

# The Classical Seifert–van Kampen Theorem

Arthur F. Ramos      David Barros Hulak  
Ruy J. G. B. de Queiroz

June 25, 2026

## Abstract

This entry formalizes a quotient-oriented proof of the classical Seifert–van Kampen theorem in Isabelle/HOL. For open subsets  $U$  and  $V$  of a topological space with  $x_0 \in U \cap V$  and path-connected intersection, it proves that the fundamental group of  $U \cup V$  at  $x_0$  is in bijection with the carrier-based amalgamated free product of the fundamental groups of  $U$  and  $V$  over the fundamental group of  $U \cap V$ . The development also provides reusable infrastructure for explicit path homotopies, pushout-style quotients, free-product words, carrier-based amalgamated free products, and the abstract encode/decode interface used to organize the final argument. AI assistance was used for proof engineering. The final definitions, statements, and proofs are checked by Isabelle.

## 1 Overview

The main unconditional result of the entry is the theorem `classical_seifert_van_kampen_bij_betw` in `Classical_Seifert_Van_Kampen.thy`. It establishes the classical open-union form of Seifert–van Kampen through a quotient-level bijection between the fundamental group of  $U \cup V$  and the carrier-based amalgamated free product assembled from the fundamental groups of  $U$ ,  $V$ , and  $U \cap V$ .

Supporting theories package the quotient-level encode/decode interface and the topological pushout constructions used by the classical proof. These auxiliary layers are internal to the entry; the completed headline result remains the classical open-union theorem above.

## 2 Development structure

The theories are organized in three layers:

- *Quotients and pushouts*: `Equivalence_Quotients.thy`, `Pushout_Scaffold.thy`, `Binary_Sum_Topology.thy`, and `Topological_`

Pushout\_Scaffold.thy isolate the generic equivalence-class and topological-pushout constructions.

- *The algebraic target:* Free\_Product\_Words.thy, Amalgamated\_Free\_Product.thy, Carrier\_Group\_Scaffold.thy, Carrier\_Amalgamated\_Free\_Product\_Enum.thy, and Carrier\_Amalgamated\_Free\_Product\_Eval.thy provide the word calculus, carrier-group locales, and evaluation map for the carrier-based amalgamated free product.
- *The topological argument:* Explicit\_Path\_Homotopy\_Scaffold.thy, Explicit\_Fundamental\_Group\_Scaffold.thy, and Fundamental\_Group\_Scaffold.thy develop the quotient view of the fundamental group, while Seifert\_Van\_Kampen\_Scaffold.thy, Topological\_Seifert\_Van\_Kampen.thy, and Classical\_Seifert\_Van\_Kampen.thy package the abstract interface and its classical open-union instantiation.

### 3 Sources

The informal mathematics follows the classical theorem due to Seifert and van Kampen [3, 4] and standard textbook presentations in algebraic topology, especially Hatcher [2] and Brown [1].

## Contents

<b>1 Overview</b>	<b>1</b>
<b>2 Development structure</b>	<b>1</b>
<b>3 Sources</b>	<b>2</b>
<b>4 Equivalence classes as quotient infrastructure</b>	<b>3</b>
<b>5 Free-product words</b>	<b>4</b>
<b>6 Amalgamation quotients of free-product words</b>	<b>7</b>
<b>7 Carrier-based amalgamated free products</b>	<b>9</b>
<b>8 Groups on carriers</b>	<b>14</b>
<b>9 Evaluating carrier amalgamated free-product words</b>	<b>15</b>
<b>10 Explicit-topology paths and homotopies</b>	<b>18</b>
<b>11 Fundamental-group quotients</b>	<b>24</b>

<b>12</b>	<b>Explicit-topology fundamental-group quotients</b>	<b>31</b>
<b>13</b>	<b>Pushout-style quotients of disjoint sums</b>	<b>36</b>
<b>14</b>	<b>Conditional Seifert–van Kampen interface</b>	<b>38</b>
<b>15</b>	<b>Classical Seifert–van Kampen for open unions</b>	<b>40</b>
15.1	Carrier-side setup . . . . .	41
15.2	Partitions and encoded loop words . . . . .	45
15.3	Homotopy invariance of the partition encoding . . . . .	68
15.4	Encoding, decoding, and the final bijection . . . . .	68
<b>16</b>	<b>Binary coproduct topology</b>	<b>70</b>
<b>17</b>	<b>Topological pushouts</b>	<b>72</b>
<b>18</b>	<b>Concrete decode data for topological pushouts</b>	<b>74</b>
<b>19</b>	<b>Top-level entry point</b>	<b>78</b>
	<b>theory</b> <i>Equivalence-Quotients</i>	
	<b>imports</b> <i>Main</i>	
	<b>begin</b>	

## 4 Equivalence classes as quotient infrastructure

The later pushout and fundamental-group constructions are phrased in terms of explicit equivalence classes and quotient carriers. This small theory keeps that quotient infrastructure elementary and reusable, so subsequent theories can concentrate on the specific relations that matter for Seifert–van Kampen.

**definition** *equiv-class* :: ('a => 'a => bool) => 'a => 'a set **where**  
*equiv-class* R x = {y. R y x}

**definition** *quotient-space* :: ('a => 'a => bool) => 'a set set **where**  
*quotient-space* R = *equiv-class* R ‘ UNIV

**lemma** *quotient-spaceI*:  
*equiv-class* R x ∈ *quotient-space* R  
 ⟨*proof*⟩

**lemma** *equiv-class-iff* [*simp*]:  
 y ∈ *equiv-class* R x ↔ R y x  
 ⟨*proof*⟩

**locale** *equivalence-relation* =  
**fixes** R :: 'a => 'a => bool  
**assumes** *refl* [*intro!*, *simp*]: R x x

```

    and sym: R x y ==> R y x
    and transitive: R x y ==> R y z ==> R x z
begin

lemma equiv-class-eqI:
  assumes R x y
  shows equiv-class R x = equiv-class R y
  <proof>

lemma equiv-class-eq-iff:
  equiv-class R x = equiv-class R y <math>\longleftrightarrow</math> R x y
  <proof>

lemma quotient-spaceE:
  assumes Q ∈ quotient-space R
  obtains x where Q = equiv-class R x
  <proof>

end

end

theory Free-Product-Words
  imports Main
begin

```

## 5 Free-product words

Encodings of loops in the Seifert–van Kampen argument are expressed as words that alternate between generators from the left and right factors. This theory provides the basic word combinators, reversal operations, and reduction-oriented bookkeeping used by the later amalgamation quotients.

```

datatype ('a, 'b) free-product-word =
  WordNil
  | WordLeft 'a ('a, 'b) free-product-word
  | WordRight 'b ('a, 'b) free-product-word

primrec fpw-concat ::
  ('a, 'b) free-product-word => ('a, 'b) free-product-word => ('a, 'b)
  free-product-word
where
  fpw-concat WordNil w2 = w2
| fpw-concat (WordLeft a rest) w2 = WordLeft a (fpw-concat rest w2)
| fpw-concat (WordRight b rest) w2 = WordRight b (fpw-concat rest w2)

primrec fpw-length :: ('a, 'b) free-product-word => nat where
  fpw-length WordNil = 0
| fpw-length (WordLeft a rest) = Suc (fpw-length rest)
| fpw-length (WordRight b rest) = Suc (fpw-length rest)

```

**primrec** *fpw-reverse* :: ('a, 'b) free-product-word => ('a, 'b) free-product-word  
**where**

*fpw-reverse* WordNil = WordNil  
| *fpw-reverse* (WordLeft a rest) = *fpw-concat* (*fpw-reverse* rest) (WordLeft a WordNil)  
| *fpw-reverse* (WordRight b rest) = *fpw-concat* (*fpw-reverse* rest) (WordRight b WordNil)

**primrec** *fpw-map* ::

('a => 'c) => ('b => 'd) => ('a, 'b) free-product-word => ('c, 'd) free-product-word

**where**

*fpw-map* fl fr WordNil = WordNil  
| *fpw-map* fl fr (WordLeft a rest) = WordLeft (fl a) (*fpw-map* fl fr rest)  
| *fpw-map* fl fr (WordRight b rest) = WordRight (fr b) (*fpw-map* fl fr rest)

**lemma** *fpw-concat-nil-right* [simp]:

*fpw-concat* w WordNil = w  
<proof>

**lemma** *fpw-concat-assoc* [simp]:

*fpw-concat* (*fpw-concat* u v) w = *fpw-concat* u (*fpw-concat* v w)  
<proof>

**lemma** *fpw-reverse-concat* [simp]:

*fpw-reverse* (*fpw-concat* u v) = *fpw-concat* (*fpw-reverse* v) (*fpw-reverse* u)  
<proof>

**lemma** *fpw-reverse-reverse* [simp]:

*fpw-reverse* (*fpw-reverse* w) = w  
<proof>

**fun** *fpw-reduced* :: ('a, 'b) free-product-word => bool **where**

*fpw-reduced* WordNil = True  
| *fpw-reduced* (WordLeft a WordNil) = True  
| *fpw-reduced* (WordLeft a (WordLeft b rest)) = False  
| *fpw-reduced* (WordLeft a (WordRight b rest)) = *fpw-reduced* (WordRight b rest)  
| *fpw-reduced* (WordRight b WordNil) = True  
| *fpw-reduced* (WordRight b (WordRight c rest)) = False  
| *fpw-reduced* (WordRight b (WordLeft a rest)) = *fpw-reduced* (WordLeft a rest)

**fun** *fpw-inverse* ::

('a::group-add, 'b::group-add) free-product-word =>  
( 'a, 'b) free-product-word

**where**

*fpw-inverse* WordNil = WordNil  
| *fpw-inverse* (WordLeft a rest) =  
*fpw-concat* (*fpw-inverse* rest) (WordLeft (- a) WordNil)

| *fpw-inverse* (*WordRight* *b* *rest*) =  
*fpw-concat* (*fpw-inverse* *rest*) (*WordRight* (- *b*) *WordNil*)

**lemma** *fpw-inverse-concat*:

*fpw-inverse* (*fpw-concat* *u* *v*) = *fpw-concat* (*fpw-inverse* *v*) (*fpw-inverse* *u*)  
⟨*proof*⟩

**lemma** *fpw-inverse-inverse* [*simp*]:

*fpw-inverse* (*fpw-inverse* *w*) = *w*  
⟨*proof*⟩

**inductive** *fpw-reduction-step* ::

('a::group-add, 'b::group-add) *free-product-word* =>  
('a, 'b) *free-product-word* => *bool*

**where**

*combine-left*:

*fpw-reduction-step*  
(*WordLeft* *a* (*WordLeft* *b* *rest*))  
(*WordLeft* (*a* + *b*) *rest*)

| *combine-right*:

*fpw-reduction-step*  
(*WordRight* *a* (*WordRight* *b* *rest*))  
(*WordRight* (*a* + *b*) *rest*)

| *remove-left-zero*:

*fpw-reduction-step* (*WordLeft* 0 *rest*) *rest*

| *remove-right-zero*:

*fpw-reduction-step* (*WordRight* 0 *rest*) *rest*

| *context-left*:

*fpw-reduction-step* *u* *v* ==>  
*fpw-reduction-step* (*WordLeft* *a* *u*) (*WordLeft* *a* *v*)

| *context-right*:

*fpw-reduction-step* *u* *v* ==>  
*fpw-reduction-step* (*WordRight* *b* *u*) (*WordRight* *b* *v*)

**inductive** *fpw-reduction* ::

('a::group-add, 'b::group-add) *free-product-word* =>  
('a, 'b) *free-product-word* => *bool*

**where**

*refl* [*intro!*, *simp*]: *fpw-reduction* *w* *w*

| *sym*: *fpw-reduction* *u* *v* ==> *fpw-reduction* *v* *u*

| *trans*: *fpw-reduction* *u* *v* ==> *fpw-reduction* *v* *w* ==> *fpw-reduction* *u* *w*

| *step*: *fpw-reduction-step* *u* *v* ==> *fpw-reduction* *u* *v*

**lemma** *fpw-reduction-left-context*:

**assumes** *fpw-reduction* *u* *v*

**shows** *fpw-reduction* (*WordLeft* *a* *u*) (*WordLeft* *a* *v*)

⟨*proof*⟩

**lemma** *fpw-reduction-right-context*:

```

assumes fpw-reduction u v
shows fpw-reduction (WordRight b u) (WordRight b v)
  ⟨proof⟩

end
theory Amalgamated-Free-Product
  imports Equivalence-Quotients Free-Product-Words
begin

```

## 6 Amalgamation quotients of free-product words

This is the purely algebraic target of the later topological theorem. Starting from free-product words, the theory quotients by the relations induced by the common interface and packages the resulting equivalence classes as the abstract amalgamated free product.

```

inductive amalgam-step ::
  ('h => 'a) => ('h => 'b) =>
    ('a, 'b) free-product-word => ('a, 'b) free-product-word => bool
  for i1 :: 'h => 'a and i2 :: 'h => 'b
where
  identify:
    amalgam-step i1 i2 (WordLeft (i1 h) rest) (WordRight (i2 h) rest)

```

```

inductive amalgam-equiv ::
  ('h => 'a) => ('h => 'b) =>
    ('a, 'b) free-product-word => ('a, 'b) free-product-word => bool
  for i1 :: 'h => 'a and i2 :: 'h => 'b
where
  refl [intro!, simp]: amalgam-equiv i1 i2 w w
| sym: amalgam-equiv i1 i2 u v ==> amalgam-equiv i1 i2 v u
| trans: amalgam-equiv i1 i2 u v ==> amalgam-equiv i1 i2 v w ==> amalgam-equiv
i1 i2 u w
| step: amalgam-step i1 i2 u v ==> amalgam-equiv i1 i2 u v
| left-context:
  amalgam-equiv i1 i2 u v ==> amalgam-equiv i1 i2 (WordLeft a u) (WordLeft
a v)
| right-context:
  amalgam-equiv i1 i2 u v ==> amalgam-equiv i1 i2 (WordRight b u) (WordRight
b v)

```

```

interpretation amalgam-equiv-equiv: equivalence-relation amalgam-equiv i1 i2
  ⟨proof⟩

```

```

definition amalgam-class ::
  ('h => 'a) => ('h => 'b) =>
    ('a, 'b) free-product-word => (('a, 'b) free-product-word) set
where
  amalgam-class i1 i2 w = equiv-class (amalgam-equiv i1 i2) w

```

**definition** *amalgamated-free-product-space* ::

$(h \Rightarrow a) \Rightarrow (h \Rightarrow b) \Rightarrow$   
 $((a, b) \text{ free-product-word}) \text{ set set}$

**where**

$\text{amalgamated-free-product-space } i1 \ i2 = \text{quotient-space } (\text{amalgam-equiv } i1 \ i2)$

**lemma** *amalgam-class-eq-iff*:

$\text{amalgam-class } i1 \ i2 \ u = \text{amalgam-class } i1 \ i2 \ v \iff \text{amalgam-equiv } i1 \ i2 \ u \ v$   
{proof}

**inductive** *full-amalg-equiv* ::

$(h \Rightarrow a::\text{group-add}) \Rightarrow (h \Rightarrow b::\text{group-add}) \Rightarrow$   
 $((a, b) \text{ free-product-word}) \Rightarrow ((a, b) \text{ free-product-word}) \Rightarrow \text{bool}$   
**for**  $i1 :: h \Rightarrow a$  **and**  $i2 :: h \Rightarrow b$

**where**

*refl* [*intro!*, *simp*]:  $\text{full-amalg-equiv } i1 \ i2 \ w \ w$

| *sym*:  $\text{full-amalg-equiv } i1 \ i2 \ u \ v \implies \text{full-amalg-equiv } i1 \ i2 \ v \ u$

| *trans*:

$\text{full-amalg-equiv } i1 \ i2 \ u \ v \implies \text{full-amalg-equiv } i1 \ i2 \ v \ w \implies \text{full-amalg-equiv } i1 \ i2 \ u \ w$

| *of-amalg*:

$\text{amalgam-equiv } i1 \ i2 \ u \ v \implies \text{full-amalg-equiv } i1 \ i2 \ u \ v$

| *of-reduction*:

$\text{fpw-reduction } u \ v \implies \text{full-amalg-equiv } i1 \ i2 \ u \ v$

**interpretation** *full-amalg-equiv-equiv*: *equivalence-relation full-amalg-equiv i1 i2*

{proof}

**definition** *full-amalg-class* ::

$(h \Rightarrow a::\text{group-add}) \Rightarrow (h \Rightarrow b::\text{group-add}) \Rightarrow$   
 $((a, b) \text{ free-product-word}) \Rightarrow ((a, b) \text{ free-product-word}) \text{ set}$

**where**

$\text{full-amalg-class } i1 \ i2 \ w = \text{equiv-class } (\text{full-amalg-equiv } i1 \ i2) \ w$

**definition** *full-amalgamated-free-product-space* ::

$(h \Rightarrow a::\text{group-add}) \Rightarrow (h \Rightarrow b::\text{group-add}) \Rightarrow$   
 $((a, b) \text{ free-product-word}) \text{ set set}$

**where**

$\text{full-amalgamated-free-product-space } i1 \ i2 =$   
 $\text{quotient-space } (\text{full-amalg-equiv } i1 \ i2)$

**lemma** *full-amalg-class-eq-iff*:

$\text{full-amalg-class } i1 \ i2 \ u = \text{full-amalg-class } i1 \ i2 \ v \iff \text{full-amalg-equiv } i1 \ i2 \ u \ v$   
{proof}

**lemma** *full-amalg-class-in-space* [*intro*]:

$\text{full-amalg-class } i1 \ i2 \ w \in \text{full-amalgamated-free-product-space } i1 \ i2$   
{proof}

**definition** *full-amalg-one* ::  
 ('h => 'a::group-add) => ('h => 'b::group-add) =>  
 (('a, 'b) free-product-word) set  
**where**  
*full-amalg-one* i1 i2 = *full-amalg-class* i1 i2 WordNil

**end**  
**theory** *Carrier-Amalgamated-Free-Product*  
**imports** *Amalgamated-Free-Product*  
**begin**

## 7 Carrier-based amalgamated free products

The abstract amalgamated free product is refined here to words whose letters lie in fixed carrier sets. That carrier-level control matches the way loop representatives are produced on the topological side and is what eventually lets the Seifert–van Kampen theorem talk about concrete fundamental-group carriers.

**fun** *fpw-in-space* :: 'a set => 'b set => ('a, 'b) free-product-word => bool **where**  
*fpw-in-space* G H WordNil = True  
 | *fpw-in-space* G H (WordLeft a rest) = (a ∈ G ∧ *fpw-in-space* G H rest)  
 | *fpw-in-space* G H (WordRight b rest) = (b ∈ H ∧ *fpw-in-space* G H rest)

**definition** *carrier-fpw-space* :: 'a set => 'b set => (('a, 'b) free-product-word) set  
**where**  
*carrier-fpw-space* G H = {w. *fpw-in-space* G H w}

**lemma** *carrier-fpw-space-iff* [*simp*]:  
 w ∈ *carrier-fpw-space* G H ↔ *fpw-in-space* G H w  
 ⟨*proof*⟩

**inductive** *carrier-amalgam-step* ::  
 'h set => ('h => 'a) => ('h => 'b) =>  
 ('a, 'b) free-product-word => ('a, 'b) free-product-word => bool  
**for** K :: 'h set **and** i1 :: 'h => 'a **and** i2 :: 'h => 'b  
**where**  
*identify*:  
 h ∈ K ⇒  
*carrier-amalgam-step* K i1 i2  
 (WordLeft (i1 h) rest)  
 (WordRight (i2 h) rest)

**inductive** *carrier-amalgam-equiv* ::  
 'h set => ('h => 'a) => ('h => 'b) =>  
 ('a, 'b) free-product-word => ('a, 'b) free-product-word => bool  
**for** K :: 'h set **and** i1 :: 'h => 'a **and** i2 :: 'h => 'b  
**where**

*refl* [*intro!*, *simp*]: *carrier-amalgam-equiv*  $K$   $i1$   $i2$   $w$   $w$   
| *sym*:  
  *carrier-amalgam-equiv*  $K$   $i1$   $i2$   $u$   $v$   $\implies$   
  *carrier-amalgam-equiv*  $K$   $i1$   $i2$   $v$   $u$   
| *trans*:  
  *carrier-amalgam-equiv*  $K$   $i1$   $i2$   $u$   $v$   $\implies$   
  *carrier-amalgam-equiv*  $K$   $i1$   $i2$   $v$   $w$   $\implies$   
  *carrier-amalgam-equiv*  $K$   $i1$   $i2$   $u$   $w$   
| *step*:  
  *carrier-amalgam-step*  $K$   $i1$   $i2$   $u$   $v$   $\implies$   
  *carrier-amalgam-equiv*  $K$   $i1$   $i2$   $u$   $v$   
| *left-context*:  
  *carrier-amalgam-equiv*  $K$   $i1$   $i2$   $u$   $v$   $\implies$   
  *carrier-amalgam-equiv*  $K$   $i1$   $i2$  (*WordLeft*  $a$   $u$ ) (*WordLeft*  $a$   $v$ )  
| *right-context*:  
  *carrier-amalgam-equiv*  $K$   $i1$   $i2$   $u$   $v$   $\implies$   
  *carrier-amalgam-equiv*  $K$   $i1$   $i2$  (*WordRight*  $b$   $u$ ) (*WordRight*  $b$   $v$ )

**interpretation** *carrier-amalgam-equiv-equiv*:  
  *equivalence-relation carrier-amalgam-equiv*  $K$   $i1$   $i2$   
  ⟨*proof*⟩

**definition** *carrier-amalgam-class* ::  
  ' $h$  *set*  $\implies$  (' $h$   $\implies$  ' $a$ )  $\implies$  (' $h$   $\implies$  ' $b$ )  $\implies$   
  ('' $a$ , ' $b$ ) *free-product-word*  $\implies$  (('' $a$ , ' $b$ ) *free-product-word*) *set*  
**where**  
  *carrier-amalgam-class*  $K$   $i1$   $i2$   $w$  =  
  *equiv-class* (*carrier-amalgam-equiv*  $K$   $i1$   $i2$ )  $w$

**lemma** *carrier-amalgam-class-eq-iff*:  
  *carrier-amalgam-class*  $K$   $i1$   $i2$   $u$  = *carrier-amalgam-class*  $K$   $i1$   $i2$   $v$   
   $\iff$  *carrier-amalgam-equiv*  $K$   $i1$   $i2$   $u$   $v$   
  ⟨*proof*⟩

**inductive** *carrier-fpw-reduction-step* ::  
  ' $a$  *set*  $\implies$  ' $b$  *set*  $\implies$   
  ('' $a$   $\implies$  ' $a$   $\implies$  ' $a$ )  $\implies$  ' $a$   $\implies$  ('' $b$   $\implies$  ' $b$   $\implies$  ' $b$ )  $\implies$  ' $b$   $\implies$   
  ('' $a$ , ' $b$ ) *free-product-word*  $\implies$  (('' $a$ , ' $b$ ) *free-product-word*)  $\implies$  *bool*  
**for**  $G1$  :: ' $a$  *set* **and**  $G2$  :: ' $b$  *set*  
  **and** *mult1* :: ' $a$   $\implies$  ' $a$   $\implies$  ' $a$  **and** *one1* :: ' $a$   
  **and** *mult2* :: ' $b$   $\implies$  ' $b$   $\implies$  ' $b$  **and** *one2* :: ' $b$   
**where**  
  *combine-left*:  
  [[ $a \in G1$ ;  $b \in G1$ ; *mult1*  $a$   $b \in G1$ ; *fpw-in-space*  $G1$   $G2$  *rest*]]  $\implies$   
  *carrier-fpw-reduction-step*  $G1$   $G2$  *mult1* *one1* *mult2* *one2*  
  (*WordLeft*  $a$  (*WordLeft*  $b$  *rest*))  
  (*WordLeft* (*mult1*  $a$   $b$ ) *rest*)  
| *combine-right*:  
  [[ $a \in G2$ ;  $b \in G2$ ; *mult2*  $a$   $b \in G2$ ; *fpw-in-space*  $G1$   $G2$  *rest*]]  $\implies$

```

    carrier-fpw-reduction-step G1 G2 mult1 one1 mult2 one2
    (WordRight a (WordRight b rest))
    (WordRight (mult2 a b) rest)
| remove-left-one:
  [[one1 ∈ G1; fpw-in-space G1 G2 rest]] ⇒
  carrier-fpw-reduction-step G1 G2 mult1 one1 mult2 one2
  (WordLeft one1 rest) rest
| remove-right-one:
  [[one2 ∈ G2; fpw-in-space G1 G2 rest]] ⇒
  carrier-fpw-reduction-step G1 G2 mult1 one1 mult2 one2
  (WordRight one2 rest) rest
| context-left:
  [[a ∈ G1; carrier-fpw-reduction-step G1 G2 mult1 one1 mult2 one2 u v]] ⇒
  carrier-fpw-reduction-step G1 G2 mult1 one1 mult2 one2
  (WordLeft a u) (WordLeft a v)
| context-right:
  [[b ∈ G2; carrier-fpw-reduction-step G1 G2 mult1 one1 mult2 one2 u v]] ⇒
  carrier-fpw-reduction-step G1 G2 mult1 one1 mult2 one2
  (WordRight b u) (WordRight b v)

