrbracket$   
 $\implies \text{set}(\text{facesAt } g' v) = \text{set}(\text{facesAt } g v)$   
 ⟨proof⟩

**lemma** *next-plane0-len-filter-eq*:

**assumes**  $g' \in \text{set}(\text{next-plane0}_p g) \text{ inv } g v \in \mathcal{V} g \text{ finalVertex } g v$   
**shows**  $|\text{filter } P(\text{facesAt } g' v)| = |\text{filter } P(\text{facesAt } g v)|$   
 ⟨proof⟩

## 14.11 Properties of *subdivFace'*

**lemma** *new-edge-subdivFace'*:

$\bigwedge f v n g.$   
 $\text{pre-subdivFace}' g f u v n \text{ ovs} \implies \text{minGraphProps } g \implies f \in \mathcal{F} g \implies$   
 $\text{subdivFace}' g f v n \text{ ovs} = \text{makeFaceFinal } f g \vee$   
 $(\forall f' \in \mathcal{F}(\text{subdivFace}' g f v n \text{ ovs}) - (\mathcal{F} g - \{f\})).$   
 $\exists e \in \mathcal{E} f'. e \notin \mathcal{E} g)$   
 ⟨proof⟩

**lemma** *dist-edges-subdivFace'*:

$\text{pre-subdivFace}' g f u v n \text{ ovs} \implies \text{minGraphProps } g \implies f \in \mathcal{F} g \implies$   
 $\text{subdivFace}' g f v n \text{ ovs} = \text{makeFaceFinal } f g \vee$   
 $(\forall f' \in \mathcal{F}(\text{subdivFace}' g f v n \text{ ovs}) - (\mathcal{F} g - \{f\}). \mathcal{E} f' \neq \mathcal{E} f)$   
 ⟨proof⟩

**lemma** *between-last*:  $\llbracket \text{distinct}(\text{vertices } f); u \in \mathcal{V} f \rrbracket \implies$

$\text{between}(\text{vertices } f) u (\text{last}(\text{verticesFrom } f u)) =$   
 $\text{butlast}(\text{tl}(\text{verticesFrom } f u))$   
 ⟨proof⟩

**lemma** *final-subdivFace'*:  $\bigwedge f u n g. \text{minGraphProps } g \implies$

$\text{pre-subdivFace}' g f r u n \text{ ovs} \implies f \in \mathcal{F} g \implies$   
 $(\text{ovs} = [] \implies n=0 \wedge u = \text{last}(\text{verticesFrom } f r)) \implies$   
 $\exists f' \in \text{set}(\text{finals}(\text{subdivFace}' g f u n \text{ ovs})) - \text{set}(\text{finals } g).$   
 $(f^{-1} \cdot r, r) \in \mathcal{E} f' \wedge |\text{vertices } f'| =$   
 $n + |\text{ovs}| + (\text{if } r=u \text{ then } 1 \text{ else } |\text{between}(\text{vertices } f) r u| + 2)$   
 ⟨proof⟩

**lemma** *Seed-max-final-ex*:

$\exists f \in \text{set}(\text{finals}(\text{Seed } p)). |\text{vertices } f| = \text{maxGon } p$   
 ⟨proof⟩

**lemma** *max-face-ex*: **assumes**  $a: \text{Seed}_p [\text{next-plane}0_p] \rightarrow^* g$   
**shows**  $\exists f \in \text{set } (\text{finals } g). |\text{vertices } f| = \text{maxGon } p$   
 $\langle \text{proof} \rangle$

**end**

## 15 Summation Over Lists

**theory** *ListSum*  
**imports** *ListAux*  
**begin**

**primrec** *ListSum* ::  $'b \text{ list} \Rightarrow ('b \Rightarrow 'a::\text{comm-monoid-add}) \Rightarrow 'a::\text{comm-monoid-add}$   
**where**

$\text{ListSum } [] f = 0$   
 $|\text{ListSum } (l\#ls) f = f l + \text{ListSum } ls f$

**syntax**  $-\text{ListSum} :: \text{idt} \Rightarrow 'b \text{ list} \Rightarrow ('a::\text{comm-monoid-add}) \Rightarrow$   
 $('a::\text{comm-monoid-add}) \quad (\sum -\in- - [0, 0, 10] 10)$

**translations**  $\sum_{x \in xs} f == \text{CONST } \text{ListSum } xs (\lambda x. f)$

**lemma** [*simp*]:  $(\sum_{v \in V} 0) = (0::\text{nat}) \langle \text{proof} \rangle$

**lemma** *ListSum-compl1*:

$(\sum_{x \in [x \leftarrow xs. \neg P x]} f x) + (\sum_{x \in [x \leftarrow xs. P x]} f x) = (\sum_{x \in xs} (f x::\text{nat}))$   
 $\langle \text{proof} \rangle$

**lemma** *ListSum-compl2*:

$(\sum_{x \in [x \leftarrow xs. P x]} f x) + (\sum_{x \in [x \leftarrow xs. \neg P x]} f x) = (\sum_{x \in xs} (f x::\text{nat}))$   
 $\langle \text{proof} \rangle$

**lemmas** *ListSum-compl* = *ListSum-compl1 ListSum-compl2*

**lemma** *ListSum-conv-sum*:

$\text{distinct } xs \Longrightarrow \text{ListSum } xs f = \text{sum } f (\text{set } xs)$   
 $\langle \text{proof} \rangle$

**lemma** *listsum-cong*:

$\llbracket xs = ys; \bigwedge y. y \in \text{set } ys \Longrightarrow f y = g y \rrbracket$   
 $\Longrightarrow \text{ListSum } xs f = \text{ListSum } ys g$   
 $\langle \text{proof} \rangle$

**lemma** *strong-listsum-cong*[*cong*]:  

$$\llbracket xs = ys; \bigwedge y. y \in set\ ys = simp \Rightarrow f\ y = g\ y \rrbracket$$

$$\Rightarrow ListSum\ xs\ f = ListSum\ ys\ g$$

$$\langle proof \rangle$$

**lemma** *ListSum-eq* [*trans*]:  

$$(\bigwedge v. v \in set\ V \Rightarrow f\ v = g\ v) \Rightarrow (\sum v \in V\ f\ v) = (\sum v \in V\ g\ v)$$

$$\langle proof \rangle$$

**lemma** *ListSum-disj-union*:  

$$distinct\ A \Rightarrow distinct\ B \Rightarrow distinct\ C \Rightarrow$$

$$set\ C = set\ A \cup set\ B \Rightarrow$$

$$set\ A \cap set\ B = \{\} \Rightarrow$$

$$(\sum a \in C\ (f\ a)) = (\sum a \in A\ f\ a) + (\sum a \in B\ (f\ a::nat))$$

$$\langle proof \rangle$$

**lemma** *listsum-const*[*simp*]:  

$$(\sum x \in xs\ k) = length\ xs * k$$

$$\langle proof \rangle$$

**lemma** *ListSum-add*:  

$$(\sum x \in V\ f\ x) + (\sum x \in V\ g\ x) = (\sum x \in V\ (f\ x + (g\ x::nat)))$$

$$\langle proof \rangle$$

**lemma** *ListSum-le*:  

$$(\bigwedge v. v \in set\ V \Rightarrow f\ v \leq g\ v) \Rightarrow (\sum v \in V\ f\ v) \leq (\sum v \in V\ (g\ v::nat))$$

$$\langle proof \rangle$$

**lemma** *ListSum1-bound*:  

$$a \in set\ F \Rightarrow (d\ a::nat) \leq (\sum f \in F\ d\ f)$$

$$\langle proof \rangle$$

**end**

## 16 Tameness

**theory** *Tame*  
**imports** *Graph ListSum*  
**begin**

### 16.1 Constants

**definition** *squanderTarget* :: *nat* **where**  

$$squanderTarget \equiv 15410$$

**definition** *excessTCount* :: nat **where**

a  $\equiv$  6295

**definition** *squanderVertex* :: nat  $\Rightarrow$  nat  $\Rightarrow$  nat **where**

b  $p\ q \equiv$  if  $p = 0 \wedge q = 3$  then 6177  
 else if  $p = 0 \wedge q = 4$  then 9696  
 else if  $p = 1 \wedge q = 2$  then 6557  
 else if  $p = 1 \wedge q = 3$  then 6176  
 else if  $p = 2 \wedge q = 1$  then 7967  
 else if  $p = 2 \wedge q = 2$  then 4116  
 else if  $p = 2 \wedge q = 3$  then 12846  
 else if  $p = 3 \wedge q = 1$  then 3106  
 else if  $p = 3 \wedge q = 2$  then 8165  
 else if  $p = 4 \wedge q = 0$  then 3466  
 else if  $p = 4 \wedge q = 1$  then 3655  
 else if  $p = 5 \wedge q = 0$  then 395  
 else if  $p = 5 \wedge q = 1$  then 11354  
 else if  $p = 6 \wedge q = 0$  then 6854  
 else if  $p = 7 \wedge q = 0$  then 14493  
 else *squanderTarget*

**definition** *squanderFace* :: nat  $\Rightarrow$  nat **where**

d  $n \equiv$  if  $n = 3$  then 0  
 else if  $n = 4$  then 2058  
 else if  $n = 5$  then 4819  
 else if  $n = 6$  then 7120  
 else *squanderTarget*

## 16.2 Separated sets of vertices

A set of vertices  $V$  is *separated*, iff the following conditions hold:

2. No two vertices in  $V$  are adjacent:

**definition** *separated<sub>2</sub>* :: graph  $\Rightarrow$  vertex set  $\Rightarrow$  bool **where**  
*separated<sub>2</sub>*  $g\ V \equiv \forall v \in V. \forall f \in \text{set } (\text{facesAt } g\ v). f \cdot v \notin V$

3. No two vertices lie on a common quadrilateral:

**definition** *separated<sub>3</sub>* :: graph  $\Rightarrow$  vertex set  $\Rightarrow$  bool **where**  
*separated<sub>3</sub>*  $g\ V \equiv$   
 $\forall v \in V. \forall f \in \text{set } (\text{facesAt } g\ v). |\text{vertices } f| \leq 4 \longrightarrow \mathcal{V} f \cap V = \{v\}$

A set of vertices is called *separated*, iff no two vertices are adjacent or lie on a common quadrilateral:

**definition** *separated* :: graph  $\Rightarrow$  vertex set  $\Rightarrow$  bool **where**  
*separated*  $g\ V \equiv \text{separated}_2\ g\ V \wedge \text{separated}_3\ g\ V$

### 16.3 Admissible weight assignments

A weight assignment  $w :: \text{face} \Rightarrow \text{nat}$  assigns a natural number to every face.

We formalize the admissibility requirements as follows:

**definition**  $\text{admissible}_1 :: (\text{face} \Rightarrow \text{nat}) \Rightarrow \text{graph} \Rightarrow \text{bool}$  **where**  
 $\text{admissible}_1 w g \equiv \forall f \in \mathcal{F} g. \text{d } |\text{vertices } f| \leq w f$

**definition**  $\text{admissible}_2 :: (\text{face} \Rightarrow \text{nat}) \Rightarrow \text{graph} \Rightarrow \text{bool}$  **where**  
 $\text{admissible}_2 w g \equiv$   
 $\forall v \in \mathcal{V} g. \text{except } g v = 0 \longrightarrow \text{b } (\text{tri } g v) (\text{quad } g v) \leq (\sum_{f \in \text{facesAt } g v} w f)$

**definition**  $\text{admissible}_3 :: (\text{face} \Rightarrow \text{nat}) \Rightarrow \text{graph} \Rightarrow \text{bool}$  **where**  
 $\text{admissible}_3 w g \equiv$   
 $\forall v \in \mathcal{V} g. \text{vertextype } g v = (5,0,1) \longrightarrow (\sum_{f \in \text{filter triangle (facesAt } g v)} w(f)) \geq$   
a

Finally we define admissibility of weights functions.

**definition**  $\text{admissible} :: (\text{face} \Rightarrow \text{nat}) \Rightarrow \text{graph} \Rightarrow \text{bool}$  **where**  
 $\text{admissible } w g \equiv \text{admissible}_1 w g \wedge \text{admissible}_2 w g \wedge \text{admissible}_3 w g$

### 16.4 Tameness

**definition**  $\text{tame9a} :: \text{graph} \Rightarrow \text{bool}$  **where**  
 $\text{tame9a } g \equiv \forall f \in \mathcal{F} g. 3 \leq |\text{vertices } f| \wedge |\text{vertices } f| \leq 6$

**definition**  $\text{tame10} :: \text{graph} \Rightarrow \text{bool}$  **where**  
 $\text{tame10 } g = (\text{let } n = \text{countVertices } g \text{ in } 13 \leq n \wedge n \leq 15)$

**definition**  $\text{tame10ub} :: \text{graph} \Rightarrow \text{bool}$  **where**  
 $\text{tame10ub } g = (\text{countVertices } g \leq 15)$

**definition**  $\text{tame11a} :: \text{graph} \Rightarrow \text{bool}$  **where**  
 $\text{tame11a } g = (\forall v \in \mathcal{V} g. 3 \leq \text{degree } g v)$

**definition**  $\text{tame11b} :: \text{graph} \Rightarrow \text{bool}$  **where**  
 $\text{tame11b } g = (\forall v \in \mathcal{V} g. \text{degree } g v \leq (\text{if except } g v = 0 \text{ then } 7 \text{ else } 6))$

**definition**  $\text{tame12o} :: \text{graph} \Rightarrow \text{bool}$  **where**  
 $\text{tame12o } g =$   
 $(\forall v \in \mathcal{V} g. \text{except } g v \neq 0 \wedge \text{degree } g v = 6 \longrightarrow \text{vertextype } g v = (5,0,1))$

7. There exists an admissible weight assignment of total weight less than the target:

**definition**  $\text{tame13a} :: \text{graph} \Rightarrow \text{bool}$  **where**  
 $\text{tame13a } g = (\exists w. \text{admissible } w g \wedge (\sum_{f \in \text{faces } g} w f) < \text{squanderTarget})$

Finally we define the notion of tameness.



**definition** *tame* :: *graph*  $\Rightarrow$  *bool* **where**  
*tame* *g*  $\equiv$  *tame9a* *g*  $\wedge$  *tame10* *g*  $\wedge$  *tame11a* *g*  $\wedge$  *tame11b* *g*  $\wedge$  *tame12o* *g*  $\wedge$  *tame13a*  
*g*

**theory** *Plane1Props*  
**imports** *Plane1 PlaneProps Tame*  
**begin**

**lemma** *next-plane-subset*:  
 $\forall f \in \mathcal{F} \ g. \text{vertices } f \neq [] \implies$   
 $\text{set } (\text{next-plane}_p \ g) \subseteq \text{set } (\text{next-plane0}_p \ g)$   
 $\langle \text{proof} \rangle$

**lemma** *mgp-next-plane0-if-next-plane*:  
 $\text{minGraphProps } g \implies g [\text{next-plane}_p] \rightarrow g' \implies g [\text{next-plane0}_p] \rightarrow g'$   
 $\langle \text{proof} \rangle$

**lemma** *inv-inv-next-plane*: *invariant inv next-plane<sub>p</sub>*  
 $\langle \text{proof} \rangle$

**end**

## 17 Enumeration of Tame Plane Graphs

**theory** *Generator*  
**imports** *Plane1 Tame*  
**begin**

**definition** *faceSquanderLowerBound* :: *graph*  $\Rightarrow$  *nat* **where**  
*faceSquanderLowerBound* *g*  $\equiv \sum_{f \in \text{finals } g} d \ |\text{vertices } f|$

**definition** *d3-const* :: *nat* **where**  
*d3-const* == *d* 3

**definition** *d4-const* :: *nat* **where**  
*d4-const* == *d* 4

**definition** *excessAtType* :: *nat*  $\Rightarrow$  *nat*  $\Rightarrow$  *nat*  $\Rightarrow$  *nat* **where**  
*excessAtType* *t* *q* *e*  $\equiv$   
 $\text{if } e = 0 \text{ then if } 7 < t + q \text{ then squanderTarget}$   
 $\quad \text{else b } t \ q \ - \ t * \text{d3-const} \ - \ q * \text{d4-const}$   
 $\text{else if } t + q + e \neq 6 \text{ then } 0$   
 $\quad \text{else if } t=5 \text{ then a else squanderTarget}$

**declare** *d3-const-def*[simp] *d4-const-def*[simp]

**definition** *ExcessAt* :: *graph*  $\Rightarrow$  *vertex*  $\Rightarrow$  *nat* **where**  
*ExcessAt* *g v*  $\equiv$  *if*  $\neg$  *finalVertex* *g v* *then* 0  
*else* *excessAtType* (*tri* *g v*) (*quad* *g v*) (*except* *g v*)

**definition** *ExcessTable* :: *graph*  $\Rightarrow$  *vertex list*  $\Rightarrow$  (*vertex*  $\times$  *nat*) *list* **where**  
*ExcessTable* *g vs*  $\equiv$   
 $[(v, \text{ExcessAt } g \ v). \ v \leftarrow [v \leftarrow \text{vs}. \ 0 < \text{ExcessAt } g \ v]]$

Implementation:

**lemma** [code]:  
*ExcessTable* *g* =  
*List.map-filter* ( $\lambda v.$  *let* *e* = *ExcessAt* *g v* *in* *if*  $0 < e$  *then* *Some* (*v, e*) *else* *None*)  
 $\langle$ *proof* $\rangle$

**definition** *deleteAround* :: *graph*  $\Rightarrow$  *vertex*  $\Rightarrow$  (*vertex*  $\times$  *nat*) *list*  $\Rightarrow$  (*vertex*  $\times$  *nat*) *list* **where**  
*deleteAround* *g v ps*  $\equiv$   
*let* *fs* = *facesAt* *g v*;  
*ws* =  $\bigsqcup_{f \in fs}$  *if*  $|vertices \ f| = 4$  *then* [*f.v, f<sup>2</sup>.v*] *else* [*f.v*] *in*  
*removeKeyList* *ws ps*

Implementation:

**lemma** [code]: *deleteAround* *g v ps* =  
 $(let \ vs = (\lambda f. \ let \ n = f.v$   
 $\quad \quad \quad \text{in } if \ |vertices \ f| = 4 \ \text{then } [n, f.v] \ \text{else } [n])$   
 $\quad \text{in } removeKeyList \ (concat(map \ vs \ (facesAt \ g \ v))) \ ps)$   
 $\langle$ *proof* $\rangle$