```

**inductive** *carrier-fpw-reduction* ::

```

'a set => 'b set =>
('a => 'a => 'a) => 'a => ('b => 'b => 'b) => 'b =>
('a, 'b) free-product-word => ('a, 'b) free-product-word => bool
for G1 :: 'a set and G2 :: 'b set
and mult1 :: 'a => 'a => 'a and one1 :: 'a
and mult2 :: 'b => 'b => 'b and one2 :: 'b

```

**where**

```

refl [intro!, simp]: carrier-fpw-reduction G1 G2 mult1 one1 mult2 one2 w w
| sym:
  carrier-fpw-reduction G1 G2 mult1 one1 mult2 one2 u v ⇒
  carrier-fpw-reduction G1 G2 mult1 one1 mult2 one2 v u
| trans:
  carrier-fpw-reduction G1 G2 mult1 one1 mult2 one2 u v ⇒
  carrier-fpw-reduction G1 G2 mult1 one1 mult2 one2 v w ⇒
  carrier-fpw-reduction G1 G2 mult1 one1 mult2 one2 u w
| step:
  carrier-fpw-reduction-step G1 G2 mult1 one1 mult2 one2 u v ⇒
  carrier-fpw-reduction G1 G2 mult1 one1 mult2 one2 u v

```

**lemma** *carrier-fpw-reduction-left-context*:

```

assumes carrier-fpw-reduction G1 G2 mult1 one1 mult2 one2 u v
and a ∈ G1
shows carrier-fpw-reduction G1 G2 mult1 one1 mult2 one2 (WordLeft a u)
(WordLeft a v)
⟨proof⟩

```

**lemma** *carrier-fpw-reduction-right-context*:

```

assumes carrier-fpw-reduction G1 G2 mult1 one1 mult2 one2 u v

```

**and**  $b \in G2$   
**shows** *carrier-fpw-reduction*  $G1\ G2\ mult1\ one1\ mult2\ one2$  (*WordRight*  $b\ u$ )  
(*WordRight*  $b\ v$ )  
*<proof>*

**lemma** *carrier-fpw-reduction-step-preserves-space*:  
**assumes** *step: carrier-fpw-reduction-step*  $G1\ G2\ mult1\ one1\ mult2\ one2\ u\ v$   
**shows** *fpw-in-space*  $G1\ G2\ u$  **and** *fpw-in-space*  $G1\ G2\ v$   
*<proof>*

**lemma** *carrier-fpw-reduction-step-preserves-space-iff*:  
**assumes** *step: carrier-fpw-reduction-step*  $G1\ G2\ mult1\ one1\ mult2\ one2\ u\ v$   
**shows** *fpw-in-space*  $G1\ G2\ u \longleftrightarrow$  *fpw-in-space*  $G1\ G2\ v$   
*<proof>*

**lemma** *carrier-fpw-reduction-preserves-space-iff*:  
**assumes** *rel: carrier-fpw-reduction*  $G1\ G2\ mult1\ one1\ mult2\ one2\ u\ v$   
**shows** *fpw-in-space*  $G1\ G2\ u \longleftrightarrow$  *fpw-in-space*  $G1\ G2\ v$   
*<proof>*

**inductive** *carrier-full-amalg-equiv* ::  
 $'a\ set \Rightarrow 'b\ set \Rightarrow 'h\ set \Rightarrow ('h \Rightarrow 'a) \Rightarrow ('h \Rightarrow 'b) \Rightarrow$   
 $('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow 'a \Rightarrow ('b \Rightarrow 'b \Rightarrow 'b) \Rightarrow 'b \Rightarrow$   
 $('a, 'b)\ free-product-word \Rightarrow ('a, 'b)\ free-product-word \Rightarrow bool$   
**for**  $G1 :: 'a\ set$  **and**  $G2 :: 'b\ set$   
**and**  $K :: 'h\ set$  **and**  $i1 :: 'h \Rightarrow 'a$  **and**  $i2 :: 'h \Rightarrow 'b$   
**and**  $mult1 :: 'a \Rightarrow 'a \Rightarrow 'a$  **and**  $one1 :: 'a$   
**and**  $mult2 :: 'b \Rightarrow 'b \Rightarrow 'b$  **and**  $one2 :: 'b$

**where**  
*refl* [*intro!*, *simp*]:  
*carrier-full-amalg-equiv*  $G1\ G2\ K\ i1\ i2\ mult1\ one1\ mult2\ one2\ w\ w$   
| *sym*:  
*carrier-full-amalg-equiv*  $G1\ G2\ K\ i1\ i2\ mult1\ one1\ mult2\ one2\ u\ v \Longrightarrow$   
*carrier-full-amalg-equiv*  $G1\ G2\ K\ i1\ i2\ mult1\ one1\ mult2\ one2\ v\ u$   
| *trans*:  
*carrier-full-amalg-equiv*  $G1\ G2\ K\ i1\ i2\ mult1\ one1\ mult2\ one2\ u\ v \Longrightarrow$   
*carrier-full-amalg-equiv*  $G1\ G2\ K\ i1\ i2\ mult1\ one1\ mult2\ one2\ v\ w \Longrightarrow$   
*carrier-full-amalg-equiv*  $G1\ G2\ K\ i1\ i2\ mult1\ one1\ mult2\ one2\ u\ w$   
| *of-amalg*:  
*carrier-amalgam-equiv*  $K\ i1\ i2\ u\ v \Longrightarrow$   
*carrier-full-amalg-equiv*  $G1\ G2\ K\ i1\ i2\ mult1\ one1\ mult2\ one2\ u\ v$   
| *of-reduction*:  
*carrier-fpw-reduction*  $G1\ G2\ mult1\ one1\ mult2\ one2\ u\ v \Longrightarrow$   
*carrier-full-amalg-equiv*  $G1\ G2\ K\ i1\ i2\ mult1\ one1\ mult2\ one2\ u\ v$

**interpretation** *carrier-full-amalg-equiv-equiv*:  
*equivalence-relation*  
*carrier-full-amalg-equiv*  $G1\ G2\ K\ i1\ i2\ mult1\ one1\ mult2\ one2$   
*<proof>*

**definition** *carrier-full-amalg-class* ::

*'a set => 'b set => 'h set => ('h => 'a) => ('h => 'b) =>*  
*('a => 'a => 'a) => 'a => ('b => 'b => 'b) => 'b =>*  
*('a, 'b) free-product-word => (('a, 'b) free-product-word) set*

**where**

*carrier-full-amalg-class G1 G2 K i1 i2 mult1 one1 mult2 one2 w =*  
*equiv-class (carrier-full-amalg-equiv G1 G2 K i1 i2 mult1 one1 mult2 one2) w*

**definition** *carrier-full-amalgamated-free-product-space* ::

*'a set => 'b set => 'h set => ('h => 'a) => ('h => 'b) =>*  
*('a => 'a => 'a) => 'a => ('b => 'b => 'b) => 'b =>*  
*((a, 'b) free-product-word) set set*

**where**

*carrier-full-amalgamated-free-product-space G1 G2 K i1 i2 mult1 one1 mult2 one2*  
*=*  
*carrier-full-amalg-class G1 G2 K i1 i2 mult1 one1 mult2 one2 ' carrier-fpw-space*  
*G1 G2*

**lemma** *carrier-full-amalg-class-eq-iff*:

*carrier-full-amalg-class G1 G2 K i1 i2 mult1 one1 mult2 one2 u =*  
*carrier-full-amalg-class G1 G2 K i1 i2 mult1 one1 mult2 one2 v*  
*⟷ carrier-full-amalg-equiv G1 G2 K i1 i2 mult1 one1 mult2 one2 u v*  
*<proof>*

**lemma** *carrier-full-amalg-class-in-space* [intro]:

**assumes** *fpw-in-space G1 G2 w*

**shows**

*carrier-full-amalg-class G1 G2 K i1 i2 mult1 one1 mult2 one2 w ∈*  
*carrier-full-amalgamated-free-product-space G1 G2 K i1 i2 mult1 one1 mult2*  
*one2*  
*<proof>*

**definition** *carrier-full-amalg-one* ::

*'a set => 'b set => 'h set => ('h => 'a) => ('h => 'b) =>*  
*('a => 'a => 'a) => 'a => ('b => 'b => 'b) => 'b =>*  
*((a, 'b) free-product-word) set*

**where**

*carrier-full-amalg-one G1 G2 K i1 i2 mult1 one1 mult2 one2 =*  
*carrier-full-amalg-class G1 G2 K i1 i2 mult1 one1 mult2 one2 WordNil*

**end**

**theory** *Carrier-Group-Scaffold*

**imports** *Main*

**begin**

## 8 Groups on carriers

The main Seifert–van Kampen statement is formulated on concrete carrier sets rather than on type-class groups. This locale package isolates the carrier algebra laws and homomorphism notions that the amalgamation and fundamental-group constructions need later on.

```
locale carrier-group =  
  fixes  $G :: 'a \text{ set}$   
    and  $\text{mult} :: 'a \Rightarrow 'a \Rightarrow 'a$   
    and  $\text{one} :: 'a$   
    and  $\text{inv} :: 'a \Rightarrow 'a$   
  assumes one-closed [intro, simp]:  $\text{one} \in G$   
    and mult-closed [intro]:  $\llbracket x \in G; y \in G \rrbracket \Longrightarrow \text{mult } x \ y \in G$   
    and inv-closed [intro]:  $x \in G \Longrightarrow \text{inv } x \in G$   
    and mult-assoc:  $\llbracket x \in G; y \in G; z \in G \rrbracket \Longrightarrow \text{mult } (\text{mult } x \ y) \ z = \text{mult } x \ (\text{mult } y \ z)$   
    and mult-one-left:  $x \in G \Longrightarrow \text{mult } \text{one } x = x$   
    and mult-one-right:  $x \in G \Longrightarrow \text{mult } x \ \text{one} = x$   
    and mult-inv-left:  $x \in G \Longrightarrow \text{mult } (\text{inv } x) \ x = \text{one}$   
    and mult-inv-right:  $x \in G \Longrightarrow \text{mult } x \ (\text{inv } x) = \text{one}$   
begin
```

```
lemma left-cancel:  
  assumes  $x: x \in G$   
    and  $y: y \in G$   
    and  $z: z \in G$   
    and  $\text{eq}: \text{mult } x \ y = \text{mult } x \ z$   
  shows  $y = z$   
<proof>
```

```
lemma right-cancel:  
  assumes  $x: x \in G$   
    and  $y: y \in G$   
    and  $z: z \in G$   
    and  $\text{eq}: \text{mult } y \ x = \text{mult } z \ x$   
  shows  $y = z$   
<proof>
```

```
lemma left-inverse-unique:  
  assumes  $x: x \in G$   
    and  $y: y \in G$   
    and  $\text{eq}: \text{mult } y \ x = \text{one}$   
  shows  $y = \text{inv } x$   
<proof>
```

```
lemma right-inverse-unique:  
  assumes  $x: x \in G$   
    and  $y: y \in G$   
    and  $\text{eq}: \text{mult } x \ y = \text{one}$ 
```

**shows**  $y = \text{inv } x$   
 $\langle \text{proof} \rangle$

**end**

**locale** *carrier-group-hom* =

*G*: *carrier-group* *G* *mult* *one* *inv* +

*H*: *carrier-group* *H* *mult'* *one'* *inv'*

**for** *G* :: '*a* set **and** *mult* :: '*a* => '*a* => '*a* **and** *one* :: '*a* **and** *inv* :: '*a* => '*a*

**and** *H* :: '*b* set **and** *mult'* :: '*b* => '*b* => '*b* **and** *one'* :: '*b* **and** *inv'* :: '*b* => '*b*

**and** *h* :: '*a* => '*b* +

**assumes** *map-closed*:  $x \in G \implies h x \in H$

**and** *map-mult*:  $\llbracket x \in G; y \in G \rrbracket \implies h (\text{mult } x y) = \text{mult}' (h x) (h y)$

**begin**

**lemma** *map-one*:

$h \text{ one} = \text{one}'$

$\langle \text{proof} \rangle$

**lemma** *map-inv*:

**assumes** *x*:  $x \in G$

**shows**  $h (\text{inv } x) = \text{inv}' (h x)$

$\langle \text{proof} \rangle$

**end**

**end**

**theory** *Carrier-Amalgamated-Free-Product-Eval*

**imports** *Carrier-Amalgamated-Free-Product* *Carrier-Group-Scaffold*

**begin**

## 9 Evaluating carrier amalgamated free-product words

Once the carrier-side amalgamated words have been defined, one still needs a way to evaluate them in a target carrier group. The evaluation locale proved here is the algebraic engine behind the later decoding maps in both the topological and classical Seifert–van Kampen statements.

**locale** *carrier-full-amalg-word-eval* =

*Grp1*: *carrier-group* *G1* *mult1* *one1* *inv1* +

*Grp2*: *carrier-group* *G2* *mult2* *one2* *inv2* +

*Cod*: *carrier-group* *K* *multK* *oneK* *invK* +

*J1*: *carrier-group-hom* *G1* *mult1* *one1* *inv1* *K* *multK* *oneK* *invK* *j1* +

*J2*: *carrier-group-hom* *G2* *mult2* *one2* *inv2* *K* *multK* *oneK* *invK* *j2*

**for** *G1* :: '*a* set

**and** *mult1* :: '*a* => '*a* => '*a*

**and** *one1* :: '*a*

```

and inv1 :: 'a => 'a
and G2 :: 'b set
and mult2 :: 'b => 'b => 'b
and one2 :: 'b
and inv2 :: 'b => 'b
and H :: 'h set
and i1 :: 'h => 'a
and i2 :: 'h => 'b
and K :: 'k set
and multK :: 'k => 'k => 'k
and oneK :: 'k
and invK :: 'k => 'k
and j1 :: 'a => 'k
and j2 :: 'b => 'k +
assumes i1-closed:  $h \in H \implies i1\ h \in G1$ 
and i2-closed:  $h \in H \implies i2\ h \in G2$ 
and agree:  $h \in H \implies j1\ (i1\ h) = j2\ (i2\ h)$ 
begin

fun carrier-full-amalg-eval :: ('a, 'b) free-product-word => 'k where
  carrier-full-amalg-eval WordNil = oneK
| carrier-full-amalg-eval (WordLeft a rest) =
  multK (j1 a) (carrier-full-amalg-eval rest)
| carrier-full-amalg-eval (WordRight b rest) =
  multK (j2 b) (carrier-full-amalg-eval rest)

lemma carrier-full-amalg-eval-in-carrier:
assumes fpw-in-space G1 G2 w
shows carrier-full-amalg-eval w  $\in K$ 
  <proof>

lemma carrier-amalgam-step-preserves-space-iff:
assumes step: carrier-amalgam-step H i1 i2 u v
shows fpw-in-space G1 G2 u  $\longleftrightarrow$  fpw-in-space G1 G2 v
  <proof>

lemma carrier-amalgam-equiv-preserves-space-iff:
assumes rel: carrier-amalgam-equiv H i1 i2 u v
shows fpw-in-space G1 G2 u  $\longleftrightarrow$  fpw-in-space G1 G2 v
  <proof>

lemma carrier-amalgam-step-preserves-eval:
assumes step: carrier-amalgam-step H i1 i2 u v
shows carrier-full-amalg-eval u = carrier-full-amalg-eval v
  <proof>

lemma carrier-amalgam-equiv-preserves-eval:
assumes rel: carrier-amalgam-equiv H i1 i2 u v
shows carrier-full-amalg-eval u = carrier-full-amalg-eval v

```

*<proof>*

**lemma** *carrier-fpw-reduction-step-preserves-eval:*

**assumes** *step: carrier-fpw-reduction-step G1 G2 mult1 one1 mult2 one2 u v*

**shows** *carrier-full-amalg-eval u = carrier-full-amalg-eval v*

*<proof>*

**lemmas** *carrier-fpw-reduction-space-iff =*

*Carrier-Amalgamated-Free-Product.carrier-fpw-reduction-preserves-space-iff*

**lemma** *carrier-fpw-reduction-preserves-space-iff:*

**assumes** *rel: carrier-fpw-reduction G1 G2 mult1 one1 mult2 one2 u v*

**shows** *fpw-in-space G1 G2 u  $\longleftrightarrow$  fpw-in-space G1 G2 v*

*<proof>*

**lemma** *carrier-fpw-reduction-preserves-eval:*

**assumes** *rel: carrier-fpw-reduction G1 G2 mult1 one1 mult2 one2 u v*

**shows** *carrier-full-amalg-eval u = carrier-full-amalg-eval v*

*<proof>*

**lemma** *carrier-full-amalg-equiv-preserves-space-iff:*

**assumes** *rel: carrier-full-amalg-equiv G1 G2 H i1 i2 mult1 one1 mult2 one2 u v*

**shows** *fpw-in-space G1 G2 u  $\longleftrightarrow$  fpw-in-space G1 G2 v*

*<proof>*

**lemma** *carrier-full-amalg-equiv-preserves-eval:*

**assumes** *rel: carrier-full-amalg-equiv G1 G2 H i1 i2 mult1 one1 mult2 one2 u v*

**shows** *carrier-full-amalg-eval u = carrier-full-amalg-eval v*

*<proof>*

**definition** *carrier-full-amalg-has-good-rep :: ('a, 'b) free-product-word  $\Rightarrow$  bool*

**where**

*carrier-full-amalg-has-good-rep w  $\longleftrightarrow$*

*( $\exists v$ . carrier-full-amalg-equiv G1 G2 H i1 i2 mult1 one1 mult2 one2 w v*

*$\wedge$  fpw-in-space G1 G2 v)*

**definition** *carrier-full-amalg-some-good-rep ::*

*('a, 'b) free-product-word  $\Rightarrow$  ('a, 'b) free-product-word*

**where**

*carrier-full-amalg-some-good-rep w =*

*(SOME v. carrier-full-amalg-equiv G1 G2 H i1 i2 mult1 one1 mult2 one2 w v*

*$\wedge$  fpw-in-space G1 G2 v)*

**definition** *carrier-full-amalg-decode :: ('a, 'b) free-product-word  $\Rightarrow$  'k* **where**

*carrier-full-amalg-decode w =*

*(if carrier-full-amalg-has-good-rep w*

*then carrier-full-amalg-eval (carrier-full-amalg-some-good-rep w)*

*else oneK)*

```

lemma carrier-full-amalg-has-good-repI:
  assumes fpw-in-space G1 G2 w
  shows carrier-full-amalg-has-good-rep w
  ⟨proof⟩

lemma carrier-full-amalg-some-good-rep:
  assumes carrier-full-amalg-has-good-rep w
  shows carrier-full-amalg-equiv G1 G2 H i1 i2 mult1 one1 mult2 one2
  w (carrier-full-amalg-some-good-rep w)
  and fpw-in-space G1 G2 (carrier-full-amalg-some-good-rep w)
  ⟨proof⟩

lemma carrier-full-amalg-has-good-rep-respects:
  assumes wv: carrier-full-amalg-equiv G1 G2 H i1 i2 mult1 one1 mult2 one2 u v
  shows carrier-full-amalg-has-good-rep u ↔ carrier-full-amalg-has-good-rep v
  ⟨proof⟩

lemma carrier-full-amalg-decode-in-carrier:
  carrier-full-amalg-decode w ∈ K
  ⟨proof⟩

lemma carrier-full-amalg-decode-respects:
  assumes wv: carrier-full-amalg-equiv G1 G2 H i1 i2 mult1 one1 mult2 one2 u v
  shows carrier-full-amalg-decode u = carrier-full-amalg-decode v
  ⟨proof⟩

lemma carrier-full-amalg-decode-eq-eval:
  assumes w-in: fpw-in-space G1 G2 w
  shows carrier-full-amalg-decode w = carrier-full-amalg-eval w
  ⟨proof⟩

end

end

theory Explicit-Path-Homotopy-Scaffold
  imports HOL-Analysis.Homotopy
begin

```

## 10 Explicit-topology paths and homotopies

HOL-Analysis already provides paths and homotopies for subsets of a type. For the pushout-oriented development, however, it is convenient to work directly with arbitrary topologies. This theory therefore rephrases the basic path and homotopy notions in explicit-topology form.

```

definition loopin-space ::
  'a topology ⇒ 'a ⇒ (real ⇒ 'a) set
where
  loopin-space X x = {p. pathin X p ∧ p 0 = x ∧ p 1 = x}

```

**definition** *homotopic-paths* ::

$'a \text{ topology} \Rightarrow (\text{real} \Rightarrow 'a) \Rightarrow (\text{real} \Rightarrow 'a) \Rightarrow \text{bool}$

**where**

$\text{homotopic-paths} X p q \equiv$

$\text{homotopic-with } (\lambda r. r 0 = p 0 \wedge r 1 = p 1) (\text{top-of-set } \{0..1\}) X p q$

**definition** *homotopic-loops* ::

$'a \text{ topology} \Rightarrow (\text{real} \Rightarrow 'a) \Rightarrow (\text{real} \Rightarrow 'a) \Rightarrow \text{bool}$

**where**

$\text{homotopic-loops} X p q \equiv$

$\text{homotopic-with } (\lambda r. r 1 = r 0) (\text{top-of-set } \{0..1\}) X p q$

**definition** *reversepath* ::  $(\text{real} \Rightarrow 'a) \Rightarrow \text{real} \Rightarrow 'a$

**where**

$\text{reversepath} p = (\lambda t. p (1 - t))$

**definition** *joinpath* ::  $(\text{real} \Rightarrow 'a) \Rightarrow (\text{real} \Rightarrow 'a) \Rightarrow \text{real} \Rightarrow 'a$

**where**

$\text{joinpath} p q = (\lambda t. \text{if } t \leq 1 / 2 \text{ then } p (2 * t) \text{ else } q (2 * t - 1))$

**definition** *subpath* ::  $\text{real} \Rightarrow \text{real} \Rightarrow (\text{real} \Rightarrow 'a) \Rightarrow \text{real} \Rightarrow 'a$

**where**

$\text{subpath} u v p = (\lambda t. p ((v - u) * t + u))$

**lemma** *reversepath-eq-reversepath* [*simp*]:

$\text{reversepath} p = \text{reversepath} p$

$\langle \text{proof} \rangle$

**lemma** *reversepath-reversepath* [*simp*]:

$\text{reversepath} (\text{reversepath} p) = p$

$\langle \text{proof} \rangle$

**lemma** *joinpath-eq-joinpaths* [*simp*]:

$\text{joinpath} p q = p +++ q$

$\langle \text{proof} \rangle$

**lemma** *subpath-image*:

$\text{subpath} u v p \text{ ' } \{0..1\} = p \text{ ' } \text{closed-segment } u v$

$\langle \text{proof} \rangle$

**lemma** *subpath-image-eq*:

**assumes**  $u \leq v$

**shows**  $\text{subpath} u v p \text{ ' } \{0..1\} = p \text{ ' } \{u..v\}$

$\langle \text{proof} \rangle$

**lemma** *subpath-0-1* [*simp*]:

$\text{subpath} 0 1 p = p$

$\langle \text{proof} \rangle$

**lemma** *joinpathin-subpathin-middle* [simp]:  
 $joinpathin (subpathin 0 (1 / 2) p) (subpathin (1 / 2) 1 p) = p$   
 ⟨proof⟩

**lemma** *continuous-map-top01-id*:  
 $continuous-map (top-of-set \{0..1\}) euclideanreal id$   
 ⟨proof⟩

**lemma** *loopin-spaceI*:  
 assumes  $pathin X p$   
 and  $p 0 = x$   
 and  $p 1 = x$   
 shows  $p \in loopin-space X x$   
 ⟨proof⟩

**lemma** *loopin-spaceE*:  
 assumes  $p \in loopin-space X x$   
 obtains  $pathin X p$   $p 0 = x$   $p 1 = x$   
 ⟨proof⟩