**lemma** *length-deleteAround*:  $length \ (deleteAround \ g \ v \ ps) \leq length \ ps$   
 $\langle$ *proof* $\rangle$

**function** *ExcessNotAtRec* :: (*nat, nat*) *table*  $\Rightarrow$  *graph*  $\Rightarrow$  *nat* **where**  
*ExcessNotAtRec* [] = ( $\lambda g.$  0)  
 $| \ ExcessNotAtRec \ ((x, y)\#ps) = (\lambda g. \ max \ (ExcessNotAtRec \ ps \ g)$   
 $\quad \quad \quad (y + ExcessNotAtRec \ (deleteAround \ g \ x \ ps) \ g))$   
 $\langle$ *proof* $\rangle$

**termination**  $\langle$ *proof* $\rangle$

**definition** *ExcessNotAt* :: *graph*  $\Rightarrow$  *vertex option*  $\Rightarrow$  *nat* **where**  
*ExcessNotAt* *g v-opt*  $\equiv$   
*let* *ps* = *ExcessTable* *g* (*vertices* *g*) *in*  
*case* *v-opt* *of* *None*  $\Rightarrow$  *ExcessNotAtRec* *ps g*  
 $| \ Some \ v \Rightarrow ExcessNotAtRec \ (deleteAround \ g \ v \ ps) \ g$

**definition** *squanderLowerBound* :: *graph* ⇒ *nat* **where**  
*squanderLowerBound* *g* ≡ *faceSquanderLowerBound* *g* + *ExcessNotAt* *g* *None*

**definition** *is-tame13a* :: *graph* ⇒ *bool* **where**  
*is-tame13a* *g* ≡ *squanderLowerBound* *g* < *squanderTarget*

**definition** *notame* :: *graph* ⇒ *bool* **where**  
*notame* *g* ≡ ¬ (*tame10ub* *g* ∧ *tame11b* *g*)

**definition** *notame7* :: *graph* ⇒ *bool* **where**  
*notame7* *g* ≡ ¬ (*tame10ub* *g* ∧ *tame11b* *g* ∧ *is-tame13a* *g*)

**definition** *generatePolygonTame* :: *nat* ⇒ *vertex* ⇒ *face* ⇒ *graph* ⇒ *graph list*  
**where**  
*generatePolygonTame* *n v f g* ≡  
  *let*  
  *enumeration* = *enum* *n* |*vertices* *f*|;  
  *enumeration* = [*is* ← *enumeration*. ¬ *containsDuplicateEdge* *g f v is*];  
  *vertexLists* = [*indexToVertexList* *f v is*. *is* ← *enumeration*]  
  *in*  
  [*g'* ← [*subdivFace* *g f vs*. *vs* ← *vertexLists*] . ¬ *notame* *g'*]

**definition** *polysizes* :: *nat* ⇒ *graph* ⇒ *nat list* **where**  
*polysizes* *p g* ≡  
  *let* *lb* = *squanderLowerBound* *g* *in*  
  [*n* ← [*?* ..< *Suc*(*maxGon* *p*)]. *lb* + *d* *n* < *squanderTarget*]

**definition** *next-tame0* :: *nat* ⇒ *graph* ⇒ *graph list* (*next'-tame0*.) **where**  
*next-tame0* *p g* ≡  
  *let* *fs* = *nonFinals* *g* *in*  
  *if* *fs* = [] *then* []  
  *else* *let* *f* = *minimalFace* *fs*; *v* = *minimalVertex* *g f*  
  *in* [ *i* ∈ *polysizes* *p g* *generatePolygonTame* *i v f g* ]

Extensionally, *next-tame0* is just *filter* *P* ∘ *next-plane<sub>p</sub>* for some suitable *P*.  
But efficiency suffers considerably if we first create many graphs and then  
filter out the ones not in *polysizes*.

**end**

## 18 Tame Properties

**theory** *TameProps*  
**imports** *Tame RTranCl*  
**begin**

**lemma** *length-disj-filter-le*:  $\forall x \in \text{set } xs. \neg(P x \wedge Q x) \implies$   
 $\text{length}(\text{filter } P \text{ } xs) + \text{length}(\text{filter } Q \text{ } xs) \leq \text{length } xs$   
 <proof>

**lemma** *tri-quad-le-degree*:  $\text{tri } g \ v + \text{quad } g \ v \leq \text{degree } g \ v$   
 <proof>

**lemma** *faceCountMax-bound*:  
 $\llbracket \text{tame } g; v \in \mathcal{V} \ g \rrbracket \implies \text{tri } g \ v + \text{quad } g \ v \leq 7$   
 <proof>

**lemma** *filter-tame-succs*:  
**assumes** *invP*: *invariant P succs* **and** *fin*:  $\bigwedge g. \text{final } g \implies \text{succs } g = []$   
**and** *ok-untame*:  $\bigwedge g. P \ g \implies \neg \text{ok } g \implies \text{final } g \wedge \neg \text{tame } g$   
**and** *gg'*:  $g \ [\text{succs}] \rightarrow^* g'$   
**shows**  $P \ g \implies \text{final } g' \implies \text{tame } g' \implies g \ [\text{filter ok } \circ \text{succs}] \rightarrow^* g'$   
 <proof>

**definition** *untame* ::  $(\text{graph} \Rightarrow \text{bool}) \Rightarrow \text{bool}$  **where**  
 $\text{untame } P \equiv \forall g. \text{final } g \wedge P \ g \longrightarrow \neg \text{tame } g$

**lemma** *filterout-untame-succs*:  
**assumes** *invP*: *invariant P f* **and** *invPU*: *invariant*  $(\lambda g. P \ g \wedge U \ g) \ f$   
**and** *untame*:  $\text{untame}(\lambda g. P \ g \wedge U \ g)$   
**and** *new-untame*:  $\bigwedge g \ g'. \llbracket P \ g; g' \in \text{set}(f \ g); g' \notin \text{set}(f' \ g) \rrbracket \implies U \ g'$   
**and** *gg'*:  $g \ [f] \rightarrow^* g'$   
**shows**  $P \ g \implies \text{final } g' \implies \text{tame } g' \implies g \ [f'] \rightarrow^* g'$   
 <proof>

end

## 19 Neglectable Final Graphs

**theory** *TameEnum*  
**imports** *Generator*  
**begin**

**definition** *is-tame* ::  $\text{graph} \Rightarrow \text{bool}$  **where**  
 $\text{is-tame } g \equiv \text{tame10 } g \wedge \text{tame11a } g \wedge \text{tame12o } g \wedge \text{is-tame13a } g$

**definition** *next-tame* ::  $\text{nat} \Rightarrow \text{graph} \Rightarrow \text{graph list}$  (*next'-tame.*) **where**  
 $\text{next-tame}_p \equiv \text{filter } (\lambda g. \neg \text{final } g \vee \text{is-tame } g) \circ \text{next-tame0}_p$

**definition** *TameEnumP* ::  $\text{nat} \Rightarrow \text{graph set}$  (*TameEnum.*) **where**  
 $\text{TameEnum}_p \equiv \{g. \text{Seed}_p \ [\text{next-tame}_p] \rightarrow^* g \wedge \text{final } g\}$

**definition** *TameEnum* :: graph set **where**

*TameEnum*  $\equiv \bigcup_{p \leq 3}. \text{TameEnum}_p$

**end**

## 20 Properties of Lower Bound Machinery

**theory** *ScoreProps*

**imports** *ListSum TameEnum PlaneProps TameProps*

**begin**

**lemma** *deleteAround-empty[simp]*: *deleteAround* *g* *a* [] = []

*<proof>*

**lemma** *deleteAroundCons*:

*deleteAround* *g* *a* (*p*#*ps*) =  
  (*if* *fst* *p*  $\in \{v. \exists f \in \text{set } (\text{facesAt } g \ a)\}.$   
    (*length* (*vertices* *f*) = 4)  $\wedge v \in \{f \cdot a, f \cdot (f \cdot a)\}$   
     $\vee (\text{length } (\text{vertices } f) \neq 4) \wedge (v = f \cdot a)$ )  
  *then* *deleteAround* *g* *a* *ps*  
  *else* *p*#*deleteAround* *g* *a* *ps*)

*<proof>*

**lemma** *deleteAround-subset*: *set* (*deleteAround* *g* *a* *ps*)  $\subseteq$  *set* *ps*

*<proof>*

**lemma** *distinct-deleteAround*: *distinct* (*map* *fst* *ps*)  $\implies$

*distinct* (*map* *fst* (*deleteAround* *g* (*fst* (*a*, *b*)) *ps*))

*<proof>*

**definition** *deleteAround'* :: graph  $\Rightarrow$  vertex  $\Rightarrow$  (vertex  $\times$  nat) list  $\Rightarrow$

(vertex  $\times$  nat) list **where**

*deleteAround'* *g* *v* *ps*  $\equiv$   
  *let* *fs* = *facesAt* *g* *v*;  
  *vs* = ( $\lambda f. \text{let } n1 = f \cdot v;$   
     $n2 = f \cdot n1 \text{ in}$   
    *if* *length* (*vertices* *f*) = 4 *then* [*n1*, *n2*] *else* [*n1*]);  
  *ws* = *concat* (*map* *vs* *fs*) *in*  
  *removeKeyList* *ws* *ps*