**lemma** *constant-loopin-in-space*:  
 assumes  $x \in topspace X$   
 shows  $(\lambda \cdot. x) \in loopin-space X x$   
 ⟨proof⟩

**lemma** *pathin-image-subset-topospace*:  
 assumes  $pathin X p$   
 shows  $p \text{ ' } \{0..1\} \subseteq topspace X$   
 ⟨proof⟩

**lemma** *pathin-reversepathin*:  
 assumes  $pathin X p$   
 shows  $pathin X (reversepathin p)$   
 ⟨proof⟩

**lemma** *pathin-joinpathin*:  
 assumes  $p: pathin X p$   
 and  $q: pathin X q$   
 and  $pq: p 1 = q 0$   
 shows  $pathin X (joinpathin p q)$   
 ⟨proof⟩

**lemma** *pathin-subpathin*:  
 assumes  $p: pathin X p$   
 and  $u: u \in \{0..1\}$   
 and  $v: v \in \{0..1\}$   
 shows  $pathin X (subpathin u v p)$   
 ⟨proof⟩

**lemma** *pathin-subdivision-open-cover*:  
**assumes**  $p$ :  $\text{pathin } X \ p$   
**and**  $\text{cover}$ :  $p \text{ ' } \{0..1\} \subseteq \bigcup \mathcal{U}$   
**and**  $\text{openU}$ :  $\bigwedge U. U \in \mathcal{U} \implies \text{openin } X \ U$   
**shows**  $\exists n::\text{nat}. 0 < n \wedge$   
 $(\forall i < n. \exists U \in \mathcal{U}.$   
 $\text{subpathin } (\text{real } i / \text{real } n) (\text{real } (\text{Suc } i) / \text{real } n) \ p \text{ ' } \{0..1\} \subseteq U)$   
 $\langle \text{proof} \rangle$

**lemma** *loopin-space-joinpathin*:  
**assumes**  $p$ :  $p \in \text{loopin-space } X \ x$   
**and**  $q$ :  $q \in \text{loopin-space } X \ x$   
**shows**  $\text{joinpathin } p \ q \in \text{loopin-space } X \ x$   
 $\langle \text{proof} \rangle$

**lemma** *homotopic-paths-in-top-of-set* [*simp*]:  
 $\text{homotopic-paths-in } (\text{top-of-set } S) \ p \ q \longleftrightarrow \text{homotopic-paths } S \ p \ q$   
 $\langle \text{proof} \rangle$

**lemma** *homotopic-loops-in-top-of-set* [*simp*]:  
 $\text{homotopic-loops-in } (\text{top-of-set } S) \ p \ q \longleftrightarrow \text{homotopic-loops } S \ p \ q$   
 $\langle \text{proof} \rangle$

**lemma** *homotopic-paths-in-imp-pathin*:  
**assumes**  $\text{homotopic-paths-in } X \ p \ q$   
**shows**  $\text{pathin } X \ p \ \text{pathin } X \ q$   
 $\langle \text{proof} \rangle$

**lemma** *homotopic-paths-in-imp-endpoints*:  
**assumes**  $\text{homotopic-paths-in } X \ p \ q$   
**shows**  $q \ 0 = p \ 0 \ q \ 1 = p \ 1$   
 $\langle \text{proof} \rangle$

**lemma** *homotopic-paths-in-refl* [*simp*]:  
 $\text{homotopic-paths-in } X \ p \ p \longleftrightarrow \text{pathin } X \ p$   
 $\langle \text{proof} \rangle$

**lemma** *homotopic-paths-in-sym*:  
**assumes**  $\text{homotopic-paths-in } X \ p \ q$   
**shows**  $\text{homotopic-paths-in } X \ q \ p$   
 $\langle \text{proof} \rangle$

**lemma** *homotopic-paths-in-trans*:  
**assumes**  $\text{homotopic-paths-in } X \ p \ q$   
**and**  $\text{homotopic-paths-in } X \ q \ r$   
**shows**  $\text{homotopic-paths-in } X \ p \ r$   
 $\langle \text{proof} \rangle$

**lemma** *homotopic-paths-in-eq*:  
**assumes** *pathin*  $X$   $p$   
**and**  $\bigwedge t. t \in \{0..1\} \implies p\ t = q\ t$   
**shows** *homotopic-paths-in*  $X$   $p$   $q$   
 $\langle$ *proof* $\rangle$

**lemma** *continuous-map-homotopic-joinpathin-lemma*:  
**fixes**  $p\ q :: \text{real} \Rightarrow \text{real} \Rightarrow 'a$   
**assumes**  $p$ :  
  *continuous-map* (*prod-topology* (*top-of-set*  $\{0..1\}$ ) (*top-of-set*  $\{0..1\}$ ))  $X$   
   $(\lambda y. p\ (fst\ y)\ (snd\ y))$   
**and**  $q$ :  
  *continuous-map* (*prod-topology* (*top-of-set*  $\{0..1\}$ ) (*top-of-set*  $\{0..1\}$ ))  $X$   
   $(\lambda y. q\ (fst\ y)\ (snd\ y))$   
**and**  $pq$ :  $\bigwedge t. t \in \{0..1\} \implies p\ t\ 1 = q\ t\ 0$   
**shows**  
  *continuous-map* (*prod-topology* (*top-of-set*  $\{0..1\}$ ) (*top-of-set*  $\{0..1\}$ ))  $X$   
   $(\lambda y. \text{joinpathin}\ (p\ (fst\ y))\ (q\ (fst\ y))\ (snd\ y))$   
 $\langle$ *proof* $\rangle$

**lemma** *homotopic-paths-in-continuous-image*:  
**assumes**  $pq$ : *homotopic-paths-in*  $X$   $p$   $q$   
**and**  $h$ : *continuous-map*  $X$   $Y$   $f$   
**shows** *homotopic-paths-in*  $Y$   $(f \circ p)$   $(f \circ q)$   
 $\langle$ *proof* $\rangle$

**lemma** *homotopic-paths-in-reversepathin-D*:  
**assumes** *homotopic-paths-in*  $X$   $p$   $q$   
**shows** *homotopic-paths-in*  $X$  (*reversepathin*  $p$ ) (*reversepathin*  $q$ )  
 $\langle$ *proof* $\rangle$

**lemma** *homotopic-paths-in-reversepathin*:  
*homotopic-paths-in*  $X$  (*reversepathin*  $p$ ) (*reversepathin*  $q$ )  $\longleftrightarrow$  *homotopic-paths-in*  
 $X$   $p$   $q$   
 $\langle$ *proof* $\rangle$

**lemma** *homotopic-paths-in-joinpathin*:  
**assumes**  $pp'$ : *homotopic-paths-in*  $X$   $p$   $p'$   
**and**  $qq'$ : *homotopic-paths-in*  $X$   $q$   $q'$   
**and**  $pq$ :  $p\ 1 = q\ 0$   
**shows** *homotopic-paths-in*  $X$  (*joinpathin*  $p$   $q$ ) (*joinpathin*  $p'$   $q'$ )  
 $\langle$ *proof* $\rangle$

**lemma** *homotopic-paths-in-reparametrize*:  
**assumes**  $p$ : *pathin*  $X$   $p$   
**and**  $contf$ : *continuous-map* (*top-of-set*  $\{0..1\}$ ) (*top-of-set*  $\{0..1\}$ )  $f$   
**and**  $f01$ :  $f \in \{0..1\} \rightarrow \{0..1\}$   
**and**  $[simp]$ :  $f\ 0 = 0$   $f\ 1 = 1$   
**and**  $q$ :  $\bigwedge t. t \in \{0..1\} \implies q\ t = p\ (f\ t)$

**shows** *homotopic-paths*in  $X$   $p$   $q$   
*<proof>*

**lemma** *homotopic-paths*in-*rid-const*:  
**assumes**  $p$ : *path*in  $X$   $p$   
**shows** *homotopic-paths*in  $X$  (*joinpath*in  $p$  ( $\lambda$ -.  $p$  1))  $p$   
*<proof>*

**lemma** *homotopic-paths*in-*lid-const*:  
**assumes**  $p$ : *path*in  $X$   $p$   
**shows** *homotopic-paths*in  $X$  (*joinpath*in ( $\lambda$ -.  $p$  0)  $p$ )  $p$   
*<proof>*

**lemma** *homotopic-paths*in-*assoc*:  
**assumes**  $p$ : *path*in  $X$   $p$   
**and**  $q$ : *path*in  $X$   $q$   
**and**  $r$ : *path*in  $X$   $r$   
**and**  $pq$ :  $p$  1 =  $q$  0  
**and**  $qr$ :  $q$  1 =  $r$  0  
**shows** *homotopic-paths*in  $X$  (*joinpath*in  $p$  (*joinpath*in  $q$   $r$ )) (*joinpath*in (*joinpath*in  $p$   $q$ )  $r$ )  
*<proof>*

**lemma** *homotopic-paths*in-*rinv-const*:  
**assumes**  $p$ : *path*in  $X$   $p$   
**shows** *homotopic-paths*in  $X$  (*joinpath*in  $p$  (*reversepath*in  $p$ )) ( $\lambda$ -.  $p$  0)  
*<proof>*

**lemma** *homotopic-paths*in-*linv-const*:  
**assumes**  $p$ : *path*in  $X$   $p$   
**shows** *homotopic-paths*in  $X$  (*joinpath*in (*reversepath*in  $p$ )  $p$ ) ( $\lambda$ -.  $p$  1)  
*<proof>*

**lemma** *homotopic-loops*in-*imp-path*in:  
**assumes** *homotopic-loops*in  $X$   $p$   $q$   
**shows** *path*in  $X$   $p$  *path*in  $X$   $q$   
*<proof>*

**lemma** *homotopic-loops*in-*imp-loop*:  
**assumes** *homotopic-loops*in  $X$   $p$   $q$   
**shows**  $p$  1 =  $p$  0  $q$  1 =  $q$  0  
*<proof>*

**lemma** *homotopic-loops*in-*refl* [*simp*]:  
*homotopic-loops*in  $X$   $p$   $p$   $\longleftrightarrow$  *path*in  $X$   $p$   $\wedge$   $p$  1 =  $p$  0  
*<proof>*

**lemma** *homotopic-loops*in-*sym*:  
**assumes** *homotopic-loops*in  $X$   $p$   $q$

```

shows homotopic-loopsin  $X$   $q$   $p$ 
  <proof>

lemma homotopic-loopsin-trans:
  assumes homotopic-loopsin  $X$   $p$   $q$ 
    and homotopic-loopsin  $X$   $q$   $r$ 
  shows homotopic-loopsin  $X$   $p$   $r$ 
  <proof>

lemma loopin-space-reversepathin:
  assumes  $p \in \text{loopin-space } X$   $x$ 
  shows reversepathin  $p \in \text{loopin-space } X$   $x$ 
  <proof>

end
theory Fundamental-Group-Scaffold
  imports HOL-Analysis.Homotopy Carrier-Group-Scaffold
begin

```

## 11 Fundamental-group quotients

This is the set-based counterpart to the explicit-topology quotient development. It works directly with subsets of a topological space and the standard HOL-Analysis path notions, which is exactly the level needed for the final classical theorem about open sets  $U$  and  $V$ .

```

definition loop-space ::
  'a::topological-space set => 'a => (real => 'a) set
where
  loop-space  $S$   $x$  =
    { $p$ . path  $p \wedge \text{path-image } p \subseteq S \wedge \text{pathstart } p = x \wedge \text{pathfinish } p = x$ }

```

```

definition loop-class ::
  'a::topological-space set => 'a => (real => 'a) => (real => 'a) set
where
  loop-class  $S$   $x$   $p$  = { $q$ .  $q \in \text{loop-space } S$   $x \wedge \text{homotopic-paths } S$   $q$   $p$ }

```

```

definition fundamental-group-space ::
  'a::topological-space set => 'a => ((real => 'a) set) set
where
  fundamental-group-space  $S$   $x$  = loop-class  $S$   $x$  ' loop-space  $S$   $x$ 

```

```

definition some-loop ::
  'a::topological-space set => 'a => (real => 'a) set => (real => 'a)
where
  some-loop  $S$   $x$   $Q$  = (SOME  $p$ .  $p \in \text{loop-space } S$   $x \wedge Q = \text{loop-class } S$   $x$   $p$ )

```

```

definition fundamental-group-one ::
  'a::topological-space set => 'a => (real => 'a) set

```

**where**

*fundamental-group-one*  $S x = \text{loop-class } S x (\lambda-. x)$

**definition** *fundamental-group-mult* ::

*'a::topological-space set => 'a =>*  
*(real => 'a) set => (real => 'a) set => (real => 'a) set*

**where**

*fundamental-group-mult*  $S x A B =$   
*loop-class*  $S x (\text{some-loop } S x A +++ \text{some-loop } S x B)$

**definition** *fundamental-group-inv* ::

*'a::topological-space set => 'a =>*  
*(real => 'a) set => (real => 'a) set*

**where**

*fundamental-group-inv*  $S x A = \text{loop-class } S x (\text{reversepath } (\text{some-loop } S x A))$

**definition** *loop-image* ::

*('a::topological-space => 'b::topological-space) =>*  
*(real => 'a) => (real => 'b)*

**where**

*loop-image*  $h p = h \circ p$

**definition** *fundamental-group-map* ::

*'a::topological-space set => 'a =>*  
*'b::topological-space set => 'b =>*  
*('a => 'b) => (real => 'a) set => (real => 'b) set*

**where**

*fundamental-group-map*  $S x T y h A =$   
*loop-class*  $T y (\text{loop-image } h (\text{some-loop } S x A))$

**lemma** *loop-spaceI*:

**assumes** *path*  $p$   
**and** *path-image*  $p \subseteq S$   
**and** *pathstart*  $p = x$   
**and** *pathfinish*  $p = x$   
**shows**  $p \in \text{loop-space } S x$   
*<proof>*

**lemma** *constant-loop-in-space*:

**assumes**  $x \in S$   
**shows**  $(\lambda-. x) \in \text{loop-space } S x$   
*<proof>*

**lemma** *loop-class-in-space*:

**assumes**  $p \in \text{loop-space } S x$   
**shows** *loop-class*  $S x p \in \text{fundamental-group-space } S x$   
*<proof>*

**lemma** *fundamental-group-spaceE*:

**assumes**  $Q \in \text{fundamental-group-space } S \ x$   
**obtains**  $p$  **where**  $p \in \text{loop-space } S \ x \ Q = \text{loop-class } S \ x \ p$   
 $\langle \text{proof} \rangle$

**lemma** *some-loop-spec*:  
**assumes**  $Q \in \text{fundamental-group-space } S \ x$   
**shows**  $\text{some-loop } S \ x \ Q \in \text{loop-space } S \ x$   
**and**  $Q = \text{loop-class } S \ x \ (\text{some-loop } S \ x \ Q)$   
 $\langle \text{proof} \rangle$

**lemma** *loop-class-eqI*:  
**assumes**  $p: p \in \text{loop-space } S \ x$   
**and**  $q: q \in \text{loop-space } S \ x$   
**and**  $pq: \text{homotopic-paths } S \ p \ q$   
**shows**  $\text{loop-class } S \ x \ p = \text{loop-class } S \ x \ q$   
 $\langle \text{proof} \rangle$

**lemma** *loop-class-eq-iff*:  
**assumes**  $p: p \in \text{loop-space } S \ x$   
**and**  $q: q \in \text{loop-space } S \ x$   
**shows**  $\text{loop-class } S \ x \ p = \text{loop-class } S \ x \ q \longleftrightarrow \text{homotopic-paths } S \ p \ q$   
 $\langle \text{proof} \rangle$

**lemma** *loop-space-join*:  
**assumes**  $p: p \in \text{loop-space } S \ x$   
**and**  $q: q \in \text{loop-space } S \ x$   
**shows**  $p \ +++ \ q \in \text{loop-space } S \ x$   
 $\langle \text{proof} \rangle$

**lemma** *loop-space-reversepath*:  
**assumes**  $p \in \text{loop-space } S \ x$   
**shows**  $\text{reversepath } p \in \text{loop-space } S \ x$   
 $\langle \text{proof} \rangle$

**lemma** *loop-space-continuous-image*:  
**assumes**  $p: p \in \text{loop-space } S \ x$   
**and**  $\text{cont-h}: \text{continuous-on } S \ h$   
**and**  $\text{map-h}: h \in S \rightarrow T$   
**and**  $hx: h \ x = y$   
**shows**  $\text{loop-image } h \ p \in \text{loop-space } T \ y$   
 $\langle \text{proof} \rangle$

**lemma** *loop-image-join*:  
 $\text{loop-image } h \ (p \ +++ \ q) = \text{loop-image } h \ p \ +++ \ \text{loop-image } h \ q$   
 $\langle \text{proof} \rangle$

**lemma** *loop-image-reversepath*:  
 $\text{loop-image } h \ (\text{reversepath } p) = \text{reversepath } (\text{loop-image } h \ p)$   
 $\langle \text{proof} \rangle$

**lemma** *loop-class-join-eqI*:  
**assumes**  $p: p \in \text{loop-space } S \ x$   
**and**  $p': p' \in \text{loop-space } S \ x$   
**and**  $q: q \in \text{loop-space } S \ x$   
**and**  $q': q' \in \text{loop-space } S \ x$   
**and**  $pp': \text{homotopic-paths } S \ p \ p'$   
**and**  $qq': \text{homotopic-paths } S \ q \ q'$   
**shows**  $\text{loop-class } S \ x \ (p \ +++ \ q) = \text{loop-class } S \ x \ (p' \ +++ \ q')$   
 $\langle \text{proof} \rangle$

**lemma** *loop-class-reverse-eqI*:  
**assumes**  $p: p \in \text{loop-space } S \ x$   
**and**  $q: q \in \text{loop-space } S \ x$   
**and**  $pq: \text{homotopic-paths } S \ p \ q$   
**shows**  $\text{loop-class } S \ x \ (\text{reversepath } p) = \text{loop-class } S \ x \ (\text{reversepath } q)$   
 $\langle \text{proof} \rangle$

**lemma** *homotopic-paths-rid-const*:  
**assumes**  $\text{path } p$   
**and**  $\text{path-image } p \subseteq S$   
**shows**  $\text{homotopic-paths } S \ (p \ +++ \ (\lambda-. \text{pathfinish } p)) \ p$   
 $\langle \text{proof} \rangle$

**lemma** *homotopic-paths-lid-const*:  
**assumes**  $\text{path } p$   
**and**  $\text{path-image } p \subseteq S$   
**shows**  $\text{homotopic-paths } S \ ((\lambda-. \text{pathstart } p) \ +++ \ p) \ p$   
 $\langle \text{proof} \rangle$

**lemma** *homotopic-paths-rinv-const*:  
**assumes**  $\text{path } p$   
**and**  $\text{path-image } p \subseteq S$   
**shows**  $\text{homotopic-paths } S \ (p \ +++ \ \text{reversepath } p) \ (\lambda-. \text{pathstart } p)$   
 $\langle \text{proof} \rangle$

**lemma** *homotopic-paths-linv-const*:  
**assumes**  $\text{path } p$   
**and**  $\text{path-image } p \subseteq S$   
**shows**  $\text{homotopic-paths } S \ (\text{reversepath } p \ +++ \ p) \ (\lambda-. \text{pathfinish } p)$   
 $\langle \text{proof} \rangle$

**lemma** *fundamental-group-one-in-space*:  
**assumes**  $x \in S$   
**shows**  $\text{fundamental-group-one } S \ x \in \text{fundamental-group-space } S \ x$   
 $\langle \text{proof} \rangle$

**lemma** *fundamental-group-mult-eqI*:  
**assumes**  $A\text{-in}: A \in \text{fundamental-group-space } S \ x$

**and**  $B\text{-in}$ :  $B \in \text{fundamental-group-space } S \ x$   
**and**  $p$ :  $p \in \text{loop-space } S \ x$   
**and**  $q$ :  $q \in \text{loop-space } S \ x$   
**and**  $A$ :  $A = \text{loop-class } S \ x \ p$   
**and**  $B$ :  $B = \text{loop-class } S \ x \ q$   
**shows**  $\text{fundamental-group-mult } S \ x \ A \ B = \text{loop-class } S \ x \ (p \ +++ \ q)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fundamental-group-mult-in-space}$ :  
**assumes**  $A \in \text{fundamental-group-space } S \ x$   
**and**  $B \in \text{fundamental-group-space } S \ x$   
**shows**  $\text{fundamental-group-mult } S \ x \ A \ B \in \text{fundamental-group-space } S \ x$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fundamental-group-inv-eqI}$ :  
**assumes**  $A\text{-in}$ :  $A \in \text{fundamental-group-space } S \ x$   
**and**  $p$ :  $p \in \text{loop-space } S \ x$   
**and**  $A$ :  $A = \text{loop-class } S \ x \ p$   
**shows**  $\text{fundamental-group-inv } S \ x \ A = \text{loop-class } S \ x \ (\text{reversepath } p)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fundamental-group-inv-in-space}$ :  
**assumes**  $A \in \text{fundamental-group-space } S \ x$   
**shows**  $\text{fundamental-group-inv } S \ x \ A \in \text{fundamental-group-space } S \ x$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fundamental-group-map-eqI}$ :  
**assumes**  $A\text{-in}$ :  $A \in \text{fundamental-group-space } S \ x$   
**and**  $p$ :  $p \in \text{loop-space } S \ x$   
**and**  $A$ :  $A = \text{loop-class } S \ x \ p$   
**and**  $\text{cont-h}$ :  $\text{continuous-on } S \ h$   
**and**  $\text{map-h}$ :  $h \in S \rightarrow T$   
**and**  $hx$ :  $h \ x = y$   
**shows**  $\text{fundamental-group-map } S \ x \ T \ y \ h \ A = \text{loop-class } T \ y \ (\text{loop-image } h \ p)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fundamental-group-map-in-space}$ :  
**assumes**  $A\text{-in}$ :  $A \in \text{fundamental-group-space } S \ x$   
**and**  $\text{cont-h}$ :  $\text{continuous-on } S \ h$   
**and**  $\text{map-h}$ :  $h \in S \rightarrow T$   
**and**  $hx$ :  $h \ x = y$   
**shows**  $\text{fundamental-group-map } S \ x \ T \ y \ h \ A \in \text{fundamental-group-space } T \ y$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fundamental-group-map-mult}$ :  
**assumes**  $A\text{-in}$ :  $A \in \text{fundamental-group-space } S \ x$   
**and**  $B\text{-in}$ :  $B \in \text{fundamental-group-space } S \ x$   
**and**  $\text{cont-h}$ :  $\text{continuous-on } S \ h$   
**and**  $\text{map-h}$ :  $h \in S \rightarrow T$

**and**  $hx: h x = y$   
**shows**  
 $fundamental-group-map\ S\ x\ T\ y\ h\ (fundamental-group-mult\ S\ x\ A\ B) =$   
 $fundamental-group-mult\ T\ y$   
 $(fundamental-group-map\ S\ x\ T\ y\ h\ A)$   
 $(fundamental-group-map\ S\ x\ T\ y\ h\ B)$   
 $\langle proof \rangle$

**lemma** *fundamental-group-map-id*:  
**assumes**  $A-in: A \in fundamental-group-space\ S\ x$   
**shows**  $fundamental-group-map\ S\ x\ S\ x\ id\ A = A$   
 $\langle proof \rangle$

**lemma** *fundamental-group-map-one*:  
**assumes**  $x-in: x \in S$   
**and**  $cont-h: continuous-on\ S\ h$   
**and**  $map-h: h \in S \rightarrow T$   
**and**  $hx: h x = y$   
**shows**  
 $fundamental-group-map\ S\ x\ T\ y\ h\ (fundamental-group-one\ S\ x) =$   
 $fundamental-group-one\ T\ y$   
 $\langle proof \rangle$

**lemma** *fundamental-group-map-compose*:  
**assumes**  $A-in: A \in fundamental-group-space\ S\ x$   
**and**  $cont-h: continuous-on\ S\ h$   
**and**  $map-h: h \in S \rightarrow T$   
**and**  $hx: h x = y$   
**and**  $cont-k: continuous-on\ T\ k$   
**and**  $map-k: k \in T \rightarrow U$   
**and**  $ky: k y = z$   
**shows**  
 $fundamental-group-map\ T\ y\ U\ z\ k\ (fundamental-group-map\ S\ x\ T\ y\ h\ A) =$   
 $fundamental-group-map\ S\ x\ U\ z\ (k \circ h)\ A$   
 $\langle proof \rangle$

**lemma** *fundamental-group-mult-one-left*:  
**assumes**  $x \in S$   
**and**  $A \in fundamental-group-space\ S\ x$   
**shows**  $fundamental-group-mult\ S\ x\ (fundamental-group-one\ S\ x)\ A = A$   
 $\langle proof \rangle$

**lemma** *fundamental-group-mult-one-right*:  
**assumes**  $x \in S$   
**and**  $A \in fundamental-group-space\ S\ x$   
**shows**  $fundamental-group-mult\ S\ x\ A\ (fundamental-group-one\ S\ x) = A$   
 $\langle proof \rangle$

**lemma** *fundamental-group-mult-assoc*:

**assumes** *A-in*:  $A \in \text{fundamental-group-space } S \ x$   
**and** *B-in*:  $B \in \text{fundamental-group-space } S \ x$   
**and** *C-in*:  $C \in \text{fundamental-group-space } S \ x$   
**shows**  $\text{fundamental-group-mult } S \ x \ A \ (\text{fundamental-group-mult } S \ x \ B \ C) =$   
 $\text{fundamental-group-mult } S \ x \ (\text{fundamental-group-mult } S \ x \ A \ B) \ C$   
 <proof>

**lemma** *fundamental-group-mult-inv-right*:  
**assumes**  $x \in S$   
**and**  $A \in \text{fundamental-group-space } S \ x$   
**shows**  $\text{fundamental-group-mult } S \ x \ A \ (\text{fundamental-group-inv } S \ x \ A) = \text{fundamental-group-one } S \ x$   
 <proof>

**lemma** *fundamental-group-mult-inv-left*:  
**assumes**  $x \in S$   
**and**  $A \in \text{fundamental-group-space } S \ x$   
**shows**  $\text{fundamental-group-mult } S \ x \ (\text{fundamental-group-inv } S \ x \ A) \ A = \text{fundamental-group-one } S \ x$   
 <proof>

**lemma** *trivial-loop-class-in-space*:  
**assumes**  $x \in S$   
**shows**  $\text{loop-class } S \ x \ (\lambda-. \ x) \in \text{fundamental-group-space } S \ x$   
 <proof>

**lemma** *fundamental-group-carrier-group*:  
**assumes** *x-in*:  $x \in S$   
**shows** *carrier-group*  
 (*fundamental-group-space*  $S \ x$ )  
 (*fundamental-group-mult*  $S \ x$ )  
 (*fundamental-group-one*  $S \ x$ )  
 (*fundamental-group-inv*  $S \ x$ )  
 <proof>

**lemma** *fundamental-group-map-carrier-group-hom*:  
**assumes** *x-in*:  $x \in S$   
**and** *cont-h*: *continuous-on*  $S \ h$   
**and** *map-h*:  $h \in S \rightarrow T$   
**and** *hx*:  $h \ x = y$   
**shows** *carrier-group-hom*  
 (*fundamental-group-space*  $S \ x$ )  
 (*fundamental-group-mult*  $S \ x$ )  
 (*fundamental-group-one*  $S \ x$ )  
 (*fundamental-group-inv*  $S \ x$ )  
 (*fundamental-group-space*  $T \ y$ )  
 (*fundamental-group-mult*  $T \ y$ )  
 (*fundamental-group-one*  $T \ y$ )  
 (*fundamental-group-inv*  $T \ y$ )

(*fundamental-group-map*  $S\ x\ T\ y\ h$ )  
 <proof>

**end**

**theory** *Explicit-Fundamental-Group-Scaffold*

**imports** *Explicit-Path-Homotopy-Scaffold* *Fundamental-Group-Scaffold*

**begin**

## 12 Explicit-topology fundamental-group quotients

Building on the explicit-topology path layer, this theory forms loop classes, quotient carriers, and induced maps for arbitrary topological spaces. It is the main bridge from point-set topology to the carrier-group algebra used in the Seifert–van Kampen interface.

**definition** *loopin-class* ::

$'a\ topology \Rightarrow 'a \Rightarrow (real \Rightarrow 'a) \Rightarrow (real \Rightarrow 'a)\ set$

**where**

$loopin-class\ X\ x\ p = \{q. q \in loopin-space\ X\ x \wedge homotopic-pathsin\ X\ q\ p\}$

**definition** *fundamental-groupin-space* ::

$'a\ topology \Rightarrow 'a \Rightarrow ((real \Rightarrow 'a)\ set)\ set$

**where**

$fundamental-groupin-space\ X\ x = loopin-class\ X\ x\ 'loopin-space\ X\ x$

**definition** *some-loopin* ::

$'a\ topology \Rightarrow 'a \Rightarrow (real \Rightarrow 'a)\ set \Rightarrow (real \Rightarrow 'a)$

**where**

$some-loopin\ X\ x\ Q = (SOME\ p. p \in loopin-space\ X\ x \wedge Q = loopin-class\ X\ x\ p)$

**definition** *fundamental-groupin-one* ::

$'a\ topology \Rightarrow 'a \Rightarrow (real \Rightarrow 'a)\ set$

**where**

$fundamental-groupin-one\ X\ x = loopin-class\ X\ x\ (\lambda-. x)$

**definition** *fundamental-groupin-mult* ::

$'a\ topology \Rightarrow 'a \Rightarrow$

$(real \Rightarrow 'a)\ set \Rightarrow (real \Rightarrow 'a)\ set \Rightarrow (real \Rightarrow 'a)\ set$

**where**

$fundamental-groupin-mult\ X\ x\ A\ B =$

$loopin-class\ X\ x\ (joinpathin\ (some-loopin\ X\ x\ A)\ (some-loopin\ X\ x\ B))$

**definition** *fundamental-groupin-inv* ::

$'a\ topology \Rightarrow 'a \Rightarrow (real \Rightarrow 'a)\ set \Rightarrow (real \Rightarrow 'a)\ set$

**where**

$fundamental-groupin-inv\ X\ x\ A = loopin-class\ X\ x\ (reversepathin\ (some-loopin\ X\ x\ A))$

**definition** *loopin-image* ::

$('a \Rightarrow 'b) \Rightarrow (\text{real} \Rightarrow 'a) \Rightarrow (\text{real} \Rightarrow 'b)$

**where**

$\text{loopin-image } h \ p = h \circ p$

**definition** *fundamental-groupin-map* ::

$'a \ \text{topology} \Rightarrow 'a \Rightarrow 'b \ \text{topology} \Rightarrow 'b \Rightarrow$

$('a \Rightarrow 'b) \Rightarrow (\text{real} \Rightarrow 'a) \ \text{set} \Rightarrow (\text{real} \Rightarrow 'b) \ \text{set}$

**where**

$\text{fundamental-groupin-map } X \ x \ Y \ y \ h \ A =$

$\text{loopin-class } Y \ y \ (\text{loopin-image } h \ (\text{some-loopin } X \ x \ A))$

**lemma** *loopin-class-in-space*:

**assumes**  $p \in \text{loopin-space } X \ x$

**shows**  $\text{loopin-class } X \ x \ p \in \text{fundamental-groupin-space } X \ x$

$\langle \text{proof} \rangle$

**lemma** *fundamental-groupin-spaceE*:

**assumes**  $Q \in \text{fundamental-groupin-space } X \ x$

**obtains**  $p$  **where**  $p \in \text{loopin-space } X \ x \ Q = \text{loopin-class } X \ x \ p$

$\langle \text{proof} \rangle$

**lemma** *some-loopin-spec*:

**assumes**  $Q \in \text{fundamental-groupin-space } X \ x$

**shows**  $\text{some-loopin } X \ x \ Q \in \text{loopin-space } X \ x$

**and**  $Q = \text{loopin-class } X \ x \ (\text{some-loopin } X \ x \ Q)$

$\langle \text{proof} \rangle$

**lemma** *loopin-class-eqI*:

**assumes**  $p: p \in \text{loopin-space } X \ x$

**and**  $q: q \in \text{loopin-space } X \ x$

**and**  $pq: \text{homotopic-paths in } X \ p \ q$

**shows**  $\text{loopin-class } X \ x \ p = \text{loopin-class } X \ x \ q$

$\langle \text{proof} \rangle$

**lemma** *loopin-class-eq-iff*:

**assumes**  $p: p \in \text{loopin-space } X \ x$

**and**  $q: q \in \text{loopin-space } X \ x$

**shows**  $\text{loopin-class } X \ x \ p = \text{loopin-class } X \ x \ q \iff \text{homotopic-paths in } X \ p \ q$

$\langle \text{proof} \rangle$

**lemma** *fundamental-groupin-one-in-space*:

**assumes**  $x \in \text{topspace } X$

**shows**  $\text{fundamental-groupin-one } X \ x \in \text{fundamental-groupin-space } X \ x$

$\langle \text{proof} \rangle$

**lemma** *fundamental-groupin-mult-eqI*:

**assumes**  $A\text{-in}: A \in \text{fundamental-groupin-space } X \ x$

**and**  $B$ -in:  $B \in \text{fundamental-groupin-space } X \ x$   
**and**  $p$ :  $p \in \text{loopin-space } X \ x$   
**and**  $q$ :  $q \in \text{loopin-space } X \ x$   
**and**  $A$ :  $A = \text{loopin-class } X \ x \ p$   
**and**  $B$ :  $B = \text{loopin-class } X \ x \ q$   
**shows**  $\text{fundamental-groupin-mult } X \ x \ A \ B = \text{loopin-class } X \ x \ (\text{joinpathin } p \ q)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fundamental-groupin-mult-in-space}$ :  
**assumes**  $A \in \text{fundamental-groupin-space } X \ x$   
**and**  $B \in \text{fundamental-groupin-space } X \ x$   
**shows**  $\text{fundamental-groupin-mult } X \ x \ A \ B \in \text{fundamental-groupin-space } X \ x$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fundamental-groupin-inv-eqI}$ :  
**assumes**  $A$ -in:  $A \in \text{fundamental-groupin-space } X \ x$   
**and**  $p$ :  $p \in \text{loopin-space } X \ x$   
**and**  $A$ :  $A = \text{loopin-class } X \ x \ p$   
**shows**  $\text{fundamental-groupin-inv } X \ x \ A = \text{loopin-class } X \ x \ (\text{reversepathin } p)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fundamental-groupin-inv-in-space}$ :  
**assumes**  $A \in \text{fundamental-groupin-space } X \ x$   
**shows**  $\text{fundamental-groupin-inv } X \ x \ A \in \text{fundamental-groupin-space } X \ x$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{loopin-image-in-space}$ :  
**assumes**  $p$ :  $p \in \text{loopin-space } X \ x$   
**and**  $h$ :  $\text{continuous-map } X \ Y \ f$   
**and**  $fx$ :  $f \ x = y$   
**shows**  $\text{loopin-image } f \ p \in \text{loopin-space } Y \ y$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fundamental-groupin-map-rep}$ :  
**assumes**  $A$ :  $A \in \text{fundamental-groupin-space } X \ x$   
**and**  $p$ :  $p \in \text{loopin-space } X \ x$   
**and**  $rep$ :  $A = \text{loopin-class } X \ x \ p$   
**and**  $h$ :  $\text{continuous-map } X \ Y \ f$   
**and**  $fx$ :  $f \ x = y$   
**shows**  $\text{fundamental-groupin-map } X \ x \ Y \ y \ f \ A = \text{loopin-class } Y \ y \ (\text{loopin-image } f \ p)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fundamental-groupin-map-in-space}$ :  
**assumes**  $A$ :  $A \in \text{fundamental-groupin-space } X \ x$   
**and**  $h$ :  $\text{continuous-map } X \ Y \ f$   
**and**  $fx$ :  $f \ x = y$   
**shows**  $\text{fundamental-groupin-map } X \ x \ Y \ y \ f \ A \in \text{fundamental-groupin-space } Y \ y$   
 $\langle \text{proof} \rangle$

**lemma** *loopin-image-joinpathin* [simp]:  
 $loopin-image\ h\ (joinpathin\ p\ q) = joinpathin\ (loopin-image\ h\ p)\ (loopin-image\ h\ q)$   
 ⟨proof⟩

**lemma** *loopin-image-reversepathin* [simp]:  
 $loopin-image\ h\ (reversepathin\ p) = reversepathin\ (loopin-image\ h\ p)$   
 ⟨proof⟩

**lemma** *fundamental-groupin-map-mult*:  
**assumes** *A-in*:  $A \in fundamental-groupin-space\ X\ x$   
**and** *B-in*:  $B \in fundamental-groupin-space\ X\ x$   
**and** *h*: *continuous-map*  $X\ Y\ f$   
**and** *fx*:  $f\ x = y$   
**shows**  $fundamental-groupin-map\ X\ x\ Y\ y\ f\ (fundamental-groupin-mult\ X\ x\ A\ B)$   
 =  
 $fundamental-groupin-mult\ Y\ y$   
 $(fundamental-groupin-map\ X\ x\ Y\ y\ f\ A)$   
 $(fundamental-groupin-map\ X\ x\ Y\ y\ f\ B)$   
 ⟨proof⟩

**lemma** *fundamental-groupin-mult-one-left*:  
**assumes** *x-in*:  $x \in topspace\ X$   
**and** *A-in*:  $A \in fundamental-groupin-space\ X\ x$   
**shows**  $fundamental-groupin-mult\ X\ x\ (fundamental-groupin-one\ X\ x)\ A = A$   
 ⟨proof⟩

**lemma** *fundamental-groupin-mult-one-right*:  
**assumes** *x-in*:  $x \in topspace\ X$   
**and** *A-in*:  $A \in fundamental-groupin-space\ X\ x$   
**shows**  $fundamental-groupin-mult\ X\ x\ A\ (fundamental-groupin-one\ X\ x) = A$   
 ⟨proof⟩

**lemma** *fundamental-groupin-mult-assoc*:  
**assumes** *A-in*:  $A \in fundamental-groupin-space\ X\ x$   
**and** *B-in*:  $B \in fundamental-groupin-space\ X\ x$   
**and** *C-in*:  $C \in fundamental-groupin-space\ X\ x$   
**shows**  $fundamental-groupin-mult\ X\ x\ A\ (fundamental-groupin-mult\ X\ x\ B\ C) =$   
 $fundamental-groupin-mult\ X\ x\ (fundamental-groupin-mult\ X\ x\ A\ B)\ C$   
 ⟨proof⟩

**lemma** *fundamental-groupin-mult-inv-right*:  
**assumes** *x-in*:  $x \in topspace\ X$   
**and** *A-in*:  $A \in fundamental-groupin-space\ X\ x$   
**shows**  $fundamental-groupin-mult\ X\ x\ A\ (fundamental-groupin-inv\ X\ x\ A) = fundamental-groupin-one\ X\ x$   
 ⟨proof⟩

**lemma** *fundamental-groupin-mult-inv-left*:  
**assumes** *x-in*:  $x \in \text{topspace } X$   
**and** *A-in*:  $A \in \text{fundamental-groupin-space } X x$   
**shows**  $\text{fundamental-groupin-mult } X x (\text{fundamental-groupin-inv } X x A) A =$   
 $\text{fundamental-groupin-one } X x$   
 $\langle \text{proof} \rangle$

**lemma** *fundamental-groupin-carrier-group*:  
**assumes** *x-in*:  $x \in \text{topspace } X$   
**shows** *carrier-group*  
 $(\text{fundamental-groupin-space } X x)$   
 $(\text{fundamental-groupin-mult } X x)$   
 $(\text{fundamental-groupin-one } X x)$   
 $(\text{fundamental-groupin-inv } X x)$   
 $\langle \text{proof} \rangle$

**lemma** *fundamental-groupin-map-carrier-group-hom*:  
**assumes** *x-in*:  $x \in \text{topspace } X$   
**and** *h*: *continuous-map*  $X Y f$   
**and** *fx*:  $f x = y$   
**shows** *carrier-group-hom*  
 $(\text{fundamental-groupin-space } X x)$   
 $(\text{fundamental-groupin-mult } X x)$   
 $(\text{fundamental-groupin-one } X x)$   
 $(\text{fundamental-groupin-inv } X x)$   
 $(\text{fundamental-groupin-space } Y y)$   
 $(\text{fundamental-groupin-mult } Y y)$   
 $(\text{fundamental-groupin-one } Y y)$   
 $(\text{fundamental-groupin-inv } Y y)$   
 $(\text{fundamental-groupin-map } X x Y y f)$   
 $\langle \text{proof} \rangle$

**lemma** *loopin-space-top-of-set [simp]*:  
 $\text{loopin-space } (\text{top-of-set } S) x = \text{loop-space } S x$   
 $\langle \text{proof} \rangle$

**lemma** *loopin-class-top-of-set [simp]*:  
 $\text{loopin-class } (\text{top-of-set } S) x p = \text{loop-class } S x p$   
 $\langle \text{proof} \rangle$

**lemma** *fundamental-groupin-space-top-of-set [simp]*:  
 $\text{fundamental-groupin-space } (\text{top-of-set } S) x = \text{fundamental-group-space } S x$   
 $\langle \text{proof} \rangle$

**lemma** *fundamental-groupin-one-top-of-set [simp]*:  
 $\text{fundamental-groupin-one } (\text{top-of-set } S) x = \text{fundamental-group-one } S x$   
 $\langle \text{proof} \rangle$

**lemma** *loopin-image-eq-loop-image [simp]*:

*loopin-image h p = loop-image h p*  
 ⟨*proof*⟩

**end**  
**theory** *Pushout-Scaffold*  
**imports** *Equivalence-Quotients*  
**begin**

### 13 Pushout-style quotients of disjoint sums

The pushout of two maps is first presented here as a quotient of the disjoint sum by the obvious glue relation. This algebraic infrastructure is independent of any topology; the topological quotient is added later, once the point-set infrastructure is in place.

**inductive** *pushout-rel* ::  
 (*'c* => *'a*) => (*'c* => *'b*) => (*'a* + *'b*) => (*'a* + *'b*) => *bool*  
**for** *f* :: *'c* => *'a* **and** *g* :: *'c* => *'b*  
**where**  
*refl* [*intro!*, *simp*]: *pushout-rel f g x x*  
| *sym*: *pushout-rel f g x y ==> pushout-rel f g y x*  
| *trans*: *pushout-rel f g x y ==> pushout-rel f g y z ==> pushout-rel f g x z*  
| *glue* [*intro*]: *pushout-rel f g (Inl (f c)) (Inr (g c))*

**interpretation** *pushout-equiv*: *equivalence-relation pushout-rel f g*  
 ⟨*proof*⟩

**definition** *pushout-class* ::  
 (*'c* => *'a*) => (*'c* => *'b*) => (*'a* + *'b*) => (*'a* + *'b*) *set*  
**where**  
*pushout-class f g x = equiv-class (pushout-rel f g) x*

**definition** *pushout-space* ::  
 (*'c* => *'a*) => (*'c* => *'b*) => (*'a* + *'b*) *set set*  
**where**  
*pushout-space f g = quotient-space (pushout-rel f g)*

**definition** *pushout-inl* ::  
 (*'c* => *'a*) => (*'c* => *'b*) => *'a* => (*'a* + *'b*) *set*  
**where**  
*pushout-inl f g a = pushout-class f g (Inl a)*

**definition** *pushout-inr* ::  
 (*'c* => *'a*) => (*'c* => *'b*) => *'b* => (*'a* + *'b*) *set*  
**where**  
*pushout-inr f g b = pushout-class f g (Inr b)*

**lemma** *pushout-class-eq-iff*:  
*pushout-class f g x = pushout-class f g y  $\longleftrightarrow$  pushout-rel f g x y*

*<proof>*

**lemma** *pushout-inl-in-space* [*intro*]:  
*pushout-inl f g a ∈ pushout-space f g*  
*<proof>*

**lemma** *pushout-inr-in-space* [*intro*]:  
*pushout-inr f g b ∈ pushout-space f g*  
*<proof>*

**lemma** *pushout-glue*:  
*pushout-inl f g (f c) = pushout-inr f g (g c)*  
*<proof>*

**definition** *pushout-case-compatible* ::  
*('c => 'a) => ('c => 'b) => ('a => 'd) => ('b => 'd) => bool*  
**where**  
*pushout-case-compatible f g left right ⟷ (∀ c. left (f c) = right (g c))*

**lemma** *pushout-rel-case-cong*:  
**assumes** *compat: pushout-case-compatible f g left right*  
**and** *rel: pushout-rel f g x y*  
**shows** *case-sum left right x = case-sum left right y*  
*<proof>*

**definition** *pushout-rec* ::  
*('c => 'a) => ('c => 'b) => ('a => 'd) => ('b => 'd) => ('a + 'b) set =>*  
*'d*  
**where**  
*pushout-rec f g left right X =*  
*(THE z. ∃ x. X = pushout-class f g x ∧ z = case-sum left right x)*

**lemma** *pushout-rec-well-defined*:  
**assumes** *compat: pushout-case-compatible f g left right*  
**and** *X-in: X ∈ pushout-space f g*  
**shows** *∃! z. ∃ x. X = pushout-class f g x ∧ z = case-sum left right x*  
*<proof>*

**lemma** *pushout-rec-inl*:  
**assumes** *compat: pushout-case-compatible f g left right*  
**shows** *pushout-rec f g left right (pushout-inl f g a) = left a*  
*<proof>*

**lemma** *pushout-rec-inr*:  
**assumes** *compat: pushout-case-compatible f g left right*  
**shows** *pushout-rec f g left right (pushout-inr f g b) = right b*  
*<proof>*

**end**

```

theory Seifert-Van-Kampen-Scaffold
  imports Pushout-Scaffold Carrier-Amalgamated-Free-Product
begin

```

## 14 Conditional Seifert–van Kampen interface

This theory packages a general quotient-level Seifert–van Kampen argument behind explicit encode/decode obligations. Once a decoding map factors through the full amalgamation quotient, and once the usual round-trip laws are available, the quotient-level bijection follows abstractly.

```

locale svk-encode-decode-interface =
  fixes i1 :: 'h => 'a::group-add
    and i2 :: 'h => 'b::group-add
    and encode :: 'pi1-pushout =>
      ('a, 'b) free-product-word
    and decode :: ('a, 'b) free-product-word => 'pi1-pushout
  assumes decode-respects:
    full-amalg-equiv i1 i2 u v ==> decode u = decode v
  and decode-encode:
    decode (encode x) = x
  and encode-decode:
    full-amalg-equiv i1 i2 (encode (decode w)) w
begin

```

```

definition svk-quotient-map ::
  'pi1-pushout => (('a, 'b) free-product-word) set
where
  svk-quotient-map x = full-amalg-class i1 i2 (encode x)

```

```

lemma svk-quotient-map-in-space:
  svk-quotient-map x ∈ full-amalgamated-free-product-space i1 i2
  ⟨proof⟩

```

```

lemma svk-quotient-map-injective:
  assumes svk-quotient-map x = svk-quotient-map y
  shows x = y
  ⟨proof⟩

```

```

lemma svk-quotient-map-surjective:
  assumes Q ∈ full-amalgamated-free-product-space i1 i2
  shows  $\exists x. svk-quotient-map x = Q$ 
  ⟨proof⟩

```

```

theorem seifert-van-kampen-bij-betw:
  bij-betw svk-quotient-map UNIV (full-amalgamated-free-product-space i1 i2)
  ⟨proof⟩

```

```

end

```

```

locale carrier-svk-encode-decode-interface =
  fixes  $G1 :: 'a \text{ set}$ 
    and  $G2 :: 'b \text{ set}$ 
    and  $H :: 'h \text{ set}$ 
    and  $i1 :: 'h \Rightarrow 'a$ 
    and  $i2 :: 'h \Rightarrow 'b$ 
    and  $mult1 :: 'a \Rightarrow 'a \Rightarrow 'a$ 
    and  $one1 :: 'a$ 
    and  $mult2 :: 'b \Rightarrow 'b \Rightarrow 'b$ 
    and  $one2 :: 'b$ 
    and  $encode :: 'pi1\text{-pushout} \Rightarrow ('a, 'b) \text{ free-product-word}$ 
    and  $decode :: ('a, 'b) \text{ free-product-word} \Rightarrow 'pi1\text{-pushout}$ 
  assumes encode-in-space:
    fpw-in-space  $G1 G2 (encode\ x)$ 
  and decode-respects:
    carrier-full-amalg-equiv  $G1 G2 H i1 i2 mult1 one1 mult2 one2 u v \implies decode\ u = decode\ v$ 
  and decode-encode:
     $decode\ (encode\ x) = x$ 
  and encode-decode:
    carrier-full-amalg-equiv  $G1 G2 H i1 i2 mult1 one1 mult2 one2 (encode\ (decode\ w))\ w$ 
begin

definition carrier-svk-quotient-map ::
   $'pi1\text{-pushout} \Rightarrow (('a, 'b) \text{ free-product-word}) \text{ set}$ 
where
  carrier-svk-quotient-map  $x =$ 
    carrier-full-amalg-class  $G1 G2 H i1 i2 mult1 one1 mult2 one2 (encode\ x)$ 

lemma carrier-svk-quotient-map-in-space:
  carrier-svk-quotient-map  $x \in$ 
    carrier-full-amalgamated-free-product-space  $G1 G2 H i1 i2 mult1 one1 mult2$ 
   $one2$ 
   $\langle proof \rangle$ 

lemma carrier-svk-quotient-map-injective:
  assumes carrier-svk-quotient-map  $x = \text{carrier-svk-quotient-map } y$ 
  shows  $x = y$ 
   $\langle proof \rangle$ 

lemma carrier-svk-quotient-map-surjective:
  assumes
     $Q \in \text{carrier-full-amalgamated-free-product-space } G1 G2 H i1 i2 mult1 one1$ 
   $mult2 one2$ 
  shows  $\exists x. \text{carrier-svk-quotient-map } x = Q$ 
   $\langle proof \rangle$ 

```

```

theorem carrier-seifert-van-kampen-bij-betw:
  bij-betw carrier-svk-quotient-map UNIV
  (carrier-full-amalgamated-free-product-space G1 G2 H i1 i2 mult1 one1 mult2
one2)
  ⟨proof⟩

```

**end**

**end**

**theory** *Classical-Seifert-Van-Kampen*

**imports**

*Carrier-Amalgamated-Free-Product-Eval*

*Explicit-Fundamental-Group-Scaffold*

*Seifert-Van-Kampen-Scaffold*

**begin**

## 15 Classical Seifert–van Kampen for open unions

This theory specializes the general encode/decode interface to the classical open-cover hypotheses. Its long middle portion constructs encodings of loops by subdividing them into pieces that lie in  $U$  or  $V$ , proves invariance under refinement and homotopy, and then packages the resulting quotient as the carrier-based amalgamated free product of the three relevant fundamental groups.