**lemma** *deleteAround-eq*: *deleteAround* *g* *v* *ps* = *deleteAround'* *g* *v* *ps*

*<proof>*

**lemma** *deleteAround-nextVertex*:

*f*  $\in$  *set* (*facesAt* *g* *a*)  $\implies$

$(f \cdot a, b) \notin \text{set } (\text{deleteAround } g \ a \ ps)$   
 $\langle \text{proof} \rangle$

**lemma** *deleteAround-nextVertex-nextVertex*:  
 $f \in \text{set } (\text{facesAt } g \ a) \implies |\text{vertices } f| = 4 \implies$   
 $(f \cdot (f \cdot a), b) \notin \text{set } (\text{deleteAround } g \ a \ ps)$   
 $\langle \text{proof} \rangle$

**lemma** *deleteAround-prevVertex*:  
 $\text{minGraphProps } g \implies a : \mathcal{V} \ g \implies f \in \text{set } (\text{facesAt } g \ a) \implies$   
 $(f^{-1} \cdot a, b) \notin \text{set } (\text{deleteAround } g \ a \ ps)$   
 $\langle \text{proof} \rangle$

**lemma** *deleteAround-separated*:  
**assumes** *mgp*:  $\text{minGraphProps } g$  **and** *fin*:  $\text{final } g$  **and** *ag*:  $a : \mathcal{V} \ g$  **and**  $4 : |\text{vertices } f| \leq 4$   
**and** *f*:  $f \in \text{set } (\text{facesAt } g \ a)$   
**shows**  $\mathcal{V} \ f \cap \text{set } [\text{fst } p. \ p \leftarrow \text{deleteAround } g \ a \ ps] \subseteq \{a\}$  (**is** ?A)  
 $\langle \text{proof} \rangle$

**lemma** [*iff*]: *separated*  $g \ \{\}$   
 $\langle \text{proof} \rangle$

**lemma** *separated-insert*:  
**assumes** *mgp*:  $\text{minGraphProps } g$  **and** *a*:  $a \in \mathcal{V} \ g$   
**and** *Vg*:  $V \subseteq \mathcal{V} \ g$   
**and** *ps*: *separated*  $g \ V$   
**and** *s2*:  $(\bigwedge f. f \in \text{set } (\text{facesAt } g \ a) \implies f \cdot a \notin V)$   
**and** *s3*:  $(\bigwedge f. f \in \text{set } (\text{facesAt } g \ a) \implies$   
 $|\text{vertices } f| \leq 4 \implies \mathcal{V} \ f \cap V \subseteq \{a\})$   
**shows** *separated*  $g \ (\text{insert } a \ V)$   
 $\langle \text{proof} \rangle$

**function** *ExcessNotAtRecList* ::  $(\text{vertex}, \text{nat}) \ \text{table} \Rightarrow \text{graph} \Rightarrow \text{vertex list}$  **where**  
 $\text{ExcessNotAtRecList } [] = (\lambda g. [])$   
 $|\ \text{ExcessNotAtRecList } ((x, y) \# ps) = (\lambda g.$   
 $\ \ \ \ \ \text{let } l1 = \text{ExcessNotAtRecList } ps \ g;$   
 $\ \ \ \ \ l2 = \text{ExcessNotAtRecList } (\text{deleteAround } g \ x \ ps) \ g \ \text{in}$   
 $\ \ \ \ \ \text{if } \text{ExcessNotAtRec } ps \ g$   
 $\ \ \ \ \ \leq y + \text{ExcessNotAtRec } (\text{deleteAround } g \ x \ ps) \ g$   
 $\ \ \ \ \ \text{then } x \# l2 \ \text{else } l1)$   
 $\langle \text{proof} \rangle$

**termination**  $\langle \text{proof} \rangle$

**lemma** *isTable-deleteAround*:

$isTable\ E\ vs\ ((a,b)\#ps) \implies isTable\ E\ vs\ (deleteAround\ g\ a\ ps)$   
 <proof>

**lemma** *ListSum-ExcessNotAtRecList*:  
 $isTable\ E\ vs\ ps \implies ExcessNotAtRec\ ps\ g$   
 $= (\sum\ p \in\ ExcessNotAtRecList\ ps\ g\ E\ p)\ (is\ ?T\ ps \implies ?P\ ps)$   
 <proof>

**lemma** *ExcessNotAtRecList-subset*:  
 $set\ (ExcessNotAtRecList\ ps\ g) \subseteq set\ [fst\ p.\ p \leftarrow ps]\ (is\ ?P\ ps)$   
 <proof>

**lemma** *separated-ExcessNotAtRecList*:  
 $minGraphProps\ g \implies final\ g \implies isTable\ E\ (vertices\ g)\ ps \implies$   
 $separated\ g\ (set\ (ExcessNotAtRecList\ ps\ g))$   
 <proof>

**lemma** *isTable-ExcessTable*:  
 $isTable\ (\lambda v.\ ExcessAt\ g\ v)\ vs\ (ExcessTable\ g\ vs)$   
 <proof>

**lemma** *ExcessTable-subset*:  
 $set\ (map\ fst\ (ExcessTable\ g\ vs)) \subseteq set\ vs$   
 <proof>

**lemma** *distinct-ExcessNotAtRecList*:  
 $distinct\ (map\ fst\ ps) \implies distinct\ (ExcessNotAtRecList\ ps\ g)$   
 $(is\ ?T\ ps \implies ?P\ ps)$   
 <proof>

**primrec** *ExcessTable-cont* ::  
 $(vertex \Rightarrow nat) \Rightarrow vertex\ list \Rightarrow (vertex \times nat)\ list$

**where**  
 $ExcessTable-cont\ ExcessAtPG\ [] = []\ |$   
 $ExcessTable-cont\ ExcessAtPG\ (v\#\vs) =$   
 $(let\ vi = ExcessAtPG\ v\ in$   
 $\ if\ 0 < vi$   
 $\ then\ (v,\ vi)\#ExcessTable-cont\ ExcessAtPG\ vs$   
 $\ else\ ExcessTable-cont\ ExcessAtPG\ vs)$

**definition** *ExcessTable'* ::  $graph \Rightarrow vertex\ list \Rightarrow (vertex \times nat)\ list$  **where**  
 $ExcessTable'\ g \equiv ExcessTable-cont\ (ExcessAt\ g)$

**lemma** *distinct-ExcessTable-cont*:  
 $distinct\ vs \implies$   
 $distinct\ (map\ fst\ (ExcessTable-cont\ (ExcessAt\ g)\ vs))$

$\langle \text{proof} \rangle$

**lemma** *ExcessTable-cont-eq:*

*ExcessTable-cont*  $E$   $vs =$   
[[ $(v, E v). v \leftarrow [v \leftarrow vs . 0 < E v]$ ]]  
 $\langle \text{proof} \rangle$

**lemma** *ExcessTable-eq:*  $ExcessTable = ExcessTable'$

$\langle \text{proof} \rangle$

**lemma** *distinct-ExcessTable:*

$distinct\ vs \implies distinct\ [fst\ p. p \leftarrow ExcessTable\ g\ vs]$   
 $\langle \text{proof} \rangle$

**lemma** *ExcessNotAt-eq:*

$minGraphProps\ g \implies final\ g \implies$   
 $\exists V. ExcessNotAt\ g\ None$   
 $= (\sum v \in V\ ExcessAt\ g\ v)$   
 $\wedge separated\ g\ (set\ V) \wedge set\ V \subseteq set\ (vertices\ g)$   
 $\wedge distinct\ V$   
 $\langle \text{proof} \rangle$

**lemma** *excess-eq:*

**assumes**  $7: t + q \leq 7$   
**shows**  $excessAtType\ t\ q\ 0 + t * d\ 3 + q * d\ 4 = b\ t\ q$   
 $\langle \text{proof} \rangle$

**lemma** *excess-eq1:*

[[  $inv\ g; final\ g; tame\ g; except\ g\ v = 0; v \in set(vertices\ g)$  ]]  $\implies$   
 $ExcessAt\ g\ v + (tri\ g\ v) * d\ 3 + (quad\ g\ v) * d\ 4$   
 $= b\ (tri\ g\ v)\ (quad\ g\ v)$   
 $\langle \text{proof} \rangle$

separating

**definition** *separating* ::  $'a\ set \Rightarrow ('a \Rightarrow 'b\ set) \Rightarrow bool$  **where**

$separating\ V\ F \equiv$   
 $(\forall v1 \in V. \forall v2 \in V. v1 \neq v2 \longrightarrow F\ v1 \cap F\ v2 = \{\})$

**lemma** *separating-insert1:*

$separating\ (insert\ a\ V)\ F \implies separating\ V\ F$   
 $\langle \text{proof} \rangle$

**lemma** *separating-insert2:*

$separating\ (insert\ a\ V)\ F \implies a \notin V \implies v \in V \implies$   
 $F\ a \cap F\ v = \{\}$   
 $\langle \text{proof} \rangle$



**lemma** *sum-disj-Union*:

*finite V*  $\implies$   
 $(\bigwedge f. \text{finite } (F f)) \implies$   
*separating V F*  $\implies$   
 $(\sum v \in V. \sum f \in (F v). (w f :: \text{nat})) = (\sum f \in (\bigcup v \in V. F v). w f)$   
 $\langle \text{proof} \rangle$

**lemma** *separated-separating*:

**assumes** *Vg*: *set V*  $\subseteq$  *V g*  
**and** *pS*: *separated g (set V)*  
**and** *noex*:  $\forall f \in P. |\text{vertices } f| \leq 4$   
**shows** *separating (set V)*  $(\lambda v. \text{set } (\text{facesAt } g v) \cap P)$   
 $\langle \text{proof} \rangle$

**lemma** *ListSum-V-F-eq-ListSum-F*:

**assumes** *pl*: *inv g*  
**and** *pS*: *separated g (set V)* **and** *dist*: *distinct V*  
**and** *V-subset*: *set V*  $\subseteq$  *set (vertices g)*  
**and** *noex*:  $\forall f \in \text{Collect } P. |\text{vertices } f| \leq 4$   
**shows**  $(\sum v \in V \sum f \in \text{filter } P (\text{facesAt } g v) (w :: \text{face} \implies \text{nat}) f)$   
 $= (\sum f \in [f \leftarrow \text{faces } g . \exists v \in \text{set } V. f \in \text{set } (\text{facesAt } g v) \cap \text{Collect } P] w f)$   
 $\langle \text{proof} \rangle$

**lemma** *separated-disj-Union2*:

**assumes** *pl*: *inv g* **and** *fin*: *final g* **and** *ne*: *noExceptionals g (set V)*  
**and** *pS*: *separated g (set V)* **and** *dist*: *distinct V*  
**and** *V-subset*: *set V*  $\subseteq$  *set (vertices g)*  
**shows**  $(\sum v \in V \sum f \in \text{facesAt } g v (w :: \text{face} \implies \text{nat}) f)$   
 $= (\sum f \in [f \leftarrow \text{faces } g . \exists v \in \text{set } V. f \in \text{set } (\text{facesAt } g v)] w f)$   
 $\langle \text{proof} \rangle$

**lemma** *squanderFace-distr2*: *inv g*  $\implies$  *final g*  $\implies$  *noExceptionals g (set V)*  $\implies$   
*separated g (set V)*  $\implies$  *distinct V*  $\implies$  *set V*  $\subseteq$  *set (vertices g)*  $\implies$

$(\sum f \in [f \leftarrow \text{faces } g . \exists v \in \text{set } V. f \in \text{set } (\text{facesAt } g v)]$   
 $\quad \text{d } |\text{vertices } f|)$   
 $= (\sum v \in V ((\text{tri } g v) * \text{d } 3$   
 $\quad + (\text{quad } g v) * \text{d } 4))$   
 $\langle \text{proof} \rangle$

**lemma** *separated-subset*:

*V1*  $\subseteq$  *V2*  $\implies$  *separated g V2*  $\implies$  *separated g V1*  
 $\langle \text{proof} \rangle$

**end**

## 21 Correctness of Lower Bound for Final Graphs

```

theory LowerBound
imports PlaneProps ScoreProps
begin
<proof><proof><proof><proof><proof><proof>

```

```

theorem total-weight-lowerbound:
  inv g  $\implies$  final g  $\implies$  tame g  $\implies$  admissible w g  $\implies$ 
  ( $\sum f \in \text{faces } g \text{ } w f$ ) < squanderTarget  $\implies$ 
  squanderLowerBound g  $\leq$  ( $\sum f \in \text{faces } g \text{ } w f$ )
<proof>

```

## 22 Properties of Tame Graph Enumeration (1)

```

theory GeneratorProps
imports Plane1Props Generator TameProps LowerBound
begin

```

```

lemma genPolyTame-spec:
  generatePolygonTame n v f g = [g'  $\leftarrow$  generatePolygon n v f g .  $\neg$  notame g']
<proof>

```

```

lemma genPolyTame-subset-genPoly:
  g'  $\in$  set(generatePolygonTame i v f g)  $\implies$ 
  g'  $\in$  set(generatePolygon i v f g)
<proof>

```

```

lemma next-tame0-subset-plane:
  set(next-tame0 p g)  $\subseteq$  set(next-plane p g)
<proof>

```

```

lemma genPoly-new-face:
   $\llbracket g' \in \text{set}(\text{generatePolygon } n \text{ } v \text{ } f \text{ } g); \text{minGraphProps } g; f \in \text{set}(\text{nonFinals } g);$ 
   $v \in \mathcal{V} f; n \geq 3 \rrbracket \implies$ 
   $\exists f \in \text{set}(\text{finals } g') - \text{set}(\text{finals } g). |\text{vertices } f| = n$ 
<proof>

```

```

lemma genPoly-incr-facesquander-lb:
assumes g'  $\in$  set (generatePolygon n v f g) inv g
  f  $\in$  set(nonFinals g) v  $\in$   $\mathcal{V} f$  3  $\leq$  n
shows faceSquanderLowerBound g'  $\geq$  faceSquanderLowerBound g + d n
<proof>

```

**definition**  $close :: graph \Rightarrow vertex \Rightarrow vertex \Rightarrow bool$  **where**  
 $close\ g\ u\ v \equiv$   
 $\exists f \in set(facesAt\ g\ u). \text{ if } |vertices\ f| = 4 \text{ then } v = f \cdot u \vee v = f \cdot (f \cdot u)$   
 $\text{ else } v = f \cdot u$

**lemma**  $delAround-def$ :  $deleteAround\ g\ u\ ps = [p \leftarrow ps. \neg close\ g\ u\ (fst\ p)]$   
 $\langle proof \rangle$

**lemma**  $close-sym$ : **assumes**  $mgp: minGraphProps\ g$  **and**  $ug: u : \mathcal{V}\ g$  **and**  $cl: close\ g\ u\ v$   
 $g\ u\ v$   
**shows**  $close\ g\ v\ u$   
 $\langle proof \rangle$

**lemma**  $sep-conv$ :  
**assumes**  $mgp: minGraphProps\ g$  **and**  $V \subseteq \mathcal{V}\ g$   
**shows**  $separated\ g\ V = (\forall u \in V. \forall v \in V. u \neq v \longrightarrow \neg close\ g\ u\ v)$  **(is**  $?P = ?Q$ )  
 $\langle proof \rangle$

**lemma**  $sep-ne$ :  $\exists P \subseteq M. separated\ g\ (fst\ 'P)$   
 $\langle proof \rangle$

**lemma**  $ExcessNotAtRec-conv-Max$ :  
**assumes**  $mgp: minGraphProps\ g$   
**shows**  $set(map\ fst\ ps) \subseteq \mathcal{V}\ g \implies distinct(map\ fst\ ps) \implies$   
 $ExcessNotAtRec\ ps\ g =$   
 $Max\{ \sum p \in P. snd\ p \mid P. P \subseteq set\ ps \wedge separated\ g\ (fst\ 'P) \}$   
**(is**  $- \implies - \implies - = Max(?M\ ps)$  **is**  $- \implies - \implies - = Max\{- \mid P. ?S\ ps\ P\}$ )  
 $\langle proof \rangle$

**lemma**  $dist-ExcessTab$ :  $distinct\ (map\ fst\ (ExcessTable\ g\ (vertices\ g)))$   
 $\langle proof \rangle$

**lemma**  $mono-ExcessTab$ :  $\llbracket g' \in set\ (next-plane0_p\ g); inv\ g \rrbracket \implies$   
 $set(ExcessTable\ g\ (vertices\ g)) \subseteq set(ExcessTable\ g'\ (vertices\ g'))$   
 $\langle proof \rangle$

**lemma**  $close-antimono$ :  
 $\llbracket g' \in set\ (next-plane0_p\ g); inv\ g; u \in \mathcal{V}\ g; finalVertex\ g\ u \rrbracket \implies$   
 $close\ g'\ u\ v \implies close\ g\ u\ v$   
 $\langle proof \rangle$

**lemma** *ExcessTab-final*:

$p \in \text{set}(\text{ExcessTable } g \text{ (vertices } g)) \implies \text{finalVertex } g \text{ (fst } p)$   
<proof>

**lemma** *ExcessTab-vertex*:

$p \in \text{set}(\text{ExcessTable } g \text{ (vertices } g)) \implies \text{fst } p \in \mathcal{V} \ g$   
<proof>

**lemma** *fst-set-ExcessTable-subset*:

$\text{fst } ' \text{ set } (\text{ExcessTable } g \text{ (vertices } g)) \subseteq \mathcal{V} \ g$   
<proof>

**lemma** *next-plane0-incr-ExcessNotAt*:

$\llbracket g' \in \text{set}(\text{next-plane0}_p \ g); \text{inv } g \rrbracket \implies$   
 $\text{ExcessNotAt } g \ \text{None} \leq \text{ExcessNotAt } g' \ \text{None}$   
<proof>

**lemma** *next-plane0-incr-squander-lb*:

$\llbracket g' \in \text{set}(\text{next-plane0}_p \ g); \text{inv } g \rrbracket \implies$   
 $\text{squanderLowerBound } g \leq \text{squanderLowerBound } g'$   
<proof>

**lemma** *inv-notame*:

$\llbracket g' \in \text{set}(\text{next-plane0}_p \ g); \text{inv } g; \text{notame7 } g \rrbracket$   
 $\implies \text{notame7 } g'$   
<proof>

**lemma** *inv-inv-notame*:

$\text{invariant}(\lambda g. \text{inv } g \wedge \text{notame7 } g) \ \text{next-plane}_p$   
<proof>