**lemma** *path-top-of-setI*:

**assumes** *path p*

**and** *path-image p  $\subseteq$  S*

**shows** *pathin (top-of-set S) p*

⟨*proof*⟩

**locale** *classical-svk-setup =*

**fixes** *U :: 'a::topological-space set*

**and** *V :: 'a set*

**and** *x0 :: 'a*

**assumes** *U-open: open U*

**and** *V-open: open V*

**and** *x0-in: x0  $\in$  U  $\cap$  V*

**and** *UV-path-connected: path-connected (U  $\cap$  V)*

**begin**

**abbreviation** *W where W  $\equiv$  U  $\cup$  V*

**abbreviation** *G1 where G1  $\equiv$  fundamental-group-space U x0*

**abbreviation** *G2 where G2  $\equiv$  fundamental-group-space V x0*

**abbreviation** *H where H  $\equiv$  fundamental-group-space (U  $\cap$  V) x0*

**abbreviation** *mult1 where mult1  $\equiv$  fundamental-group-mult U x0*

**abbreviation** *one1 where one1  $\equiv$  fundamental-group-one U x0*

**abbreviation** *mult2* **where** *mult2*  $\equiv$  *fundamental-group-mult* *V* *x0*  
**abbreviation** *one2* **where** *one2*  $\equiv$  *fundamental-group-one* *V* *x0*  
**abbreviation** *multW* **where** *multW*  $\equiv$  *fundamental-group-mult* *W* *x0*  
**abbreviation** *oneW* **where** *oneW*  $\equiv$  *fundamental-group-one* *W* *x0*

**abbreviation** *i1* **where** *i1*  $\equiv$  *fundamental-group-map* (*U*  $\cap$  *V*) *x0* *U* *x0* *id*  
**abbreviation** *i2* **where** *i2*  $\equiv$  *fundamental-group-map* (*U*  $\cap$  *V*) *x0* *V* *x0* *id*  
**abbreviation** *j1* **where** *j1*  $\equiv$  *fundamental-group-map* *U* *x0* *W* *x0* *id*  
**abbreviation** *j2* **where** *j2*  $\equiv$  *fundamental-group-map* *V* *x0* *W* *x0* *id*

## 15.1 Carrier-side setup

The first part of the theory fixes the inclusion-induced maps between the three fundamental groups and packages them into the carrier-side evaluation locale used by the later decode map. This isolates the algebraic compatibility conditions that are immediate from the open-union inclusions.

**lemma** *x0-in-U* [*simp*]: *x0*  $\in$  *U*  
 $\langle$ *proof* $\rangle$

**lemma** *x0-in-V* [*simp*]: *x0*  $\in$  *V*  
 $\langle$ *proof* $\rangle$

**lemma** *x0-in-W* [*simp*]: *x0*  $\in$  *W*  
 $\langle$ *proof* $\rangle$

**lemma** *x0-in-UV* [*simp*]: *x0*  $\in$  *U*  $\cap$  *V*  
 $\langle$ *proof* $\rangle$

**lemma** *i1-in-G1*:  
**assumes** *h*  $\in$  *H*  
**shows** *i1* *h*  $\in$  *G1*  
 $\langle$ *proof* $\rangle$

**lemma** *i2-in-G2*:  
**assumes** *h*  $\in$  *H*  
**shows** *i2* *h*  $\in$  *G2*  
 $\langle$ *proof* $\rangle$

**lemma** *union-fundamental-group-maps-agree*:  
**assumes** *h-in*: *h*  $\in$  *H*  
**shows** *j1* (*i1* *h*) = *j2* (*i2* *h*)  
 $\langle$ *proof* $\rangle$

**lemma** *decode-locale*:  
*carrier-full-amalg-word-eval*  
*G1* *mult1* *one1* (*fundamental-group-inv* *U* *x0*)  
*G2* *mult2* *one2* (*fundamental-group-inv* *V* *x0*)  
*H* *i1* *i2*  
(*fundamental-group-space* *W* *x0*) *multW* *oneW* (*fundamental-group-inv* *W* *x0*)

$j1\ j2$   
(proof)

**interpretation** *decode*:

*carrier-full-amalg-word-eval*  
 $G1\ mult1\ one1\ fundamental\ group\ inv\ U\ x0$   
 $G2\ mult2\ one2\ fundamental\ group\ inv\ V\ x0$   
 $H\ i1\ i2$   
*fundamental-group-space*  $W\ x0\ multW\ oneW\ fundamental\ group\ inv\ W\ x0$   
 $j1\ j2$   
(proof)

**abbreviation** *svk-word-eval* **where**  $svk\ word\ eval \equiv decode.\ carrier\ full\ amalg\ eval$   
**abbreviation** *svk-decode* **where**  $svk\ decode \equiv decode.\ carrier\ full\ amalg\ decode$

**lemma** *svk-decode-in-space*:

*svk-decode*  $w \in fundamental\ group\ space\ W\ x0$   
(proof)

**lemma** *svk-decode-respects*:

**assumes** *carrier-full-amalg-equiv*  $G1\ G2\ H\ i1\ i2\ mult1\ one1\ mult2\ one2\ u\ v$   
**shows**  $svk\ decode\ u = svk\ decode\ v$   
(proof)

**lemma** *svk-decode-eq-eval*:

**assumes** *fpw-in-space*  $G1\ G2\ w$   
**shows**  $svk\ decode\ w = svk\ word\ eval\ w$   
(proof)

**lemma** *carrier-full-amalg-equiv-left-context*:

**assumes** *rel: carrier-full-amalg-equiv*  $G1\ G2\ H\ i1\ i2\ mult1\ one1\ mult2\ one2\ u\ v$   
**and** *a-in*:  $a \in G1$   
**shows** *carrier-full-amalg-equiv*  $G1\ G2\ H\ i1\ i2\ mult1\ one1\ mult2\ one2$   
 $(WordLeft\ a\ u)\ (WordLeft\ a\ v)$   
(proof)

**lemma** *carrier-full-amalg-equiv-right-context*:

**assumes** *rel: carrier-full-amalg-equiv*  $G1\ G2\ H\ i1\ i2\ mult1\ one1\ mult2\ one2\ u\ v$   
**and** *b-in*:  $b \in G2$   
**shows** *carrier-full-amalg-equiv*  $G1\ G2\ H\ i1\ i2\ mult1\ one1\ mult2\ one2$   
 $(WordRight\ b\ u)\ (WordRight\ b\ v)$   
(proof)

**lemma** *carrier-full-amalg-equiv-left-pair-eq*:

**assumes** *a-in*:  $a \in G1$   
**and** *b-in*:  $b \in G1$   
**and** *ab-in*:  $mult1\ a\ b \in G1$   
**and** *c-in*:  $c \in G1$   
**and** *d-in*:  $d \in G1$

**and** *cd-in*:  $\text{mult1 } c \ d \in G1$   
**and** *rest-in*:  $\text{fpw-in-space } G1 \ G2 \ \text{rest}$   
**and** *eq*:  $\text{mult1 } a \ b = \text{mult1 } c \ d$   
**shows** *carrier-full-amalg-equiv*  $G1 \ G2 \ H \ i1 \ i2 \ \text{mult1 } \text{one1} \ \text{mult2 } \text{one2}$   
 $(\text{WordLeft } a \ (\text{WordLeft } b \ \text{rest}))$   
 $(\text{WordLeft } c \ (\text{WordLeft } d \ \text{rest}))$   
 $\langle \text{proof} \rangle$

**lemma** *carrier-full-amalg-equiv-right-pair-eq*:

**assumes** *a-in*:  $a \in G2$   
**and** *b-in*:  $b \in G2$   
**and** *ab-in*:  $\text{mult2 } a \ b \in G2$   
**and** *c-in*:  $c \in G2$   
**and** *d-in*:  $d \in G2$   
**and** *cd-in*:  $\text{mult2 } c \ d \in G2$   
**and** *rest-in*:  $\text{fpw-in-space } G1 \ G2 \ \text{rest}$   
**and** *eq*:  $\text{mult2 } a \ b = \text{mult2 } c \ d$   
**shows** *carrier-full-amalg-equiv*  $G1 \ G2 \ H \ i1 \ i2 \ \text{mult1 } \text{one1} \ \text{mult2 } \text{one2}$   
 $(\text{WordRight } a \ (\text{WordRight } b \ \text{rest}))$   
 $(\text{WordRight } c \ (\text{WordRight } d \ \text{rest}))$   
 $\langle \text{proof} \rangle$

**lemma** *loop-subdivision-by-cover*:

**assumes** *p-loop*:  $p \in \text{loop-space } W \ x0$   
**shows**  $\exists n::\text{nat. } 0 < n \wedge$   
 $(\forall i < n.$   
 $\text{subpathin } (\text{real } i / \text{real } n) \ (\text{real } (\text{Suc } i) / \text{real } n) \ p \ \{0..1\} \subseteq U \vee$   
 $\text{subpathin } (\text{real } i / \text{real } n) \ (\text{real } (\text{Suc } i) / \text{real } n) \ p \ \{0..1\} \subseteq V)$   
 $\langle \text{proof} \rangle$

**definition** *connector*  $:: 'a \Rightarrow \text{real} \Rightarrow 'a$  **where**

$\text{connector } a =$   
 $(\text{if } a = x0 \ \text{then } (\lambda-. \ x0)$   
 $\ \text{else } (\text{SOME } p. \ \text{path } p \wedge \text{path-image } p \subseteq U \cap V \wedge \text{pathstart } p = x0 \wedge \text{pathfinish } p = a))$

**lemma** *connector-x0* [*simp*]:

$\text{connector } x0 = (\lambda-. \ x0)$   
 $\langle \text{proof} \rangle$

**lemma** *connector-witness*:

**assumes** *a-in*:  $a \in U \cap V$   
**shows**  $\exists p. \ \text{path } p \wedge \text{path-image } p \subseteq U \cap V \wedge \text{pathstart } p = x0 \wedge \text{pathfinish } p = a$   
 $\langle \text{proof} \rangle$

**lemma** *connector-path*:

**assumes** *a-in*:  $a \in U \cap V$   
**shows**  $\text{path } (\text{connector } a)$

*<proof>*

**lemma** *connector-image-subset*:

**assumes** *a-in*:  $a \in U \cap V$

**shows** *path-image* (connector *a*)  $\subseteq U \cap V$

*<proof>*

**lemma** *connector-start*:

**assumes** *a-in*:  $a \in U \cap V$

**shows** *pathstart* (connector *a*) =  $x0$

*<proof>*

**lemma** *connector-finish*:

**assumes** *a-in*:  $a \in U \cap V$

**shows** *pathfinish* (connector *a*) = *a*

*<proof>*

**lemmas** *connector-spec* = *connector-path* *connector-image-subset* *connector-start*  
*connector-finish*

**definition** *segment-loop* ::  $(real \Rightarrow 'a) \Rightarrow real \Rightarrow real \Rightarrow real \Rightarrow 'a$  **where**

*segment-loop* *p u v* =

(connector (*p u*) +++ *subpathin* *u v p*) +++ *reversepath* (connector (*p v*))

**lemma** *segment-loop-in-set*:

**assumes** *p-path*: *path* *p*

**and** *p-image*: *path-image* *p*  $\subseteq W$

**and** *uv01*:  $u \in \{0..1\}$   $v \in \{0..1\}$

**and** *puv-in*:  $p\ u \in U \cap V$   $p\ v \in U \cap V$

**and** *conn-u*: *path-image* (connector (*p u*))  $\subseteq S$

**and** *conn-v*: *path-image* (connector (*p v*))  $\subseteq S$

**and** *seg-in*: *subpathin* *u v p* ' $\{0..1\}$ '  $\subseteq S$

**and** *x0S*:  $x0 \in S$

**shows** *segment-loop* *p u v*  $\in$  *loop-space* *S*  $x0$

*<proof>*

**lemma** *segment-loop-in-U*:

**assumes** *p-path*: *path* *p*

**and** *p-image*: *path-image* *p*  $\subseteq W$

**and** *uv01*:  $u \in \{0..1\}$   $v \in \{0..1\}$

**and** *puv-in*:  $p\ u \in U \cap V$   $p\ v \in U \cap V$

**and** *seg-in*: *subpathin* *u v p* ' $\{0..1\}$ '  $\subseteq U$

**shows** *segment-loop* *p u v*  $\in$  *loop-space* *U*  $x0$

*<proof>*

**lemma** *segment-loop-in-V*:

**assumes** *p-path*: *path* *p*

**and** *p-image*: *path-image* *p*  $\subseteq W$

**and** *uv01*:  $u \in \{0..1\}$   $v \in \{0..1\}$

**and** *puv-in*:  $p \ u \in U \cap V \ p \ v \in U \cap V$   
**and** *seg-in*:  $\text{subpathin } u \ v \ p \ ' \{0..1\} \subseteq V$   
**shows** *segment-loop*  $p \ u \ v \in \text{loop-space } V \ x0$   
 ⟨*proof*⟩

**lemma** *segment-loop-in-W*:  
**assumes** *p-path*:  $\text{path } p$   
**and** *p-image*:  $\text{path-image } p \subseteq W$   
**and** *uv01*:  $u \in \{0..1\} \ v \in \{0..1\}$   
**and** *puv-in*:  $p \ u \in U \cap V \ p \ v \in U \cap V$   
**and** *seg-in*:  $\text{subpathin } u \ v \ p \ ' \{0..1\} \subseteq W$   
**shows** *segment-loop*  $p \ u \ v \in \text{loop-space } W \ x0$   
 ⟨*proof*⟩

**lemma** *path-subpathin*:  
**assumes** *path*  $p$   
**and**  $u \in \{0..1\}$   
**and**  $v \in \{0..1\}$   
**shows**  $\text{path } (\text{subpathin } u \ v \ p)$   
 ⟨*proof*⟩

**lemma** *path-image-subpathin-subset*:  
**assumes**  $u \in \{0..1\}$   
**and**  $v \in \{0..1\}$   
**shows**  $\text{path-image } (\text{subpathin } u \ v \ p) \subseteq \text{path-image } p$   
 ⟨*proof*⟩

**lemma** *reversepath-subpathin* [*simp*]:  
 $\text{reversepath } (\text{subpathin } u \ v \ p) = \text{subpathin } v \ u \ p$   
 ⟨*proof*⟩

**lemma** *subpathin-refl* [*simp*]:  
 $\text{subpathin } u \ u \ p = (\lambda-. \ p \ u)$   
 ⟨*proof*⟩

**fun** *svk-partition* ::  $(\text{real} \Rightarrow 'a) \Rightarrow \text{real list} \Rightarrow \text{bool list} \Rightarrow \text{bool}$  **where**  
 $\text{svk-partition } p \ [] \ bs = \text{False}$   
 $|\ \text{svk-partition } p \ [t] \ [] = (t = 1 \wedge p \ t \in U \cap V)$   
 $|\ \text{svk-partition } p \ [t] \ (b \ \# \ bs) = \text{False}$   
 $|\ \text{svk-partition } p \ (t \ \# \ u \ \# \ ts) \ [] = \text{False}$   
 $|\ \text{svk-partition } p \ (t \ \# \ u \ \# \ ts) \ (b \ \# \ bs) =$   
 $\quad (t \in \{0..1\} \wedge p \ t \in U \cap V \wedge u \in \{0..1\} \wedge t < u \wedge$   
 $\quad (\text{if } b \ \text{then } \text{subpathin } t \ u \ p \ ' \{0..1\} \subseteq U \ \text{else } \text{subpathin } t \ u \ p \ ' \{0..1\} \subseteq V) \wedge$   
 $\quad \text{svk-partition } p \ (u \ \# \ ts) \ bs)$

## 15.2 Partitions and encoded loop words

A loop is encoded by subdividing the unit interval into pieces whose images lie entirely in  $U$  or entirely in  $V$ . The resulting bitstring records which side

each segment uses, and the partition word records the corresponding loop classes in the two factors of the amalgamated free product.

**definition** *valid-partition* :: (real  $\Rightarrow$  'a)  $\Rightarrow$  real list  $\Rightarrow$  bool list  $\Rightarrow$  bool **where**  
*valid-partition* p ts bs  $\longleftrightarrow$  ts  $\neq$  []  $\wedge$  hd ts = 0  $\wedge$  svk-partition p ts bs

**fun** *cover-partition* :: (real  $\Rightarrow$  'a)  $\Rightarrow$  real list  $\Rightarrow$  bool list  $\Rightarrow$  bool **where**  
*cover-partition* p [t] [] = (t = 1)  
| *cover-partition* p (t # u # ts) (b # bs) =  
(t  $\in$  {0..1}  $\wedge$  u  $\in$  {0..1}  $\wedge$  t < u  $\wedge$   
(if b then subpathin t u p ' {0..1}  $\subseteq$  U else subpathin t u p ' {0..1}  $\subseteq$  V)  $\wedge$   
*cover-partition* p (u # ts) bs)  
| *cover-partition* p ts bs = False

**fun** *rectangle-partition* ::  
((real  $\times$  real)  $\Rightarrow$  'a)  $\Rightarrow$  real  $\Rightarrow$  real  $\Rightarrow$  real list  $\Rightarrow$  bool list  $\Rightarrow$  bool  
**where**  
*rectangle-partition* h c d [t] [] = (t = 1)  
| *rectangle-partition* h c d (t # u # ts) (b # bs) =  
(t  $\in$  {0..1}  $\wedge$  u  $\in$  {0..1}  $\wedge$  t < u  $\wedge$   
(if b then h ' ({t..u}  $\times$  {c..d})  $\subseteq$  U else h ' ({t..u}  $\times$  {c..d})  $\subseteq$  V)  $\wedge$   
*rectangle-partition* h c d (u # ts) bs)  
| *rectangle-partition* h c d ts bs = False

**fun** *alternating-bits* :: bool list  $\Rightarrow$  bool **where**  
*alternating-bits* [] = True  
| *alternating-bits* [b] = True  
| *alternating-bits* (b # c # bs) = (b  $\neq$  c  $\wedge$  *alternating-bits* (c # bs))

**fun** *partition-word* ::  
(real  $\Rightarrow$  'a)  $\Rightarrow$  real list  $\Rightarrow$  bool list  $\Rightarrow$   
((real  $\Rightarrow$  'a) set, (real  $\Rightarrow$  'a) set) free-product-word  
**where**  
*partition-word* p (t # u # ts) (b # bs) =  
(if b then WordLeft (loop-class U x0 (segment-loop p t u))  
else WordRight (loop-class V x0 (segment-loop p t u)))  
(*partition-word* p (u # ts) bs)  
| *partition-word* p ts bs = WordNil

**fun** *partition-word-with-tail* ::  
(real  $\Rightarrow$  'a)  $\Rightarrow$  real list  $\Rightarrow$  bool list  $\Rightarrow$   
((real  $\Rightarrow$  'a) set, (real  $\Rightarrow$  'a) set) free-product-word  $\Rightarrow$   
((real  $\Rightarrow$  'a) set, (real  $\Rightarrow$  'a) set) free-product-word  
**where**  
*partition-word-with-tail* p (t # u # ts) (b # bs) tail =  
(if b then WordLeft (loop-class U x0 (segment-loop p t u))  
else WordRight (loop-class V x0 (segment-loop p t u)))  
(*partition-word-with-tail* p (u # ts) bs tail)  
| *partition-word-with-tail* p ts bs tail = tail

```

fun bridge-word ::
  bool  $\Rightarrow$  (real  $\Rightarrow$  'a) set  $\Rightarrow$ 
    ((real  $\Rightarrow$  'a) set, (real  $\Rightarrow$  'a) set) free-product-word  $\Rightarrow$ 
    ((real  $\Rightarrow$  'a) set, (real  $\Rightarrow$  'a) set) free-product-word
where
  bridge-word True h rest = WordLeft (i1 h) rest
| bridge-word False h rest = WordRight (i2 h) rest

fun partition-loop :: (real  $\Rightarrow$  'a)  $\Rightarrow$  real list  $\Rightarrow$  real  $\Rightarrow$  'a where
  partition-loop p (t # u # ts) = segment-loop p t u +++ partition-loop p (u #
ts)
| partition-loop p ts = ( $\lambda$ -. x0)

lemma partition-word-with-tail-nil [simp]:
  partition-word-with-tail p ts bs WordNil = partition-word p ts bs
  <proof>

lemma subpathin-joinpaths-left-half [simp]:
  subpathin 0 (1 / 2) (p +++ q) ' {0..1} = p ' {0..1}
  <proof>

lemma affine-unit-interval:
  fixes u v t :: real
  assumes u01: u  $\in$  {0..1}
    and v01: v  $\in$  {0..1}
    and t01: t  $\in$  {0..1}
  shows (v - u) * t + u  $\in$  {0..1}
  <proof>

lemma subpathin-joinpaths-tail-scaled-pointwise:
  assumes q0: pathstart q = pathfinish p
    and u01: u  $\in$  {0..1}
    and v01: v  $\in$  {0..1}
    and t01: t  $\in$  {0..1}
  shows subpathin ((1 + u) / 2) ((1 + v) / 2) (p +++ q) t = subpathin u v q t
  <proof>

lemma subpathin-joinpaths-tail-scaled [simp]:
  assumes q0: pathstart q = pathfinish p
    and u01: u  $\in$  {0..1}
    and v01: v  $\in$  {0..1}
  shows subpathin ((1 + u) / 2) ((1 + v) / 2) (p +++ q) ' {0..1} = subpathin
u v q ' {0..1}
  <proof>

lemma homotopic-paths-join-left:
  assumes qr: homotopic-paths S q r
    and p-path: path p
    and p-img: path-image p  $\subseteq$  S