**lemma** *untame-notame*:

$\text{untame } (\lambda g. \text{inv } g \wedge \text{notame7 } g)$   
<proof>

**lemma** *polysizes-tame*:

$\llbracket g' \in \text{set}(\text{generatePolygon } n \ v \ f \ g); \text{inv } g; f \in \text{set}(\text{nonFinals } g);$   
 $v \in \mathcal{V} \ f; 3 \leq n; n < 4+p; n \notin \text{set}(\text{polysizes } p \ g) \rrbracket$   
 $\implies \text{notame7 } g'$   
<proof>

**lemma** *genPolyTame-notame*:

$\llbracket g' \in \text{set}(\text{generatePolygon } n \ v \ f \ g); g' \notin \text{set}(\text{generatePolygonTame } n \ v \ f \ g);$

$inv\ g; 3 \leq n$  ]  
 $\implies notame\ \gamma\ g'$   
 <proof>

**declare** *upt-Suc*[*simp del*]

**lemma** *excess-notame*:

$\llbracket inv\ g; g' \in set\ (next-plane_p\ g); g' \notin set\ (next-tame0\ p\ g) \rrbracket$   
 $\implies notame\ \gamma\ g'$

<proof>

**declare** *upt-Suc*[*simp*]

**lemma** *next-tame0-comp*:  $\llbracket Seed_p\ [next-plane\ p] \rightarrow^* g; final\ g; tame\ g \rrbracket$   
 $\implies Seed_p\ [next-tame0\ p] \rightarrow^* g$

<proof>

**lemma** *inv-inv-next-tame0*: *invariant inv (next-tame0 p)*

<proof>

**lemma** *inv-inv-next-tame*: *invariant inv next-tame<sub>p</sub>*

<proof>

**lemma** *mgp-TameEnum*:  $g \in TameEnum_p \implies minGraphProps\ g$

<proof>

**end**

## 23 Properties of Tame Graph Enumeration (2)

**theory** *TameEnumProps*

**imports** *GeneratorProps*

**begin**

Completeness of filter for final graphs.

**lemma** *untame-negFin*:

**assumes** *pl*: *inv g* **and** *fin*: *final g* **and** *tame*: *tame g*

**shows** *is-tame g*

<proof>

**lemma** *next-tame-comp*:

$\llbracket tame\ g; final\ g; Seed_p\ [next-tame0\ p] \rightarrow^* g \rrbracket$   
 $\implies Seed_p\ [next-tame_p] \rightarrow^* g$

<proof>

**end**

**theory** *Worklist*  
**imports** *HOL-Library.While-Combinator RTranCl Quasi-Order*  
**begin**

**definition**

*worklist-aux* :: ('s ⇒ 'a ⇒ 'a list) ⇒ ('a ⇒ 's ⇒ 's)  
 ⇒ 'a list \* 's ⇒ ('a list \* 's)option

**where**

*worklist-aux succs f* =  
*while-option*  
 (λ(ws,s). ws ≠ [])  
 (λ(ws,s). case ws of x#ws' ⇒ (succs s x @ ws', f x s))

**definition** *worklist* :: ('s ⇒ 'a ⇒ 'a list) ⇒ ('a ⇒ 's ⇒ 's)  
 ⇒ 'a list ⇒ 's ⇒ 's option **where**

*worklist succs f ws s* =  
 (case *worklist-aux succs f* (ws,s) of  
 None ⇒ None | Some(ws,s) ⇒ Some s)

**lemma** *worklist-aux-Nil*: *worklist-aux succs f* ([],s) = Some([],s)  
 ⟨proof⟩

**lemma** *worklist-aux-Cons*:

*worklist-aux succs f* (x#ws',s) = *worklist-aux succs f* (succs s x @ ws', f x s)  
 ⟨proof⟩

**lemma** *worklist-aux-unfold*[code]:

*worklist-aux succs f* (ws,s) =  
 (case ws of [] ⇒ Some([],s)  
 | x#ws' ⇒ *worklist-aux succs f* (succs s x @ ws', f x s))  
 ⟨proof⟩

**definition**

*worklist-tree-aux* :: ('a ⇒ 'a list) ⇒ ('a ⇒ 's ⇒ 's)  
 ⇒ 'a list \* 's ⇒ ('a list \* 's)option

**where**

*worklist-tree-aux succs* = *worklist-aux* (λs. succs)

**lemma** *worklist-tree-aux-unfold*[code]:

*worklist-tree-aux succs f* (ws,s) =  
 (case ws of [] ⇒ Some([],s) |  
 x#ws' ⇒ *worklist-tree-aux succs f* (succs x @ ws', f x s))  
 ⟨proof⟩

**abbreviation** *Rel* :: ('a ⇒ 'a list) ⇒ ('a \* 'a)set **where**  
*Rel f* == {(x,y). y : set(f x)}

**lemma** *Image-Rel-set*:

$(Rel\ succs)^*$  “  $set(succs\ x) = (Rel\ succs)^+ \text{ “ } \{x\}$   
 ⟨proof⟩

**lemma** *RTranCl-conv*:

$g [succs] \rightarrow^* h \longleftrightarrow (g,h) : ((Rel\ succs)^*) \text{ (is } ?L = ?R)$   
 ⟨proof⟩

**lemma** *worklist-end-empty*:

$worklist-aux\ succs\ f\ (ws,s) = Some(ws',s') \implies ws' = []$   
 ⟨proof⟩

**theorem** *worklist-tree-aux-Some-foldl*:

**assumes**  $worklist-tree-aux\ succs\ f\ (ws,s) = Some(ws',s')$

**shows**  $\exists rs. set\ rs = ((Rel\ succs)^*) \text{ “ } (set\ ws) \wedge$

$$s' = foldl\ (\lambda s\ x. f\ x\ s)\ s\ rs$$

⟨proof⟩

**definition** *worklist-tree succs f ws s =*

(*case worklist-tree-aux succs f (ws,s) of*  
*None  $\Rightarrow$  None | Some(ws,s)  $\Rightarrow$  Some s*)

**theorem** *worklist-tree-Some-foldl*:

$worklist-tree\ succs\ f\ ws\ s = Some\ s' \implies$

$\exists rs. set\ rs = ((Rel\ succs)^*) \text{ “ } (set\ ws) \wedge$

$$s' = foldl\ (\lambda s\ x. f\ x\ s)\ s\ rs$$

⟨proof⟩

**lemma** *invariant-succs*:

**assumes** *invariant I succs*

**and**  $\forall x \in S. I\ x$

**shows**  $\forall x \in (Rel\ succs)^* \text{ “ } S. I\ x$

⟨proof⟩

**lemma** *worklist-tree-aux-rule*:

**assumes**  $worklist-tree-aux\ succs\ f\ (ws,s) = Some(ws',s')$

**and** *invariant I succs*

**and**  $\forall x \in set\ ws. I\ x$

**and**  $\bigwedge s. P\ []\ s\ s$

**and**  $\bigwedge r\ x\ ws\ s. I\ x \implies \forall x \in set\ ws. I\ x \implies P\ ws\ (f\ x\ s)\ r \implies P\ (x\#ws)\ s\ r$

**shows**  $\exists rs. set\ rs = ((Rel\ succs)^*) \text{ “ } (set\ ws) \wedge P\ rs\ s\ s'$

⟨proof⟩

**lemma** *worklist-tree-aux-rule2*:

**assumes**  $worklist-tree-aux\ succs\ f\ (ws,s) = Some(ws',s')$

**and** *invariant I succs*

**and**  $\forall x \in set\ ws. I\ x$

**and**  $S\ s$  **and**  $\bigwedge x\ s. I\ x \implies S\ s \implies S(f\ x\ s)$

**and**  $\bigwedge s. P\ []\ s\ s$

**and**  $\bigwedge r\ x\ ws\ s. I\ x \implies \forall x \in set\ ws. I\ x \implies S\ s$

$\implies P\ ws\ (f\ x\ s)\ r \implies P\ (x\#\!ws)\ s\ r$   
**shows**  $\exists rs. \text{set } rs = ((\text{Rel succs})^*) \text{ `` } (\text{set } ws) \wedge P\ rs\ s\ s'$   
 <proof>

**lemma** *worklist-tree-rule:*

**assumes** *worklist-tree succs*  $f\ ws\ s = \text{Some}(s')$   
**and** *invariant*  $I\ succs$   
**and**  $\forall x \in \text{set } ws. I\ x$   
**and**  $\bigwedge s. P\ []\ s\ s$   
**and**  $\bigwedge r\ x\ ws\ s. I\ x \implies \forall x \in \text{set } ws. I\ x \implies P\ ws\ (f\ x\ s)\ r \implies P\ (x\#\!ws)\ s\ r$   
**shows**  $\exists rs. \text{set } rs = ((\text{Rel succs})^*) \text{ `` } (\text{set } ws) \wedge P\ rs\ s\ s'$   
 <proof>

**lemma** *worklist-tree-rule2:*

**assumes** *worklist-tree succs*  $f\ ws\ s = \text{Some}(s')$   
**and** *invariant*  $I\ succs$   
**and**  $\forall x \in \text{set } ws. I\ x$   
**and**  $S\ s$  **and**  $\bigwedge x\ s. I\ x \implies S\ s \implies S(f\ x\ s)$   
**and**  $\bigwedge s. P\ []\ s\ s$   
**and**  $\bigwedge r\ x\ ws\ s. I\ x \implies \forall x \in \text{set } ws. I\ x \implies S\ s$   
 $\implies P\ ws\ (f\ x\ s)\ r \implies P\ (x\#\!ws)\ s\ r$   
**shows**  $\exists rs. \text{set } rs = ((\text{Rel succs})^*) \text{ `` } (\text{set } ws) \wedge P\ rs\ s\ s'$   
 <proof>