```

**and**  $pq$ :  $\text{pathfinish } p = \text{pathstart } q$   
**shows**  $\text{homotopic-paths } S (p \text{ +++ } q) (p \text{ +++ } r)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{homotopic-paths-join-right}$ :  
**assumes**  $pq$ :  $\text{homotopic-paths } S p q$   
**and**  $r\text{-path}$ :  $\text{path } r$   
**and**  $r\text{-img}$ :  $\text{path-image } r \subseteq S$   
**and**  $qr$ :  $\text{pathfinish } p = \text{pathstart } r$   
**shows**  $\text{homotopic-paths } S (p \text{ +++ } r) (q \text{ +++ } r)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{homotopic-paths-cancel-middle-local}$ :  
**assumes**  $r\text{-path}$ :  $\text{path } r$   
**and**  $r\text{-img}$ :  $\text{path-image } r \subseteq S$   
**and**  $c\text{-path}$ :  $\text{path } c$   
**and**  $c\text{-img}$ :  $\text{path-image } c \subseteq S$   
**and**  $s\text{-path}$ :  $\text{path } s$   
**and**  $s\text{-img}$ :  $\text{path-image } s \subseteq S$   
**and**  $rc$ :  $\text{pathfinish } r = \text{pathfinish } c$   
**and**  $cs$ :  $\text{pathstart } s = \text{pathfinish } c$   
**shows**  $\text{homotopic-paths } S (((r \text{ +++ } \text{reversepath } c) \text{ +++ } c) \text{ +++ } s) (r \text{ +++ } s)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{segment-loop-base-full-in-set}$ :  
**assumes**  $p\text{-loop}$ :  $p \in \text{loop-space } S x0$   
**shows**  $\text{homotopic-paths } S (\text{segment-loop } p 0 1) p$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{segment-loop-joinpaths-head}$  [simp]:  
**assumes**  $p\text{-loop}$ :  $p \in \text{loop-space } S x0$   
**and**  $q\text{-loop}$ :  $q \in \text{loop-space } W x0$   
**shows**  $\text{segment-loop } (p \text{ +++ } q) 0 (1 / 2) = \text{segment-loop } p 0 1$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{segment-loop-joinpaths-tail-scaled}$  [simp]:  
**assumes**  $p\text{-loop}$ :  $p \in \text{loop-space } W x0$   
**and**  $q\text{-loop}$ :  $q \in \text{loop-space } W x0$   
**and**  $u01$ :  $u \in \{0..1\}$   
**and**  $v01$ :  $v \in \{0..1\}$   
**shows**  $\text{segment-loop } (p \text{ +++ } q) ((1 + u) / 2) ((1 + v) / 2) = \text{segment-loop } q$   
 $u v$   
 $\langle \text{proof} \rangle$

**fun**  $\text{word-loop}$  ::  
 $((\text{real} \Rightarrow 'a) \text{ set}, (\text{real} \Rightarrow 'a) \text{ set}) \text{ free-product-word} \Rightarrow \text{real} \Rightarrow 'a$   
**where**  
 $\text{word-loop } \text{WordNil} = (\lambda \cdot. x0)$

```

| word-loop (WordLeft a rest) =
  (if rest = WordNil then some-loop U x0 a else some-loop U x0 a +++ word-loop
rest)
| word-loop (WordRight b rest) =
  (if rest = WordNil then some-loop V x0 b else some-loop V x0 b +++ word-loop
rest)

```

```

fun word-partition-times ::
  ((real ⇒ 'a) set, (real ⇒ 'a) set) free-product-word ⇒ real list
where
  word-partition-times WordNil = [0, 1]
| word-partition-times (WordLeft a rest) =
  (if rest = WordNil
  then [0, 1]
  else 0 # map (λt. (1 + t) / 2) (word-partition-times rest))
| word-partition-times (WordRight b rest) =
  (if rest = WordNil
  then [0, 1]
  else 0 # map (λt. (1 + t) / 2) (word-partition-times rest))

```

```

fun word-partition-bits ::
  ((real ⇒ 'a) set, (real ⇒ 'a) set) free-product-word ⇒ bool list
where
  word-partition-bits WordNil = [True]
| word-partition-bits (WordLeft a rest) =
  (if rest = WordNil then [True] else True # word-partition-bits rest)
| word-partition-bits (WordRight b rest) =
  (if rest = WordNil then [False] else False # word-partition-bits rest)

```

```

lemma joinpaths-tail-scaled-point [simp]:
  assumes p-loop: p ∈ loop-space W x0
  and q-loop: q ∈ loop-space W x0
  and t01: t ∈ {0..1}
  shows (p +++ q) ((1 + t) / 2) = q t
⟨proof⟩

```

```

lemma word-loop-in-W:
  assumes w-in: fpw-in-space G1 G2 w
  shows word-loop w ∈ loop-space W x0
⟨proof⟩

```

```

lemma svk-partition-joinpaths-tail-scaled:
  assumes p-loop: p ∈ loop-space W x0
  and q-loop: q ∈ loop-space W x0
  and q-part: svk-partition q ts bs
  shows svk-partition (p +++ q) (map (λt. (1 + t) / 2) ts) bs
⟨proof⟩

```

```

lemma word-loop-valid-partition:

```

**assumes**  $w\text{-in}$ :  $\text{fpw-in-space } G1 \ G2 \ w$   
**shows**  $\text{valid-partition } (\text{word-loop } w) (\text{word-partition-times } w) (\text{word-partition-bits } w)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{partition-word-joinpaths-tail-scaled}$ :  
**assumes**  $p\text{-loop}$ :  $p \in \text{loop-space } W \ x0$   
**and**  $q\text{-loop}$ :  $q \in \text{loop-space } W \ x0$   
**and**  $q\text{-part}$ :  $\text{svk-partition } q \ ts \ bs$   
**shows**  $\text{partition-word } (p \ +++ \ q) (\text{map } (\lambda t. (1 + t) / 2) \ ts) \ bs = \text{partition-word } q \ ts \ bs$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{segment-loop-some-loop-left-class}$ :  
**assumes**  $a\text{-in}$ :  $a \in G1$   
**shows**  $\text{loop-class } U \ x0 (\text{segment-loop } (\text{some-loop } U \ x0 \ a) \ 0 \ 1) = a$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{segment-loop-some-loop-right-class}$ :  
**assumes**  $b\text{-in}$ :  $b \in G2$   
**shows**  $\text{loop-class } V \ x0 (\text{segment-loop } (\text{some-loop } V \ x0 \ b) \ 0 \ 1) = b$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{partition-word-word-loop-equiv}$ :  
**assumes**  $w\text{-in}$ :  $\text{fpw-in-space } G1 \ G2 \ w$   
**shows**  $\text{carrier-full-amalg-equiv } G1 \ G2 \ H \ i1 \ i2 \ \text{mult1} \ \text{one1} \ \text{mult2} \ \text{one2}$   
 $(\text{partition-word } (\text{word-loop } w) (\text{word-partition-times } w) (\text{word-partition-bits } w))$   
 $w$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{valid-partition-hd}$ :  
**assumes**  $\text{valid-partition } p \ ts \ bs$   
**shows**  $ts \neq [] \ \text{hd } ts = 0$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{valid-partition-cases}$ :  
**assumes**  $\text{valid-partition } p \ (t \ \# \ ts) \ bs$   
**shows**  $t = 0$  **and**  $\text{svk-partition } p \ (t \ \# \ ts) \ bs$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{svk-partition-head-props}$ :  
**assumes**  $\text{svk-partition } p \ (t \ \# \ ts) \ bs$   
**shows**  $t \in \{0..1\}$  **and**  $p \ t \in U \cap V$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{svk-partition-tail}$ :  
**assumes**  $\text{svk-partition } p \ (t \ \# \ u \ \# \ ts) \ (b \ \# \ bs)$   
**shows**  $\text{svk-partition } p \ (u \ \# \ ts) \ bs$   
 $\langle \text{proof} \rangle$

**lemma** *svk-partition-step-props*:  
**assumes** *svk-partition*  $p$  ( $t \# u \# ts$ ) ( $b \# bs$ )  
**shows**  $t \in \{0..1\}$   
**and**  $p \ t \in U \cap V$   
**and**  $u \in \{0..1\}$   
**and**  $t < u$   
**and** (if  $b$  then  $\text{subpathin } t \ u \ p \ ' \ \{0..1\} \subseteq U$  else  $\text{subpathin } t \ u \ p \ ' \ \{0..1\} \subseteq V$ )  
 $\langle \text{proof} \rangle$

**lemma** *svk-partition-next-in-intersection*:  
**assumes** *svk-partition*  $p$  ( $t \# u \# ts$ ) ( $b \# bs$ )  
**shows**  $p \ u \in U \cap V$   
 $\langle \text{proof} \rangle$

**lemma** *svk-partition-nonempty*:  
**assumes** *svk-partition*  $p \ ts \ bs$   
**shows**  $ts \neq []$   
 $\langle \text{proof} \rangle$

**lemma** *svk-partition-last-eq-one*:  
**assumes** *part: svk-partition*  $p \ ts \ bs$   
**shows**  $\text{last } ts = 1$   
 $\langle \text{proof} \rangle$

**lemma** *svk-partition-last-in-intersection*:  
**assumes** *part: svk-partition*  $p \ ts \ bs$   
**shows**  $p \ (\text{last } ts) \in U \cap V$   
 $\langle \text{proof} \rangle$

**lemma** *svk-partition-last-props*:  
**assumes** *part: svk-partition*  $p \ ts \ bs$   
**shows**  $ts \neq []$  **and**  $\text{last } ts = 1$  **and**  $p \ (\text{last } ts) \in U \cap V$   
 $\langle \text{proof} \rangle$

**lemma** *valid-partition-last-props*:  
**assumes** *valid-partition*  $p \ ts \ bs$   
**shows**  $ts \neq []$  **and**  $\text{last } ts = 1$  **and**  $p \ (\text{last } ts) \in U \cap V$   
 $\langle \text{proof} \rangle$

**lemma** *subpathin-endpoints-in-set*:  
**assumes** *seg-in: subpathin*  $u \ v \ p \ ' \ \{0..1\} \subseteq S$   
**shows**  $p \ u \in S$  **and**  $p \ v \in S$   
 $\langle \text{proof} \rangle$

**lemma** *subpathin-image-subset-union*:  
**assumes**  $tu: t \leq u$   
**and**  $uv: u \leq v$   
**shows**  $\text{subpathin } t \ v \ p \ ' \ \{0..1\} \subseteq \text{subpathin } t \ u \ p \ ' \ \{0..1\} \cup \text{subpathin } u \ v \ p \ ' \ \{0..1\}$

$\{0..1\}$   
 $\langle proof \rangle$

**lemma** *subpathin-image-subset-trans:*

**assumes**  $tu: t \leq u$   
**and**  $uv: u \leq v$   
**and**  $left: subpathin\ t\ u\ p\ \{0..1\} \subseteq S$   
**and**  $right: subpathin\ u\ v\ p\ \{0..1\} \subseteq S$   
**shows**  $subpathin\ t\ v\ p\ \{0..1\} \subseteq S$   
 $\langle proof \rangle$

**lemma** *cover-partition-step-props:*

**assumes**  $cover-partition\ p\ (t\ \# u\ \# ts)\ (b\ \# bs)$   
**shows**  $t \in \{0..1\}$   
**and**  $u \in \{0..1\}$   
**and**  $t < u$   
**and**  $(if\ b\ then\ subpathin\ t\ u\ p\ \{0..1\} \subseteq U\ else\ subpathin\ t\ u\ p\ \{0..1\} \subseteq V)$   
**and**  $cover-partition\ p\ (u\ \# ts)\ bs$   
 $\langle proof \rangle$

**lemma** *cover-partition-consI:*

**assumes**  $t \in \{0..1\}$   
**and**  $u \in \{0..1\}$   
**and**  $t < u$   
**and**  $(if\ b\ then\ subpathin\ t\ u\ p\ \{0..1\} \subseteq U\ else\ subpathin\ t\ u\ p\ \{0..1\} \subseteq V)$   
**and**  $cover-partition\ p\ (u\ \# ts)\ bs$   
**shows**  $cover-partition\ p\ (t\ \# u\ \# ts)\ (b\ \# bs)$   
 $\langle proof \rangle$

**lemma** *cover-partition-switch-point:*

**assumes**  $cp: cover-partition\ p\ (t\ \# u\ \# v\ \# ts)\ (b\ \# c\ \# bs)$   
**and**  $diff: b \neq c$   
**shows**  $p\ u \in U \cap V$   
 $\langle proof \rangle$

**lemma** *cover-partition-pair-svk-partition:*

**assumes**  $cp: cover-partition\ p\ [t, u]\ [b]$   
**and**  $ptUV: p\ t \in U \cap V$   
**and**  $puUV: p\ u \in U \cap V$   
**shows**  $svk-partition\ p\ [t, u]\ [b]$   
 $\langle proof \rangle$

**lemma** *cover-partition-compress-svk-partition:*

**assumes**  $cp: cover-partition\ p\ (t\ \# ts)\ bs$   
**and**  $ptUV: p\ t \in U \cap V$   
**and**  $plastUV: p\ (last\ (t\ \# ts)) \in U \cap V$   
**shows**  $\exists ts'\ bs'. svk-partition\ p\ (t\ \# ts')\ bs'$   
 $\langle proof \rangle$

**lemma** *cover-partition-last-eq-one*:  
**assumes** *cp*: *cover-partition p ts bs*  
**and** *ts-ne*:  $ts \neq []$   
**shows**  $last\ ts = 1$   
 $\langle proof \rangle$

**lemma** *nat-real-div-in-unit-interval*:  
**assumes** *n-pos*:  $0 < n$   
**and** *i-le*:  $i \leq n$   
**shows**  $real\ i / real\ n \in \{0..1\}$   
 $\langle proof \rangle$

**lemma** *nat-real-div-strict-mono*:  
**assumes** *n-pos*:  $0 < n$   
**and** *i-lt*:  $i < n$   
**shows**  $real\ i / real\ n < real\ (Suc\ i) / real\ n$   
 $\langle proof \rangle$

**fun** *subdivision-times* ::  $nat \Rightarrow nat \Rightarrow real\ list$  **where**  
*subdivision-times* *n* 0 = [1]  
| *subdivision-times* *n* (Suc *k*) =  $real\ (n - Suc\ k) / real\ n \# subdivision-times\ n\ k$

**fun** *subdivision-bits* ::  $(nat \Rightarrow bool) \Rightarrow nat \Rightarrow nat \Rightarrow bool\ list$  **where**  
*subdivision-bits* *side* *n* 0 = []  
| *subdivision-bits* *side* *n* (Suc *k*) =  $side\ (n - Suc\ k) \# subdivision-bits\ side\ n\ k$

**lemma** *cover-partition-subdivision-from*:  
**assumes** *n-pos*:  $0 < n$   
**and** *k-le*:  $k \leq n$   
**and** *cover*:  $\bigwedge i. n - k \leq i \implies i < n \implies$   
    *(if* *side* *i*  
    then  $subpathin\ (real\ i / real\ n)\ (real\ (Suc\ i) / real\ n)\ p\ '\{0..1\} \subseteq U$   
    else  $subpathin\ (real\ i / real\ n)\ (real\ (Suc\ i) / real\ n)\ p\ '\{0..1\} \subseteq V$   
**shows** *cover-partition* *p* (*subdivision-times* *n* *k*) (*subdivision-bits* *side* *n* *k*)  
 $\langle proof \rangle$

**lemma** *subdivision-times-start*:  
**assumes** *n-pos*:  $0 < n$   
**shows**  $subdivision-times\ n\ n = 0 \# subdivision-times\ n\ (n - 1)$   
 $\langle proof \rangle$

**lemma** *loop-has-valid-partition*:  
**assumes** *p-loop*:  $p \in loop-space\ W\ x0$   
**shows**  $\exists ts\ bs. valid-partition\ p\ ts\ bs$   
 $\langle proof \rangle$

**lemma** *svk-partition-partition-word-in-space*:  
**assumes** *p-loop*:  $p \in loop-space\ W\ x0$   
**and** *part*: *svk-partition* *p* *ts* *bs*

**shows** *fpw-in-space*  $G1\ G2$  (*partition-word*  $p\ ts\ bs$ )  
(*proof*)

**lemma** *valid-partition-partition-word-in-space*:  
**assumes** *p-loop*:  $p \in \text{loop-space } W\ x0$   
**and** *part*: *valid-partition*  $p\ ts\ bs$   
**shows** *fpw-in-space*  $G1\ G2$  (*partition-word*  $p\ ts\ bs$ )  
(*proof*)

**lemma** *svk-partition-partition-loop-in-W*:  
**assumes** *p-loop*:  $p \in \text{loop-space } W\ x0$   
**and** *part*: *svk-partition*  $p\ ts\ bs$   
**shows** *partition-loop*  $p\ ts \in \text{loop-space } W\ x0$   
(*proof*)

**lemma** *valid-partition-partition-loop-in-W*:  
**assumes** *p-loop*:  $p \in \text{loop-space } W\ x0$   
**and** *part*: *valid-partition*  $p\ ts\ bs$   
**shows** *partition-loop*  $p\ ts \in \text{loop-space } W\ x0$   
(*proof*)

**lemma** *i1-loop-class-eq*:  
**assumes** *p-loop*:  $p \in \text{loop-space } (U \cap V)\ x0$   
**shows** *i1* (*loop-class*  $(U \cap V)\ x0\ p$ ) = *loop-class*  $U\ x0\ p$   
(*proof*)

**lemma** *i2-loop-class-eq*:  
**assumes** *p-loop*:  $p \in \text{loop-space } (U \cap V)\ x0$   
**shows** *i2* (*loop-class*  $(U \cap V)\ x0\ p$ ) = *loop-class*  $V\ x0\ p$   
(*proof*)

**lemma** *j1-segment-loop-eq*:  
**assumes** *segU*: *segment-loop*  $p\ t\ u \in \text{loop-space } U\ x0$   
**shows** *j1* (*loop-class*  $U\ x0$  (*segment-loop*  $p\ t\ u$ )) =  
*loop-class*  $W\ x0$  (*segment-loop*  $p\ t\ u$ )  
(*proof*)

**lemma** *j2-segment-loop-eq*:  
**assumes** *segV*: *segment-loop*  $p\ t\ u \in \text{loop-space } V\ x0$   
**shows** *j2* (*loop-class*  $V\ x0$  (*segment-loop*  $p\ t\ u$ )) =  
*loop-class*  $W\ x0$  (*segment-loop*  $p\ t\ u$ )  
(*proof*)

**lemma** *svk-partition-eval-partition-word*:  
**assumes** *p-loop*:  $p \in \text{loop-space } W\ x0$   
**and** *part*: *svk-partition*  $p\ ts\ bs$   
**shows** *svk-word-eval* (*partition-word*  $p\ ts\ bs$ ) =  
*loop-class*  $W\ x0$  (*partition-loop*  $p\ ts$ )  
(*proof*)

**lemma** *valid-partition-eval-partition-word*:

**assumes** *p-loop*:  $p \in \text{loop-space } W \ x0$   
**and** *part*: *valid-partition*  $p \ ts \ bs$   
**shows** *svk-word-eval* (*partition-word*  $p \ ts \ bs$ ) =  
*loop-class*  $W \ x0 \ (\text{partition-loop } p \ ts)$   
*<proof>*

**lemma** *valid-partition-decode-partition-word*:

**assumes** *p-loop*:  $p \in \text{loop-space } W \ x0$   
**and** *part*: *valid-partition*  $p \ ts \ bs$   
**shows** *svk-decode* (*partition-word*  $p \ ts \ bs$ ) =  
*loop-class*  $W \ x0 \ (\text{partition-loop } p \ ts)$   
*<proof>*

**lemma** *pair-interval-member*:

**fixes**  $x \ y :: \text{real} \times \text{real}$  **and**  $x1 \ x2 \ y1 \ y2 \ u \ v :: \text{real}$   
**assumes**  $x: x = (x1, x2)$   
**and**  $y: y = (y1, y2)$   
**and** *mix1*:  $u *_{\mathbb{R}} x1 + v *_{\mathbb{R}} y1 \in \{0..1\}$   
**and** *mix2*:  $u *_{\mathbb{R}} x2 + v *_{\mathbb{R}} y2 \in \{0..1\}$   
**shows**  $u *_{\mathbb{R}} x + v *_{\mathbb{R}} y \in \{0..1\} \times \{0..1\}$   
*<proof>*

**lemma** *affine-closed-segment-member*:

**fixes**  $a \ b \ u :: \text{real}$   
**assumes** *u01*:  $u \in \{0..1\}$   
**shows**  $(b - a) * u + a \in \text{closed-segment } a \ b$   
*<proof>*

**lemma** *affine-subinterval-member*:

**fixes**  $a \ b \ u :: \text{real}$   
**assumes** *ab*:  $a \leq b$   
**and** *u01*:  $u \in \{0..1\}$   
**shows**  $(b - a) * u + a \in \{a..b\}$   
*<proof>*

**lemma** *affine-unit-interval-member*:

**fixes**  $a \ b \ u :: \text{real}$   
**assumes** *a01*:  $a \in \{0..1\}$   
**and** *b01*:  $b \in \{0..1\}$   
**and** *ab*:  $a \leq b$   
**and** *u01*:  $u \in \{0..1\}$   
**shows**  $(b - a) * u + a \in \{0..1\}$   
*<proof>*

**lemma** *square-edge-homotopic*:

**fixes**  $h :: (\text{real} \times \text{real}) \Rightarrow 'a$   
**assumes** *h-cont*: *continuous-map* (*top-of-set* ( $\{0..1\} \times \{0..1\}$ )) (*top-of-set*  $S$ )  $h$

**shows** *homotopic-paths*  $S$   
 $((\lambda t. h (t, 0)) \text{ +++ } (\lambda t. h (1, t)))$   
 $((\lambda t. h (0, t)) \text{ +++ } (\lambda t. h (t, 1)))$   
 $\langle \text{proof} \rangle$

**lemma** *rectangle-edge-homotopic*:

**fixes**  $h :: (\text{real} \times \text{real}) \Rightarrow 'a$   
**assumes**  $h\text{-cont}$ : *continuous-map* (*top-of-set* ( $\{0..1\} \times \{0..1\}$ )) (*top-of-set*  $S$ )  $h$   
**and**  $a01$ :  $a \in \{0..1\}$  **and**  $b01$ :  $b \in \{0..1\}$   
**and**  $c01$ :  $c \in \{0..1\}$  **and**  $d01$ :  $d \in \{0..1\}$   
**and**  $ab$ :  $a \leq b$  **and**  $cd$ :  $c \leq d$   
**shows** *homotopic-paths*  $S$   
 $(\text{subpathin } a \ b \ (\lambda t. h (t, c)) \text{ +++ } (\lambda t. h (b, (d - c) * t + c)))$   
 $((\lambda t. h (a, (d - c) * t + c)) \text{ +++ } \text{subpathin } a \ b \ (\lambda t. h (t, d)))$   
 $\langle \text{proof} \rangle$

**lemma** *rectangle-edge-homotopic-in-set*:

**fixes**  $h :: (\text{real} \times \text{real}) \Rightarrow 'a$   
**assumes**  $h\text{-cont}$ : *continuous-map* (*top-of-set* ( $\{0..1\} \times \{0..1\}$ )) (*top-of-set*  $W$ )  
 $h$   
**and**  $a01$ :  $a \in \{0..1\}$  **and**  $b01$ :  $b \in \{0..1\}$   
**and**  $c01$ :  $c \in \{0..1\}$  **and**  $d01$ :  $d \in \{0..1\}$   
**and**  $ab$ :  $a \leq b$  **and**  $cd$ :  $c \leq d$   
**and**  $\text{rect}S$ :  $h \text{ ' } (\{a..b\} \times \{c..d\}) \subseteq S$   
**shows** *homotopic-paths*  $S$   
 $(\text{subpathin } a \ b \ (\lambda t. h (t, c)) \text{ +++ } (\lambda t. h (b, (d - c) * t + c)))$   
 $((\lambda t. h (a, (d - c) * t + c)) \text{ +++ } \text{subpathin } a \ b \ (\lambda t. h (t, d)))$   
 $\langle \text{proof} \rangle$

**definition** *vertical-strip-path* ::

$((\text{real} \times \text{real}) \Rightarrow 'a) \Rightarrow \text{real} \Rightarrow \text{real} \Rightarrow \text{real} \Rightarrow \text{real} \Rightarrow 'a$   
**where**  
 $\text{vertical-strip-path } h \ t \ c \ d = (\lambda u. h (t, (d - c) * u + c))$

**definition** *bridge-loop* ::

$((\text{real} \times \text{real}) \Rightarrow 'a) \Rightarrow \text{real} \Rightarrow \text{real} \Rightarrow \text{real} \Rightarrow \text{real} \Rightarrow 'a$   
**where**  
 $\text{bridge-loop } h \ t \ c \ d =$   
 $(\text{connector } (h (t, c)) \text{ +++ } \text{vertical-strip-path } h \ t \ c \ d) \text{ +++}$   
 $\text{reversepath } (\text{connector } (h (t, d)))$

**lemma** *bridge-loop-eq-segment-loop* [*simp*]:

$\text{bridge-loop } h \ t \ c \ d = \text{segment-loop } (\text{vertical-strip-path } h \ t \ c \ d) \ 0 \ 1$   
 $\langle \text{proof} \rangle$