**lemma** *worklist-tree-aux-state-inv:*

**assumes** *worklist-tree-aux succs*  $f\ (ws, s) = \text{Some}(ws', s')$   
**and**  $I\ s$   
**and**  $\bigwedge x\ s. I\ s \implies I(f\ x\ s)$   
**shows**  $I\ s'$   
 <proof>

**lemma** *worklist-tree-state-inv:*

*worklist-tree succs*  $f\ ws\ s = \text{Some}(s')$   
 $\implies I\ s \implies (\bigwedge x\ s. I\ s \implies I(f\ x\ s)) \implies I\ s'$   
 <proof>

**locale** *set-modulo = quasi-order +*

**fixes** *empty* :: 's

**and** *insert-mod* :: 'a  $\Rightarrow$  's  $\Rightarrow$  's

**and** *set-of* :: 's  $\Rightarrow$  'a *set*

**and**  $I$  :: 'a  $\Rightarrow$  *bool*

**and**  $S$  :: 's  $\Rightarrow$  *bool*

**assumes** *set-of-empty*: *set-of empty* = {}

**and** *set-of-insert-mod*:  $I\ x \implies S\ s \wedge (\forall x \in \text{set-of } s. I\ x)$

$\implies$

*set-of(insert-mod x s) = insert x (set-of s)  $\vee$*

*( $\exists y \in \text{set-of } s. x \preceq y$ )  $\wedge$  set-of (insert-mod x s) = set-of s*

**and** *S-empty*:  $S\ \text{empty}$



**and** *S-insert-mod*:  $S\ s \implies S\ (\text{insert-mod}\ x\ s)$   
**begin**

**definition** *insert-mod2* ::  $('b \Rightarrow \text{bool}) \Rightarrow ('b \Rightarrow 'a) \Rightarrow 'b \Rightarrow 's \Rightarrow 's$  **where**  
*insert-mod2*  $P\ f\ x\ s = (\text{if } P\ x\ \text{then } \text{insert-mod}\ (f\ x)\ s\ \text{else } s)$

**definition** *SI*  $s = (S\ s \wedge (\forall x \in \text{set-of } s. I\ x))$

**lemma** *SI-empty*: *SI empty*  
 $\langle \text{proof} \rangle$

**lemma** *SI-insert-mod*:  
 $I\ x \implies SI\ s \implies SI\ (\text{insert-mod}\ x\ s)$   
 $\langle \text{proof} \rangle$

**lemma** *SI-insert-mod2*:  $(\bigwedge x. \text{inv0}\ x \implies I\ (f\ x)) \implies$   
 $\text{inv0}\ x \implies SI\ s \implies SI\ (\text{insert-mod2}\ P\ f\ x\ s)$   
 $\langle \text{proof} \rangle$

**definition** *worklist-tree-coll-aux* ::  
 $('b \Rightarrow 'b\ \text{list}) \Rightarrow ('b \Rightarrow \text{bool}) \Rightarrow ('b \Rightarrow 'a) \Rightarrow 'b\ \text{list} \Rightarrow 's \Rightarrow 's\ \text{option}$   
**where**  
*worklist-tree-coll-aux succs*  $P\ f = \text{worklist-tree succs } (\text{insert-mod2}\ P\ f)$

**definition** *worklist-tree-coll* ::  
 $('b \Rightarrow 'b\ \text{list}) \Rightarrow ('b \Rightarrow \text{bool}) \Rightarrow ('b \Rightarrow 'a) \Rightarrow 'b\ \text{list} \Rightarrow 's\ \text{option}$   
**where**  
*worklist-tree-coll succs*  $P\ f\ ws = \text{worklist-tree-coll-aux succs } P\ f\ ws\ \text{empty}$

**lemma** *worklist-tree-coll-aux-equiv*:  
**assumes** *worklist-tree-coll-aux succs*  $P\ f\ ws\ s = \text{Some } s'$   
**and** *invariant inv0 succs*  
**and**  $\forall x \in \text{set } ws. \text{inv0}\ x$   
**and**  $\bigwedge x. \text{inv0}\ x \implies I(f\ x)$   
**and** *SI*  $s$   
**shows**  $\text{set-of } s' =_{\leq} f\ ' \{x : (\text{Rel succs})^* \text{ `` } (\text{set } ws). P\ x\} \cup \text{set-of } s$   
 $\langle \text{proof} \rangle$

**lemma** *worklist-tree-coll-equiv*:  
*worklist-tree-coll succs*  $P\ f\ ws = \text{Some } s' \implies \text{invariant inv0 succs}$   
 $\implies \forall x \in \text{set } ws. \text{inv0}\ x \implies (\bigwedge x. \text{inv0}\ x \implies I(f\ x))$   
 $\implies \text{set-of } s' =_{\leq} f\ ' \{x : (\text{Rel succs})^* \text{ `` } (\text{set } ws). P\ x\}$   
 $\langle \text{proof} \rangle$

**lemma** *worklist-tree-coll-aux-subseteq*:  
*worklist-tree-coll-aux succs*  $P\ f\ ws\ t_0 = \text{Some } t \implies$   
*invariant inv0 succs*  $\implies \forall g \in \text{set } ws. \text{inv0}\ g \implies$   
 $(\bigwedge x. \text{inv0}\ x \implies I(f\ x)) \implies SI\ t_0 \implies$

*set-of*  $t \subseteq \text{set-of } t_0 \cup f \text{ ' } \{h : (\text{Rel succs})^* \text{ " set ws. } P h\}$   
 <proof>

**lemma** *worklist-tree-coll-subseteq*:

*worklist-tree-coll succs*  $P f ws = \text{Some } t \implies$   
*invariant inv0 succs*  $\implies \forall g \in \text{set ws. inv0 } g \implies$   
 $(\bigwedge x. \text{inv0 } x \implies I(f x)) \implies$   
*set-of*  $t \subseteq f \text{ ' } \{h : (\text{Rel succs})^* \text{ " set ws. } P h\}$   
 <proof>

**lemma** *worklist-tree-coll-inv*:

*worklist-tree-coll succs*  $P f ws = \text{Some } s \implies S s$   
 <proof>

**end**

**end**

**theory** *Maps*

**imports** *Worklist Quasi-Order*

**begin**

**locale** *maps* =

**fixes** *empty* :: 'm

**and** *up* :: 'a  $\Rightarrow$  'b list  $\Rightarrow$  'm  $\Rightarrow$  'm

**and** *map-of* :: 'm  $\Rightarrow$  'a  $\Rightarrow$  'b list

**and** *M* :: 'm  $\Rightarrow$  bool

**assumes** *map-empty*: *map-of empty* =  $(\lambda a. [])$

**and** *map-up*: *map-of* (*up* a b m) = (*map-of* m)(a := b)

**and** *M-empty*: *M empty*

**and** *M-up*: *M* m  $\implies$  *M* (*up* a b m)

**begin**

**definition** *set-of* m =  $(UN x. \text{set}(\text{map-of } m x))$

**end**

**locale** *set-mod-maps* = *maps empty up map-of M + quasi-order qle*

**for** *empty* :: 'm

**and** *up* :: 'a  $\Rightarrow$  'b list  $\Rightarrow$  'm  $\Rightarrow$  'm

**and** *map-of* :: 'm  $\Rightarrow$  'a  $\Rightarrow$  'b list

**and** *M* :: 'm  $\Rightarrow$  bool

**and** *qle* :: 'b  $\Rightarrow$  'b  $\Rightarrow$  bool (**infix**  $\preceq$  60)

+

**fixes** *subsumed* :: 'b  $\Rightarrow$  'b  $\Rightarrow$  bool

**and** *I* :: 'b  $\Rightarrow$  bool

**and** *key* :: 'b  $\Rightarrow$  'a

**assumes** *equiv-iff-qle*:  $I x \implies I y \implies \text{subsumed } x y = (x \preceq y)$

**and** *key=key*

**begin**

**definition** *insert-mod*  $x\ m =$   
 (let  $k = \text{key } x$ ;  $ys = \text{map-of } m\ k$   
 in if  $(\exists y \in \text{set } ys. \text{subsumed } x\ y)$  then  $m$  else  $\text{up } k\ (x\#\text{ys})\ m$ )

**end**

**sublocale**

*set-mod-maps* <  
*set-by-maps?*: *set-modulo* *gle empty insert-mod set-of*  $I\ M$   
 <*proof*>

**end**

## 24 Archive

**theory** *Arch*

**imports** *Main HOL-Library.Code-Target-Numeral*

**begin**

<*ML*>

The definition of these constants is only ever needed at the ML level when running the eval proof method.