**lemma** *vertical-strip-path-image-subset*:

**assumes**  $cd$ :  $c \leq d$   
**shows**  $\text{vertical-strip-path } h \ t \ c \ d \text{ ' } \{0..1\} \subseteq h \text{ ' } (\{t\} \times \{c..d\})$   
 $\langle \text{proof} \rangle$

**lemma** *rectangle-segment-loop-bridge-homotopic*:  
**fixes**  $h :: (\text{real} \times \text{real}) \Rightarrow 'a$   
**assumes**  $h\text{-cont}$ : *continuous-map* (*top-of-set* ( $\{0..1\} \times \{0..1\}$ )) (*top-of-set*  $W$ )  
 $h$   
**and**  $a01$ :  $a \in \{0..1\}$  **and**  $b01$ :  $b \in \{0..1\}$   
**and**  $c01$ :  $c \in \{0..1\}$  **and**  $d01$ :  $d \in \{0..1\}$   
**and**  $ab$ :  $a \leq b$  **and**  $cd$ :  $c \leq d$   
**and**  $rectS$ :  $h \text{ ` } (\{a..b\} \times \{c..d\}) \subseteq S$   
**and**  $leftUV$ :  $h \text{ ` } (\{a\} \times \{c..d\}) \subseteq U \cap V$   
**and**  $rightUV$ :  $h \text{ ` } (\{b\} \times \{c..d\}) \subseteq U \cap V$   
**and**  $UVS$ :  $U \cap V \subseteq S$   
**shows** *homotopic-paths*  $S$   
(*segment-loop* ( $\lambda t. h (t, c)$ )  $a b$  +++ *bridge-loop*  $h b c d$ )  
(*bridge-loop*  $h a c d$  +++ *segment-loop* ( $\lambda t. h (t, d)$ )  $a b$ )  
 $\langle proof \rangle$

**lemma** *horizontal-rectangle-segment-loop-in-set*:  
**fixes**  $h :: (\text{real} \times \text{real}) \Rightarrow 'a$   
**assumes**  $h\text{-cont}$ : *continuous-map* (*top-of-set* ( $\{0..1\} \times \{0..1\}$ )) (*top-of-set*  $W$ )  
 $h$   
**and**  $a01$ :  $a \in \{0..1\}$  **and**  $b01$ :  $b \in \{0..1\}$  **and**  $c01$ :  $c \in \{0..1\}$   
**and**  $ab$ :  $a \leq b$   
**and**  $segS$ :  $h \text{ ` } (\{a..b\} \times \{c\}) \subseteq S$   
**and**  $acUV$ :  $h (a, c) \in U \cap V$   
**and**  $bcUV$ :  $h (b, c) \in U \cap V$   
**and**  $UVS$ :  $U \cap V \subseteq S$   
**shows** *segment-loop* ( $\lambda t. h (t, c)$ )  $a b \in \text{loop-space } S x0$   
 $\langle proof \rangle$

**lemma** *vertical-bridge-loop-in-set*:  
**fixes**  $h :: (\text{real} \times \text{real}) \Rightarrow 'a$   
**assumes**  $h\text{-cont}$ : *continuous-map* (*top-of-set* ( $\{0..1\} \times \{0..1\}$ )) (*top-of-set*  $W$ )  
 $h$   
**and**  $t01$ :  $t \in \{0..1\}$  **and**  $c01$ :  $c \in \{0..1\}$  **and**  $d01$ :  $d \in \{0..1\}$   
**and**  $cd$ :  $c \leq d$   
**and**  $edgeUV$ :  $h \text{ ` } (\{t\} \times \{c..d\}) \subseteq U \cap V$   
**and**  $UVS$ :  $U \cap V \subseteq S$   
**shows** *bridge-loop*  $h t c d \in \text{loop-space } S x0$   
 $\langle proof \rangle$

**lemma** *rectangle-segment-loop-bridge-class-eq*:  
**fixes**  $h :: (\text{real} \times \text{real}) \Rightarrow 'a$   
**assumes**  $h\text{-cont}$ : *continuous-map* (*top-of-set* ( $\{0..1\} \times \{0..1\}$ )) (*top-of-set*  $W$ )  
 $h$   
**and**  $a01$ :  $a \in \{0..1\}$  **and**  $b01$ :  $b \in \{0..1\}$   
**and**  $c01$ :  $c \in \{0..1\}$  **and**  $d01$ :  $d \in \{0..1\}$   
**and**  $ab$ :  $a \leq b$  **and**  $cd$ :  $c \leq d$   
**and**  $rectS$ :  $h \text{ ` } (\{a..b\} \times \{c..d\}) \subseteq S$

**and** *leftUV*:  $h \cdot (\{a\} \times \{c..d\}) \subseteq U \cap V$   
**and** *rightUV*:  $h \cdot (\{b\} \times \{c..d\}) \subseteq U \cap V$   
**and** *UVS*:  $U \cap V \subseteq S$   
**shows** *fundamental-group-mult*  $S \ x0$   
    (*loop-class*  $S \ x0$  (*segment-loop*  $(\lambda t. h \ (t, c)) \ a \ b$ ))  
    (*loop-class*  $S \ x0$  (*bridge-loop*  $h \ b \ c \ d$ )) =  
*fundamental-group-mult*  $S \ x0$   
    (*loop-class*  $S \ x0$  (*bridge-loop*  $h \ a \ c \ d$ ))  
    (*loop-class*  $S \ x0$  (*segment-loop*  $(\lambda t. h \ (t, d)) \ a \ b$ ))  
⟨*proof*⟩

**lemma** *bridge-word-identify*:  
**assumes** *h-in*:  $h \in H$   
**shows** *carrier-full-amalg-equiv*  $G1 \ G2 \ H \ i1 \ i2 \ mult1 \ one1 \ mult2 \ one2$   
    (*bridge-word*  $True \ h \ rest$ ) (*bridge-word*  $False \ h \ rest$ )  
**and** *carrier-full-amalg-equiv*  $G1 \ G2 \ H \ i1 \ i2 \ mult1 \ one1 \ mult2 \ one2$   
    (*bridge-word*  $False \ h \ rest$ ) (*bridge-word*  $True \ h \ rest$ )  
⟨*proof*⟩

**lemma** *rectangle-partition-step-props*:  
**assumes** *rectangle-partition*  $h \ c \ d \ (t \ \# \ u \ \# \ ts) \ (b \ \# \ bs)$   
**shows**  $t \in \{0..1\}$   
**and**  $u \in \{0..1\}$   
**and**  $t < u$   
**and** (*if*  $b$  *then*  $h \cdot (\{t..u\} \times \{c..d\}) \subseteq U$  *else*  $h \cdot (\{t..u\} \times \{c..d\}) \subseteq V$ )  
**and** *rectangle-partition*  $h \ c \ d \ (u \ \# \ ts) \ bs$   
⟨*proof*⟩

**lemma** *rectangle-partition-switch-edge*:  
**assumes** *rp*: *rectangle-partition*  $h \ c \ d \ (t \ \# \ u \ \# \ v \ \# \ ts) \ (b \ \# \ e \ \# \ bs)$   
**and** *diff*:  $b \neq e$   
**shows**  $h \cdot (\{u\} \times \{c..d\}) \subseteq U \cap V$   
⟨*proof*⟩

**lemma** *carrier-full-amalg-equiv-side-context*:  
**assumes** *rel*: *carrier-full-amalg-equiv*  $G1 \ G2 \ H \ i1 \ i2 \ mult1 \ one1 \ mult2 \ one2 \ u \ v$   
**and** *a-in*: (*if*  $b$  *then*  $a \in G1$  *else*  $a \in G2$ )  
**shows** *carrier-full-amalg-equiv*  $G1 \ G2 \ H \ i1 \ i2 \ mult1 \ one1 \ mult2 \ one2$   
    (*if*  $b$  *then* *WordLeft*  $a \ u$  *else* *WordRight*  $a \ u$ )  
    (*if*  $b$  *then* *WordLeft*  $a \ v$  *else* *WordRight*  $a \ v$ )  
⟨*proof*⟩

**lemma** *bridge-word-switch*:  
**assumes** *h-in*:  $h \in H$   
**and** *bc*:  $b \neq c$   
**shows** *carrier-full-amalg-equiv*  $G1 \ G2 \ H \ i1 \ i2 \ mult1 \ one1 \ mult2 \ one2$   
    (*bridge-word*  $b \ h \ rest$ ) (*bridge-word*  $c \ h \ rest$ )  
⟨*proof*⟩

**lemma** *rectangle-segment-bridge-left-equiv*:  
**fixes**  $h :: (\text{real} \times \text{real}) \Rightarrow 'a$   
**assumes**  $h\text{-cont}$ : *continuous-map* (*top-of-set* ( $\{0..1\} \times \{0..1\}$ )) (*top-of-set*  $W$ )  
 $h$   
**and**  $a01$ :  $a \in \{0..1\}$  **and**  $b01$ :  $b \in \{0..1\}$   
**and**  $c01$ :  $c \in \{0..1\}$  **and**  $d01$ :  $d \in \{0..1\}$   
**and**  $ab$ :  $a \leq b$  **and**  $cd$ :  $c \leq d$   
**and**  $rectU$ :  $h \text{ ' } (\{a..b\} \times \{c..d\}) \subseteq U$   
**and**  $leftUV$ :  $h \text{ ' } (\{a\} \times \{c..d\}) \subseteq U \cap V$   
**and**  $rightUV$ :  $h \text{ ' } (\{b\} \times \{c..d\}) \subseteq U \cap V$   
**and**  $rest\text{-in}$ : *fpw-in-space*  $G1\ G2\ rest$   
**shows** *carrier-full-amalg-equiv*  $G1\ G2\ H\ i1\ i2\ mult1\ one1\ mult2\ one2$   
(*WordLeft* (*loop-class*  $U\ x0$  (*segment-loop* ( $\lambda t. h\ (t, c)$ )  $a\ b$ ))  
(*bridge-word*  $True$  (*loop-class* ( $U \cap V$ )  $x0$  (*bridge-loop*  $h\ b\ c\ d$ ))  $rest$ ))  
(*bridge-word*  $True$  (*loop-class* ( $U \cap V$ )  $x0$  (*bridge-loop*  $h\ a\ c\ d$ ))  
(*WordLeft* (*loop-class*  $U\ x0$  (*segment-loop* ( $\lambda t. h\ (t, d)$ )  $a\ b$ ))  $rest$ ))  
 $\langle proof \rangle$

**lemma** *rectangle-segment-bridge-right-equiv*:  
**fixes**  $h :: (\text{real} \times \text{real}) \Rightarrow 'a$   
**assumes**  $h\text{-cont}$ : *continuous-map* (*top-of-set* ( $\{0..1\} \times \{0..1\}$ )) (*top-of-set*  $W$ )  
 $h$   
**and**  $a01$ :  $a \in \{0..1\}$  **and**  $b01$ :  $b \in \{0..1\}$   
**and**  $c01$ :  $c \in \{0..1\}$  **and**  $d01$ :  $d \in \{0..1\}$   
**and**  $ab$ :  $a \leq b$  **and**  $cd$ :  $c \leq d$   
**and**  $rectV$ :  $h \text{ ' } (\{a..b\} \times \{c..d\}) \subseteq V$   
**and**  $leftUV$ :  $h \text{ ' } (\{a\} \times \{c..d\}) \subseteq U \cap V$   
**and**  $rightUV$ :  $h \text{ ' } (\{b\} \times \{c..d\}) \subseteq U \cap V$   
**and**  $rest\text{-in}$ : *fpw-in-space*  $G1\ G2\ rest$   
**shows** *carrier-full-amalg-equiv*  $G1\ G2\ H\ i1\ i2\ mult1\ one1\ mult2\ one2$   
(*WordRight* (*loop-class*  $V\ x0$  (*segment-loop* ( $\lambda t. h\ (t, c)$ )  $a\ b$ ))  
(*bridge-word*  $False$  (*loop-class* ( $U \cap V$ )  $x0$  (*bridge-loop*  $h\ b\ c\ d$ ))  $rest$ ))  
(*bridge-word*  $False$  (*loop-class* ( $U \cap V$ )  $x0$  (*bridge-loop*  $h\ a\ c\ d$ ))  
(*WordRight* (*loop-class*  $V\ x0$  (*segment-loop* ( $\lambda t. h\ (t, d)$ )  $a\ b$ ))  $rest$ ))  
 $\langle proof \rangle$

**lemma** *rectangle-partition-partition-word-with-tail-in-space*:  
**fixes**  $h :: (\text{real} \times \text{real}) \Rightarrow 'a$   
**assumes**  $h\text{-cont}$ : *continuous-map* (*top-of-set* ( $\{0..1\} \times \{0..1\}$ )) (*top-of-set*  $W$ )  
 $h$   
**and**  $c01$ :  $c \in \{0..1\}$  **and**  $d01$ :  $d \in \{0..1\}$  **and**  $cd$ :  $c \leq d$   
**and**  $y\text{-in}$ :  $y \in \{c, d\}$   
**and**  $part$ : *rectangle-partition*  $h\ c\ d\ ts\ bs$   
**and**  $edgeUV$ :  $\bigwedge t. t \in \text{set } ts \implies h \text{ ' } (\{t\} \times \{c..d\}) \subseteq U \cap V$   
**and**  $tail\text{-in}$ : *fpw-in-space*  $G1\ G2\ rest$   
**shows** *fpw-in-space*  $G1\ G2$  (*partition-word-with-tail* ( $\lambda t. h\ (t, y)$ )  $ts\ bs\ rest$ )  
 $\langle proof \rangle$

**lemma** *rectangle-partition-partition-word-with-tail-equiv*:

**fixes**  $h :: (\text{real} \times \text{real}) \Rightarrow 'a$   
**assumes**  $h\text{-cont}$ : *continuous-map* (*top-of-set* ( $\{0..1\} \times \{0..1\}$ )) (*top-of-set*  $W$ )  
 $h$   
**and**  $c01$ :  $c \in \{0..1\}$  **and**  $d01$ :  $d \in \{0..1\}$  **and**  $cd$ :  $c \leq d$   
**and**  $part$ : *rectangle-partition*  $h$   $c$   $d$  ( $t \# u \# ts$ ) ( $b \# bs$ )  
**and**  $edgeUV$ :  $\bigwedge x. x \in \text{set } (t \# u \# ts) \implies h \text{ ' } (\{x\} \times \{c..d\}) \subseteq U \cap V$   
**and**  $rest\text{-in}$ : *fpw-in-space*  $G1$   $G2$   $rest$   
**shows** *carrier-full-amalg-equiv*  $G1$   $G2$   $H$   $i1$   $i2$   $mult1$   $one1$   $mult2$   $one2$   
*(partition-word-with-tail* ( $\lambda x. h$  ( $x$ ,  $c$ )) ( $t \# u \# ts$ ) ( $b \# bs$ )  
*(bridge-word* ( $\text{last}$  ( $b \# bs$ ))  
*(loop-class* ( $U \cap V$ )  $x0$  (*bridge-loop*  $h$  ( $\text{last}$  ( $t \# u \# ts$ ))  $c$   $d$ ))  $rest$ )  
*(bridge-word*  $b$  (*loop-class* ( $U \cap V$ )  $x0$  (*bridge-loop*  $h$   $t$   $c$   $d$ ))  
*(partition-word-with-tail* ( $\lambda x. h$  ( $x$ ,  $d$ )) ( $t \# u \# ts$ ) ( $b \# bs$ )  $rest$ ))  
*(proof)*

**lemma** *homotopic-paths-join-subpathin*:

**assumes**  $p\text{-path}$ : *path*  $p$   
**and**  $p\text{-img}$ : *path-image*  $p \subseteq S$   
**and**  $u01$ :  $u \in \{0..1\}$   
**and**  $v01$ :  $v \in \{0..1\}$   
**and**  $w01$ :  $w \in \{0..1\}$   
**and**  $uv$ :  $u \leq v$   
**and**  $vw$ :  $v \leq w$   
**shows** *homotopic-paths*  $S$  (*subpathin*  $u$   $v$   $p$  +++ *subpathin*  $v$   $w$   $p$ ) (*subpathin*  $u$   $w$   $p$ )  
*(proof)*

**lemma** *subpathin-full* [*simp*]:

*subpathin*  $0$   $1$   $p = p$   
*(proof)*

**lemma** *segment-loop-refl*:

**assumes**  $puUV$ :  $p$   $u \in U \cap V$   
**shows** *homotopic-paths*  $W$  (*segment-loop*  $p$   $u$   $u$ ) ( $\lambda\text{-}$ .  $x0$ )  
*(proof)*

**lemma** *segment-loop-full*:

**assumes**  $p\text{-loop}$ :  $p \in \text{loop-space } W$   $x0$   
**shows** *homotopic-paths*  $W$  (*segment-loop*  $p$   $0$   $1$ )  $p$   
*(proof)*

**lemma** *homotopic-paths-cancel-middle*:

**assumes**  $r\text{-path}$ : *path*  $r$   
**and**  $r\text{-img}$ : *path-image*  $r \subseteq S$   
**and**  $c\text{-path}$ : *path*  $c$   
**and**  $c\text{-img}$ : *path-image*  $c \subseteq S$   
**and**  $s\text{-path}$ : *path*  $s$   
**and**  $s\text{-img}$ : *path-image*  $s \subseteq S$   
**and**  $rc$ : *pathfinish*  $r = \text{pathfinish } c$

**and**  $cs$ :  $\text{pathstart } s = \text{pathfinish } c$   
**shows**  $\text{homotopic-paths } S$  ( $((r \text{ +++ } \text{reversepath } c) \text{ +++ } c) \text{ +++ } s$ ) ( $r \text{ +++ } s$ )  
 $\langle \text{proof} \rangle$

**lemma**  $\text{segment-loop-join}$ :

**assumes**  $p\text{-path}$ :  $\text{path } p$   
**and**  $p\text{-img}$ :  $\text{path-image } p \subseteq W$   
**and**  $u01$ :  $u \in \{0..1\}$   
**and**  $v01$ :  $v \in \{0..1\}$   
**and**  $w01$ :  $w \in \{0..1\}$   
**and**  $uv$ :  $u \leq v$   
**and**  $vw$ :  $v \leq w$   
**and**  $puUV$ :  $p \ u \in U \cap V$   
**and**  $pvUV$ :  $p \ v \in U \cap V$   
**and**  $pwUV$ :  $p \ w \in U \cap V$   
**shows**  $\text{homotopic-paths } W$  ( $\text{segment-loop } p \ u \ v \text{ +++ } \text{segment-loop } p \ v \ w$ )  
 $(\text{segment-loop } p \ u \ w)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{segment-loop-join-in-set}$ :

**assumes**  $p\text{-path}$ :  $\text{path } p$   
**and**  $p\text{-img}S$ :  $\text{path-image } p \subseteq S$   
**and**  $SW$ :  $S \subseteq W$   
**and**  $UVS$ :  $U \cap V \subseteq S$   
**and**  $u01$ :  $u \in \{0..1\}$   
**and**  $v01$ :  $v \in \{0..1\}$   
**and**  $w01$ :  $w \in \{0..1\}$   
**and**  $uv$ :  $u \leq v$   
**and**  $vw$ :  $v \leq w$   
**and**  $puUV$ :  $p \ u \in U \cap V$   
**and**  $pvUV$ :  $p \ v \in U \cap V$   
**and**  $pwUV$ :  $p \ w \in U \cap V$   
**shows**  $\text{homotopic-paths } S$  ( $\text{segment-loop } p \ u \ v \text{ +++ } \text{segment-loop } p \ v \ w$ )  
 $(\text{segment-loop } p \ u \ w)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{partition-loop-nil}$  [ $\text{simp}$ ]:

$\text{partition-loop } p \ [] = (\lambda-. x0)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{partition-loop-singleton}$  [ $\text{simp}$ ]:

$\text{partition-loop } p \ [t] = (\lambda-. x0)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{svk-partition-partition-loop-homotopic-segment-loop}$ :

**assumes**  $p\text{-loop}$ :  $p \in \text{loop-space } W \ x0$   
**and**  $\text{part}$ :  $\text{svk-partition } p \ (t \ \# \ ts) \ bs$   
**shows**  $\text{homotopic-paths } W$  ( $\text{partition-loop } p \ (t \ \# \ ts)$ ) ( $\text{segment-loop } p \ t \ 1$ )

*<proof>*

**lemma** *valid-partition-partition-loop-homotopic-segment-loop:*

**assumes** *p-loop:  $p \in \text{loop-space } W \ x0$*

**and** *part: valid-partition  $p \ ts \ bs$*

**shows** *homotopic-paths  $W \ (\text{partition-loop } p \ ts) \ (\text{segment-loop } p \ 0 \ 1)$*

*<proof>*

**lemma** *valid-partition-partition-loop-homotopic:*

**assumes** *p-loop:  $p \in \text{loop-space } W \ x0$*

**and** *part: valid-partition  $p \ ts \ bs$*

**shows** *homotopic-paths  $W \ (\text{partition-loop } p \ ts) \ p$*

*<proof>*

**lemma** *valid-partition-partition-loop-eq:*

**assumes** *p-loop:  $p \in \text{loop-space } W \ x0$*

**and** *part: valid-partition  $p \ ts \ bs$*

**shows** *loop-class  $W \ x0 \ (\text{partition-loop } p \ ts) = \text{loop-class } W \ x0 \ p$*

*<proof>*

**lemma** *valid-partition-decode-partition-word-eq-loop-class:*

**assumes** *p-loop:  $p \in \text{loop-space } W \ x0$*

**and** *part: valid-partition  $p \ ts \ bs$*

**shows** *svk-decode  $(\text{partition-word } p \ ts \ bs) = \text{loop-class } W \ x0 \ p$*

*<proof>*

**lemma** *subpathin-image-subset-left:*

**assumes** *t01:  $t \in \{0..1\}$*

**and** *u01:  $u \in \{0..1\}$*

**and** *v01:  $v \in \{0..1\}$*

**and** *tu:  $t \leq u$*

**and** *uv:  $u \leq v$*

**shows** *subpathin  $t \ u \ p \ \{0..1\} \subseteq \text{subpathin } t \ v \ p \ \{0..1\}$*

*<proof>*

**lemma** *subpathin-image-subset-right:*

**assumes** *t01:  $t \in \{0..1\}$*

**and** *u01:  $u \in \{0..1\}$*

**and** *v01:  $v \in \{0..1\}$*

**and** *tu:  $t \leq u$*

**and** *uv:  $u \leq v$*

**shows** *subpathin  $u \ v \ p \ \{0..1\} \subseteq \text{subpathin } t \ v \ p \ \{0..1\}$*

*<proof>*

**lemma** *subpathin-subpathin:*

*subpathin  $a \ b \ (\text{subpathin } u \ v \ p) =$*

*subpathin  $((v - u) * a) + u \ (((v - u) * b) + u) \ p$*

*<proof>*

**lemma** *segment-loop-subpathin*:  
 $segment-loop (subpathin\ u\ v\ p)\ a\ b =$   
 $segment-loop\ p\ (((v - u) * a) + u)\ (((v - u) * b) + u)$   
 $\langle proof \rangle$

**lemma** *segment-loop-mult-eq-left*:  
**assumes**  $p\text{-path}$ :  $path\ p$   
**and**  $p\text{-img}W$ :  $path\text{-image}\ p \subseteq W$   
**and**  $t01$ :  $t \in \{0..1\}$   
**and**  $u01$ :  $u \in \{0..1\}$   
**and**  $v01$ :  $v \in \{0..1\}$   
**and**  $tu$ :  $t < u$   
**and**  $uv$ :  $u < v$   
**and**  $ptUV$ :  $p\ t \in U \cap V$   
**and**  $puUV$ :  $p\ u \in U \cap V$   
**and**  $pvUV$ :  $p\ v \in U \cap V$   
**and**  $seg\text{-}tvU$ :  $subpathin\ t\ v\ p\ \{0..1\} \subseteq U$   
**shows**  $mult1 (loop\text{-}class\ U\ x0 (segment\text{-}loop\ p\ t\ u))$   
 $(loop\text{-}class\ U\ x0 (segment\text{-}loop\ p\ u\ v)) =$   
 $loop\text{-}class\ U\ x0 (segment\text{-}loop\ p\ t\ v)$   
 $\langle proof \rangle$