**definition** *Tri* :: *nat list list list*

**where**

$\text{Tri} = (\text{map} \circ \text{map} \circ \text{map})\ \text{nat-of-integer}\ \text{Tri}'$

**definition** *Quad* :: *nat list list list*

**where**

$\text{Quad} = (\text{map} \circ \text{map} \circ \text{map})\ \text{nat-of-integer}\ \text{Quad}'$

**definition** *Pent* :: *nat list list list*

**where**

$\text{Pent} = (\text{map} \circ \text{map} \circ \text{map})\ \text{nat-of-integer}\ \text{Pent}'$

**definition** *Hex* :: *nat list list list*

**where**

$\text{Hex} = (\text{map} \circ \text{map} \circ \text{map})\ \text{nat-of-integer}\ \text{Hex}'$

**end**

## 25 Comparing Enumeration and Archive

**theory** *ArchCompAux*

**imports** *TameEnum Trie.Tries Maps Arch Worklist*

**begin**

**function** *qsort* :: ('a ⇒ 'a ⇒ bool) ⇒ 'a list ⇒ 'a list **where**  
*qsort* le [] = [] |  
*qsort* le (x#xs) = *qsort* le [y←xs . ¬ le x y] @ [x] @  
                  *qsort* le [y←xs . le x y]  
⟨*proof*⟩  
**termination** ⟨*proof*⟩

**definition** *nof-vertices* :: 'a fgraph ⇒ nat **where**  
*nof-vertices* = length ∘ remdups ∘ concat

**definition** *fgraph* :: graph ⇒ nat fgraph **where**  
*fgraph* g = map vertices (faces g)

**definition** *hash* :: nat fgraph ⇒ nat list **where**  
*hash* fs = (let n = *nof-vertices* fs in  
          [n, size fs] @  
          *qsort* (λx y. y < x) (map (λi. foldl (+) 0 (map size [f←fs. i ∈ set f]))  
                                  [0..<n]))

**definition** *samet* :: (nat, nat fgraph) tries option ⇒ nat fgraph list ⇒ bool  
**where**  
*samet* fgto ags = (case fgto of None ⇒ False | Some tfgs ⇒  
          let tags = tries-of-list hash ags in  
          (all-tries (λfg. list-ex (iso-test fg) (lookup-tries tags (hash fg))) tfgs ∧  
          all-tries (λag. list-ex (iso-test ag) (lookup-tries tfgs (hash ag))) tags))

**definition** *pre-iso-test* :: vertex fgraph ⇒ bool **where**  
*pre-iso-test* Fs ↔  
          [] ∉ set Fs ∧ (∀ F ∈ set Fs. distinct F) ∧ distinct (map rotate-min Fs)

**interpretation** *map*:  
          maps Trie None [] update-trie lookup-tries invar-trie  
⟨*proof*⟩

**lemma** *set-of-conv*: set-tries = maps.set-of lookup-tries  
⟨*proof*⟩

**end**

## 26 Completeness of Archive Test

**theory** *ArchCompProps*  
**imports** *TameEnumProps ArchCompAux*  
**begin**  
**lemma** *mgp-pre-iso-test*: minGraphProps g ⇒ pre-iso-test(*fgraph* g)

*<proof>*

**corollary** *iso-test-correct*:

$\llbracket \text{pre-iso-test } Fs_1; \text{pre-iso-test } Fs_2 \rrbracket \implies$   
 $\text{iso-test } Fs_1 \text{ } Fs_2 = (Fs_1 \simeq Fs_2)$   
*<proof>*

**lemma** *trie-all-eq-set-of-trie*:

$\text{invar-trie } t \implies \text{all-trie } (\text{list-all } P) \text{ } t = (\forall v \in \text{set-tries } t. P \ v)$   
*<proof>*

**lemma** *samet-imp-iso-seteq*:

**assumes** *pre1*:  $\bigwedge gs \ g. \text{gsopt} = \text{Some } gs \implies g \in \text{set-tries } gs \implies \text{pre-iso-test } g$   
**and** *pre2*:  $\bigwedge g. g \in \text{set arch} \implies \text{pre-iso-test } g$   
**and** *inv*:  $\bigwedge gs. \text{gsopt} = \text{Some } gs \implies \text{invar-trie } gs$   
**and** *same*: *samet gsopt arch*  
**shows**  $\exists gs. \text{gsopt} = \text{Some } gs \wedge \text{set-tries } gs \simeq \text{set arch}$   
*<proof>*

**lemma** *samet-imp-iso-subseteq*:

**assumes** *pre1*:  $\bigwedge gs \ g. \text{gsopt} = \text{Some } gs \implies g \in \text{set-tries } gs \implies \text{pre-iso-test } g$   
**and** *pre2*:  $\bigwedge g. g \in \text{set arch} \implies \text{pre-iso-test } g$   
**and** *inv*:  $\bigwedge gs. \text{gsopt} = \text{Some } gs \implies \text{invar-trie } gs$   
**and** *same*: *samet gsopt arch*  
**shows**  $\exists gs. \text{gsopt} = \text{Some } gs \wedge \text{set-tries } gs \subseteq \simeq \text{set arch}$   
*<proof>*

**global-interpretation** *set-mod-trie*:

*set-mod-maps* *Trie None*  $\llbracket$  *update-trie lookup-tries invar-trie*  $(\simeq)$  *iso-test pre-iso-test hash*

**defines** *insert-mod-trie* = *set-mod-maps.insert-mod update-trie lookup-tries iso-test hash*

**and** *worklist-tree-coll-trie* = *set-modulo.worklist-tree-coll* (*Trie None*  $\llbracket$ ) *insert-mod-trie*

**and** *worklist-tree-coll-aux-trie* = *set-modulo.worklist-tree-coll-aux insert-mod-trie*

**and** *insert-mod2-trie* = *set-modulo.insert-mod2 insert-mod-trie*

*<proof>*

**definition** *enum-filter-finals* ::

$(\text{graph} \Rightarrow \text{graph list}) \Rightarrow \text{graph list}$

$\Rightarrow (\text{nat}, \text{nat } f\text{graph}) \text{ tries option}$  **where**

*enum-filter-finals succs* = *set-mod-trie.worklist-tree-coll succs final fgraph*

**definition** *tameEnumFilter* ::  $\text{nat} \Rightarrow (\text{nat}, \text{nat } f\text{graph}) \text{ tries option}$  **where**

*tameEnumFilter p* = *enum-filter-finals* (*next-tame p*) [*Seed p*]

**lemma** *TameEnum-tameEnumFilter*:

$\text{tameEnumFilter } p = \text{Some } t \implies \text{set-tries } t \simeq f\text{graph } \text{' } \text{TameEnum}_p$   
*<proof>*

**lemma** *tameEnumFilter-subseteq-TameEnum*:  
 $tameEnumFilter\ p = Some\ t \implies set\ tries\ t \subseteq fgraph\ 'TameEnum_p$   
 <proof>

**lemma** *inv-tries-tameEnumFilter*:  
 $tameEnumFilter\ p = Some\ t \implies invar\ trie\ t$   
 <proof>

**theorem** *combine-evals-filter*:  
 $\forall g \in set\ arch.\ pre\ iso\ test\ g \implies samet\ (tameEnumFilter\ p)\ arch$   
 $\implies fgraph\ 'TameEnum_p \subseteq_{\approx} set\ arch$   
 <proof>

**end**

## 27 Completeness Proofs under hypothetical computations

**theory** *Relative-Completeness*  
**imports** *ArchCompProps*  
**begin**

**definition** *Archive* :: *vertex fgraph set where*  
 $Archive \equiv set(Tri\ @\ Quad\ @\ Pent\ @\ Hex)$

**locale** *archive-by-computation* =  
**assumes** *pre-iso-test3*:  $\forall g \in set\ Tri.\ pre\ iso\ test\ g$   
**assumes** *pre-iso-test4*:  $\forall g \in set\ Quad.\ pre\ iso\ test\ g$   
**assumes** *pre-iso-test5*:  $\forall g \in set\ Pent.\ pre\ iso\ test\ g$   
**assumes** *pre-iso-test6*:  $\forall g \in set\ Hex.\ pre\ iso\ test\ g$   
**assumes** *same3*:  $samet\ (tameEnumFilter\ 0)\ Tri$   
**assumes** *same4*:  $samet\ (tameEnumFilter\ 1)\ Quad$   
**assumes** *same5*:  $samet\ (tameEnumFilter\ 2)\ Pent$   
**assumes** *same6*:  $samet\ (tameEnumFilter\ 3)\ Hex$   
**begin**

**theorem** *TameEnum-Archive*:  $fgraph\ 'TameEnum \subseteq_{\approx} Archive$   
 <proof>

**lemma** *TameEnum-comp*:  
**assumes**  $Seed_p\ [next\ plane_p] \rightarrow^* g$  **and** *final g* **and** *tame g*  
**shows**  $Seed_p\ [next\ tame_p] \rightarrow^* g$   
 <proof>

**lemma** *tame5*:

**assumes**  $g: \text{Seed}_p [\text{next-plane}0_p] \rightarrow^* g$  **and** *final*  $g$  **and** *tame*  $g$   
**shows**  $p \leq 3$   
*<proof>*

**theorem** *completeness*:  
**assumes**  $g \in \text{PlaneGraphs}$  **and** *tame*  $g$  **shows** *fgraph*  $g \in_{\sim} \text{Archive}$   
*<proof>*

**end**

**end**

## References

- [1] Tobias Nipkow, Gertrud Bauer, and Paula Schultz. Flyspeck I: Tame graphs. In U. Furbach and N. Shankar, editors, *Automated Reasoning (IJCAR 2006)*, volume 4130 of *LNCS*, pages 21–35. Springer, 2006.