**lemma** *segment-loop-mult-eq-right*:  
**assumes**  $p\text{-path}$ :  $path\ p$   
**and**  $p\text{-img}W$ :  $path\text{-image}\ p \subseteq W$   
**and**  $t01$ :  $t \in \{0..1\}$   
**and**  $u01$ :  $u \in \{0..1\}$   
**and**  $v01$ :  $v \in \{0..1\}$   
**and**  $tu$ :  $t < u$   
**and**  $uv$ :  $u < v$   
**and**  $ptUV$ :  $p\ t \in U \cap V$   
**and**  $puUV$ :  $p\ u \in U \cap V$   
**and**  $pvUV$ :  $p\ v \in U \cap V$   
**and**  $seg\text{-}tvV$ :  $subpathin\ t\ v\ p\ \{0..1\} \subseteq V$   
**shows**  $mult2 (loop\text{-}class\ V\ x0 (segment\text{-}loop\ p\ t\ u))$   
 $(loop\text{-}class\ V\ x0 (segment\text{-}loop\ p\ u\ v)) =$   
 $loop\text{-}class\ V\ x0 (segment\text{-}loop\ p\ t\ v)$   
 $\langle proof \rangle$

**lemma** *segment-word-split-left-equiv*:  
**assumes**  $p\text{-loop}$ :  $p \in loop\text{-}space\ W\ x0$   
**and**  $t01$ :  $t \in \{0..1\}$   
**and**  $u01$ :  $u \in \{0..1\}$   
**and**  $v01$ :  $v \in \{0..1\}$   
**and**  $tu$ :  $t < u$   
**and**  $uv$ :  $u < v$   
**and**  $ptUV$ :  $p\ t \in U \cap V$   
**and**  $puUV$ :  $p\ u \in U \cap V$   
**and**  $pvUV$ :  $p\ v \in U \cap V$

**and**  $seg\text{-}tvU$ :  $subpathin\ t\ v\ p\ \{0..1\} \subseteq U$   
**and**  $rest\text{-}in$ :  $fpw\text{-}in\text{-}space\ G1\ G2\ rest$   
**shows**  $carrier\text{-}full\text{-}amalg\text{-}equiv\ G1\ G2\ H\ i1\ i2\ mult1\ one1\ mult2\ one2$   
 $(WordLeft\ (loop\text{-}class\ U\ x0\ (segment\text{-}loop\ p\ t\ u))$   
 $(WordLeft\ (loop\text{-}class\ U\ x0\ (segment\text{-}loop\ p\ u\ v))\ rest))$   
 $(WordLeft\ (loop\text{-}class\ U\ x0\ (segment\text{-}loop\ p\ t\ v))\ rest)$   
 $\langle proof \rangle$

**lemma**  $segment\text{-}word\text{-}split\text{-}right\text{-}equiv$ :  
**assumes**  $p\text{-}loop$ :  $p \in loop\text{-}space\ W\ x0$   
**and**  $t01$ :  $t \in \{0..1\}$   
**and**  $u01$ :  $u \in \{0..1\}$   
**and**  $v01$ :  $v \in \{0..1\}$   
**and**  $tu$ :  $t < u$   
**and**  $uv$ :  $u < v$   
**and**  $ptUV$ :  $p\ t \in U \cap V$   
**and**  $puUV$ :  $p\ u \in U \cap V$   
**and**  $pvUV$ :  $p\ v \in U \cap V$   
**and**  $seg\text{-}tvV$ :  $subpathin\ t\ v\ p\ \{0..1\} \subseteq V$   
**and**  $rest\text{-}in$ :  $fpw\text{-}in\text{-}space\ G1\ G2\ rest$   
**shows**  $carrier\text{-}full\text{-}amalg\text{-}equiv\ G1\ G2\ H\ i1\ i2\ mult1\ one1\ mult2\ one2$   
 $(WordRight\ (loop\text{-}class\ V\ x0\ (segment\text{-}loop\ p\ t\ u))$   
 $(WordRight\ (loop\text{-}class\ V\ x0\ (segment\text{-}loop\ p\ u\ v))\ rest))$   
 $(WordRight\ (loop\text{-}class\ V\ x0\ (segment\text{-}loop\ p\ t\ v))\ rest)$   
 $\langle proof \rangle$

**lemma**  $segment\text{-}word\text{-}switch$ :  
**assumes**  $p\text{-}path$ :  $path\ p$   
**and**  $p\text{-}imgW$ :  $path\text{-}image\ p \subseteq W$   
**and**  $t01$ :  $t \in \{0..1\}$   
**and**  $u01$ :  $u \in \{0..1\}$   
**and**  $tu$ :  $t < u$   
**and**  $ptUV$ :  $p\ t \in U \cap V$   
**and**  $puUV$ :  $p\ u \in U \cap V$   
**and**  $segUV$ :  $subpathin\ t\ u\ p\ \{0..1\} \subseteq U \cap V$   
**and**  $rest\text{-}in$ :  $fpw\text{-}in\text{-}space\ G1\ G2\ rest$   
**and**  $bc$ :  $b \neq c$   
**shows**  $carrier\text{-}full\text{-}amalg\text{-}equiv\ G1\ G2\ H\ i1\ i2\ mult1\ one1\ mult2\ one2$   
 $(if\ b\ then\ WordLeft\ (loop\text{-}class\ U\ x0\ (segment\text{-}loop\ p\ t\ u))\ rest$   
 $else\ WordRight\ (loop\text{-}class\ V\ x0\ (segment\text{-}loop\ p\ t\ u))\ rest)$   
 $(if\ c\ then\ WordLeft\ (loop\text{-}class\ U\ x0\ (segment\text{-}loop\ p\ t\ u))\ rest$   
 $else\ WordRight\ (loop\text{-}class\ V\ x0\ (segment\text{-}loop\ p\ t\ u))\ rest)$   
 $\langle proof \rangle$

**lemma**  $svk\text{-}partition\text{-}split\text{-}head$ :  
**assumes**  $part$ :  $svk\text{-}partition\ p\ (t\ \# \ v\ \# \ us)\ (b\ \# \ bs)$   
**and**  $tu$ :  $t < u$   
**and**  $uv$ :  $u < v$   
**and**  $puUV$ :  $p\ u \in U \cap V$

**shows** *svk-partition*  $p$  ( $t \# u \# v \# us$ ) ( $b \# b \# bs$ )  
 ⟨*proof*⟩

**lemma** *svk-partition-same-start-equiv*:

**assumes** *p-loop*:  $p \in \text{loop-space } W \ x0$

**and** *part1*: *svk-partition*  $p$  ( $t \# ts$ )  $bs$

**and** *part2*: *svk-partition*  $p$  ( $t \# us$ )  $cs$

**shows** *carrier-full-amalg-equiv*  $G1 \ G2 \ H \ i1 \ i2 \ mult1 \ one1 \ mult2 \ one2$

(*partition-word*  $p$  ( $t \# ts$ )  $bs$ )

(*partition-word*  $p$  ( $t \# us$ )  $cs$ )

⟨*proof*⟩

**lemma** *valid-partition-same-loop-partition-word-equiv*:

**assumes** *p-loop*:  $p \in \text{loop-space } W \ x0$

**and** *part1*: *valid-partition*  $p \ ts \ bs$

**and** *part2*: *valid-partition*  $p \ us \ cs$

**shows** *carrier-full-amalg-equiv*  $G1 \ G2 \ H \ i1 \ i2 \ mult1 \ one1 \ mult2 \ one2$

(*partition-word*  $p \ ts \ bs$ ) (*partition-word*  $p \ us \ cs$ )

⟨*proof*⟩

**lemma** *strip-neighbourhood*:

**fixes**  $h :: (\text{real} \times \text{real}) \Rightarrow 'a$

**assumes** *h-cont*: *continuous-map* (*top-of-set* ( $\{0..1\} \times \{0..1\}$ )) (*top-of-set*  $W$ )

$h$

**and** *a01*:  $a \in \{0..1\}$

**and** *b01*:  $b \in \{0..1\}$

**and** *y0-01*:  $y0 \in \{0..1\}$

**and** *ab*:  $a \leq b$

**and** *rowS*:  $h \text{ ' } (\{a..b\} \times \{y0\}) \subseteq S$

**and** *openS*: *open*  $S$

**shows**  $\exists N. \text{openin } (\text{top-of-set } \{0..1\}) \ N \wedge y0 \in N \wedge h \text{ ' } (\{a..b\} \times N) \subseteq S$

⟨*proof*⟩

**lemma** *svk-partition-local-neighbourhood*:

**fixes**  $h :: (\text{real} \times \text{real}) \Rightarrow 'a$

**assumes** *h-cont*: *continuous-map* (*top-of-set* ( $\{0..1\} \times \{0..1\}$ )) (*top-of-set*  $W$ )

$h$

**and** *y0-01*:  $y0 \in \{0..1\}$

**and** *part*: *svk-partition*  $(\lambda x. h \ (x, y0)) \ ts \ bs$

**shows**  $\exists N. \text{openin } (\text{top-of-set } \{0..1\}) \ N \wedge y0 \in N \wedge$

$(\forall x \in \text{set } ts. h \text{ ' } (\{x\} \times N) \subseteq U \cap V) \wedge$

$(\forall y \ z. y \leq z \longrightarrow \{y..z\} \subseteq N \longrightarrow \text{rectangle-partition } h \ y \ z \ ts \ bs)$

⟨*proof*⟩

**lemma** *rectangle-partition-svk-partition-row*:

**fixes**  $h :: (\text{real} \times \text{real}) \Rightarrow 'a$

**assumes** *part*: *rectangle-partition*  $h \ c \ d \ ts \ bs$

**and** *edgeUV*:  $\bigwedge x. x \in \text{set } ts \implies h \text{ ' } (\{x\} \times \{c..d\}) \subseteq U \cap V$

**and** *y-in*:  $y \in \{c..d\}$

**shows** *svk-partition*  $(\lambda x. h (x, y)) ts bs$   
 $\langle proof \rangle$

**lemma** *rectangle-partition-valid-partition-row*:

**fixes**  $h :: (real \times real) \Rightarrow 'a$   
**assumes** *part*: *rectangle-partition*  $h c d ts bs$   
**and** *edgeUV*:  $\bigwedge x. x \in set\ ts \implies h ' (\{x\} \times \{c..d\}) \subseteq U \cap V$   
**and** *y-in*:  $y \in \{c..d\}$   
**and** *ts-ne*:  $ts \neq []$   
**and** *hd0*:  $hd\ ts = 0$   
**shows** *valid-partition*  $(\lambda x. h (x, y)) ts bs$   
 $\langle proof \rangle$

**lemma** *homotopy-row-in-loop-space*:

**fixes**  $h :: (real \times real) \Rightarrow 'a$   
**assumes** *h-cont*: *continuous-map*  $(top-of-set (\{0..1\} \times \{0..1\})) (top-of-set W)$   
 $h$   
**and** *y01*:  $y \in \{0..1\}$   
**and** *start*:  $h (0, y) = x0$   
**and** *finish*:  $h (1, y) = x0$   
**shows**  $(\lambda x. h (x, y)) \in loop-space\ W\ x0$   
 $\langle proof \rangle$

**lemma** *bridge-word-one-equiv*:

**assumes** *rest-in*: *fpw-in-space*  $G1\ G2\ rest$   
**shows** *carrier-full-amalg-equiv*  $G1\ G2\ H\ i1\ i2\ mult1\ one1\ mult2\ one2$   
 $(bridge-word\ b\ (fundamental-group-one\ (U \cap V)\ x0)\ rest)\ rest$   
 $\langle proof \rangle$

**lemma** *partition-word-with-tail-respects*:

**assumes** *p-loop*:  $p \in loop-space\ W\ x0$   
**and** *part*: *svk-partition*  $p\ ts\ bs$   
**and** *rel*: *carrier-full-amalg-equiv*  $G1\ G2\ H\ i1\ i2\ mult1\ one1\ mult2\ one2\ r\ s$   
**shows** *carrier-full-amalg-equiv*  $G1\ G2\ H\ i1\ i2\ mult1\ one1\ mult2\ one2$   
 $(partition-word-with-tail\ p\ ts\ bs\ r)$   
 $(partition-word-with-tail\ p\ ts\ bs\ s)$   
 $\langle proof \rangle$

**lemma** *valid-partition-tail-bridge-one-equiv*:

**assumes** *p-loop*:  $p \in loop-space\ W\ x0$   
**and** *part*: *valid-partition*  $p (t \# u \# ts) (b \# bs)$   
**shows** *carrier-full-amalg-equiv*  $G1\ G2\ H\ i1\ i2\ mult1\ one1\ mult2\ one2$   
 $(partition-word-with-tail\ p (t \# u \# ts) (b \# bs))$   
 $(bridge-word (last (b \# bs)) (fundamental-group-one (U \cap V) x0) WordNil)$   
 $(partition-word\ p (t \# u \# ts) (b \# bs))$   
 $\langle proof \rangle$

**lemma** *bridge-loop-constant-one*:

**fixes**  $h :: (real \times real) \Rightarrow 'a$

**assumes**  $h\text{-cont}$ : *continuous-map* (*top-of-set* ( $\{0..1\} \times \{0..1\}$ )) (*top-of-set*  $W$ )  
 $h$   
**and**  $t01$ :  $t \in \{0..1\}$   
**and**  $c01$ :  $c \in \{0..1\}$   
**and**  $d01$ :  $d \in \{0..1\}$   
**and**  $cd$ :  $c \leq d$   
**and**  $const$ :  $\forall y \in \{c..d\}. h(t, y) = x0$   
**shows** *loop-class* ( $U \cap V$ )  $x0$  (*bridge-loop*  $h t c d$ ) = *fundamental-group-one* ( $U \cap V$ )  $x0$   
 $\langle proof \rangle$

**lemma** *rectangle-partition-partition-word-equiv*:

**fixes**  $h :: (\text{real} \times \text{real}) \Rightarrow 'a$   
**assumes**  $h\text{-cont}$ : *continuous-map* (*top-of-set* ( $\{0..1\} \times \{0..1\}$ )) (*top-of-set*  $W$ )  
 $h$   
**and**  $c01$ :  $c \in \{0..1\}$   
**and**  $d01$ :  $d \in \{0..1\}$   
**and**  $cd$ :  $c \leq d$   
**and**  $part$ : *rectangle-partition*  $h c d ts bs$   
**and**  $ts\text{-ne}$ :  $ts \neq []$   
**and**  $hd0$ :  $hd ts = 0$   
**and**  $edgeUV$ :  $\bigwedge x. x \in \text{set } ts \implies h'(\{x\} \times \{c..d\}) \subseteq U \cap V$   
**and**  $end0$ :  $\forall y \in \{c..d\}. h(0, y) = x0$   
**and**  $end1$ :  $\forall y \in \{c..d\}. h(1, y) = x0$   
**shows** *carrier-full-amalg-equiv*  $G1 G2 H i1 i2 mult1 one1 mult2 one2$   
*(partition-word* ( $\lambda x. h(x, c)$ )  $ts bs$ )  
*(partition-word* ( $\lambda x. h(x, d)$ )  $ts bs$ )  
 $\langle proof \rangle$

**lemma** *valid-partition-nearby-partition-word-equiv*:

**fixes**  $h :: (\text{real} \times \text{real}) \Rightarrow 'a$   
**assumes**  $h\text{-cont}$ : *continuous-map* (*top-of-set* ( $\{0..1\} \times \{0..1\}$ )) (*top-of-set*  $W$ )  
 $h$   
**and**  $end0$ :  $\forall y \in \{0..1\}. h(0, y) = x0$   
**and**  $end1$ :  $\forall y \in \{0..1\}. h(1, y) = x0$   
**and**  $y0\text{-}01$ :  $y0 \in \{0..1\}$   
**and**  $part0$ : *valid-partition* ( $\lambda x. h(x, y0)$ )  $ts bs$   
**shows**  $\exists e > 0. \forall z \in \{0..1\}. \text{dist } z y0 < e \longrightarrow$   
*valid-partition* ( $\lambda x. h(x, z)$ )  $ts bs \wedge$   
*carrier-full-amalg-equiv*  $G1 G2 H i1 i2 mult1 one1 mult2 one2$   
*(partition-word* ( $\lambda x. h(x, y0)$ )  $ts bs$ )  
*(partition-word* ( $\lambda x. h(x, z)$ )  $ts bs$ )  
 $\langle proof \rangle$

**definition** *some-valid-partition* ::  $(\text{real} \Rightarrow 'a) \Rightarrow \text{real list} \times \text{bool list}$  **where**

*some-valid-partition*  $p =$   
 $(\text{SOME } tb. \text{case } tb \text{ of } (ts, bs) \Rightarrow \text{valid-partition } p \text{ } ts \text{ } bs)$

**lemma** *some-valid-partition-spec*:

**assumes** *p-loop*:  $p \in \text{loop-space } W \ x0$   
**shows** *valid-partition p*  
 (*fst (some-valid-partition p)*)  
 (*snd (some-valid-partition p)*)  
 ⟨*proof*⟩

### 15.3 Homotopy invariance of the partition encoding

The central technical step is that different valid partitions of homotopic loops produce equivalent words in the carrier-side amalgamated free product. The proof uses a rectangular decomposition of a homotopy and keeps the word encoding stable while moving from one boundary loop to the other.

**lemma** *valid-partition-homotopic-partition-word-equiv*:  
**assumes** *p-loop*:  $p \in \text{loop-space } W \ x0$   
**and** *q-loop*:  $q \in \text{loop-space } W \ x0$   
**and** *pq*: *homotopic-paths*  $W \ p \ q$   
**and** *p-part*: *valid-partition*  $p \ ts \ bs$   
**and** *q-part*: *valid-partition*  $q \ us \ cs$   
**shows** *carrier-full-amalg-equiv*  $G1 \ G2 \ H \ i1 \ i2 \ mult1 \ one1 \ mult2 \ one2$   
 (*partition-word p ts bs*) (*partition-word q us cs*)  
 ⟨*proof*⟩

**lemma** *valid-partition-loop-class-partition-word-equiv*:  
**assumes** *p-loop*:  $p \in \text{loop-space } W \ x0$   
**and** *q-loop*:  $q \in \text{loop-space } W \ x0$   
**and** *eq*: *loop-class*  $W \ x0 \ p = \text{loop-class } W \ x0 \ q$   
**and** *p-part*: *valid-partition*  $p \ ts \ bs$   
**and** *q-part*: *valid-partition*  $q \ us \ cs$   
**shows** *carrier-full-amalg-equiv*  $G1 \ G2 \ H \ i1 \ i2 \ mult1 \ one1 \ mult2 \ one2$   
 (*partition-word p ts bs*) (*partition-word q us cs*)  
 ⟨*proof*⟩

**lemma** *svk-decode-word-loop*:  
**assumes** *w-in*: *fpw-in-space*  $G1 \ G2 \ w$   
**shows** *svk-decode*  $w = \text{loop-class } W \ x0 \ (\text{word-loop } w)$   
 ⟨*proof*⟩

### 15.4 Encoding, decoding, and the final bijection

With existence, refinement invariance, and homotopy invariance in place, the encode/decode pair can now be defined directly on loop classes. The remaining lemmas verify the round-trip laws required by the abstract interface and turn them into the classical Seifert–van Kampen bijection.

**definition** *svk-encode* ::  
 (*real*  $\Rightarrow$  *'a*) *set*  $\Rightarrow$  ((*real*  $\Rightarrow$  *'a*) *set*, (*real*  $\Rightarrow$  *'a*) *set*) *free-product-word*  
**where**  
*svk-encode*  $A =$   
 (*let*  $p = \text{some-loop } W \ x0 \ A;$

$tb = \text{some-valid-partition } p$   
*in case*  $tb$  of  $(ts, bs) \Rightarrow \text{partition-word } p \ ts \ bs$ )

**lemma** *svk-encode-in-space*:

**assumes** *A-in*:  $A \in \text{fundamental-group-space } W \ x0$

**shows** *fpw-in-space*  $G1 \ G2$  (*svk-encode*  $A$ )

*<proof>*

**lemma** *svk-decode-encode*:

**assumes** *A-in*:  $A \in \text{fundamental-group-space } W \ x0$

**shows** *svk-decode* (*svk-encode*  $A$ ) =  $A$

*<proof>*

**lemma** *svk-encode-decode*:

**assumes** *w-in*: *fpw-in-space*  $G1 \ G2 \ w$

**shows** *carrier-full-amalg-equiv*  $G1 \ G2 \ H \ i1 \ i2 \ \text{mult1} \ \text{one1} \ \text{mult2} \ \text{one2}$

(*svk-encode* (*svk-decode*  $w$ ))  $w$

*<proof>*

**lemma** *svk-decode-surjective*:

**assumes** *A-in*:  $A \in \text{fundamental-group-space } W \ x0$

**shows**  $\exists w. \text{fpw-in-space } G1 \ G2 \ w \wedge \text{svk-decode } w = A$

*<proof>*

**theorem** *svk-decode-image*:

*svk-decode* ‘  $\{w. \text{fpw-in-space } G1 \ G2 \ w\} = \text{fundamental-group-space } W \ x0$

*<proof>*

**definition** *classical-svk-quotient-map* ::

$(\text{real} \Rightarrow 'a) \ \text{set} \Rightarrow$

$((\text{real} \Rightarrow 'a) \ \text{set}, (\text{real} \Rightarrow 'a) \ \text{set}) \ \text{free-product-word} \ \text{set}$

**where**

*classical-svk-quotient-map*  $A =$

*carrier-full-amalg-class*  $G1 \ G2 \ H \ i1 \ i2 \ \text{mult1} \ \text{one1} \ \text{mult2} \ \text{one2}$  (*svk-encode*  $A$ )

**lemma** *classical-svk-quotient-map-in-space*:

**assumes** *A-in*:  $A \in \text{fundamental-group-space } W \ x0$

**shows** *classical-svk-quotient-map*  $A \in$

*carrier-full-amalgamated-free-product-space*  $G1 \ G2 \ H \ i1 \ i2 \ \text{mult1} \ \text{one1} \ \text{mult2} \ \text{one2}$

*<proof>*

**lemma** *classical-svk-quotient-map-injective*:

**assumes** *AB*: *classical-svk-quotient-map*  $A = \text{classical-svk-quotient-map } B$

**and** *A-in*:  $A \in \text{fundamental-group-space } W \ x0$

**and** *B-in*:  $B \in \text{fundamental-group-space } W \ x0$

**shows**  $A = B$

*<proof>*

**lemma** *classical-svk-quotient-map-surjective:*

**assumes** *Q-in:*

*Q ∈ carrier-full-amalgamated-free-product-space G1 G2 H i1 i2 mult1 one1 mult2 one2*

**shows**  $\exists A \in \text{fundamental-group-space } W \ x0. \text{ classical-svk-quotient-map } A = Q$   
*<proof>*

At this point the encoding and decoding maps satisfy the abstract round-trip laws from the interface theory. The final theorem therefore presents the classical Seifert–van Kampen statement as a bijection between the fundamental group of  $U \cup V$  at  $x0$  and the carrier-based amalgamated free product assembled from  $U$ ,  $V$ , and  $U \cap V$ .

**theorem** *classical-seifert-van-kampen-bij-betw:*

*bij-betw classical-svk-quotient-map (fundamental-group-space W x0)*

*(carrier-full-amalgamated-free-product-space G1 G2 H i1 i2 mult1 one1 mult2 one2)*

*<proof>*

**end**

**end**

**theory** *Binary-Sum-Topology*

**imports** *HOL–Analysis.Abstract-Topology*

**begin**

## 16 Binary coproduct topology

Topological pushouts are constructed by first equipping the disjoint sum with its coproduct topology and only then quotienting by the glue relation. This short theory records that coproduct topology together with the basic continuity lemmas used by the later pushout arguments.

**definition** *binary-sum-topology* :: *'a topology => 'b topology => ('a + 'b) topology*  
**where**

*binary-sum-topology X Y =*

*topology ( $\lambda U.$*

*U ⊆ Inl ' topspace X ∪ Inr ' topspace Y ∧*

*openin X {x. Inl x ∈ U} ∧*

*openin Y {y. Inr y ∈ U})*

**lemma** *is-binary-sum-topology:*

*istopology ( $\lambda U.$*

*U ⊆ Inl ' topspace X ∪ Inr ' topspace Y ∧*

*openin X {x. Inl x ∈ U} ∧*

*openin Y {y. Inr y ∈ U})*

*<proof>*

**lemma** *openin-binary-sum-topology:*

```

openin (binary-sum-topology X Y) U  $\longleftrightarrow$ 
  U  $\subseteq$  Inl ' topspace X  $\cup$  Inr ' topspace Y  $\wedge$ 
  openin X {x. Inl x  $\in$  U}  $\wedge$ 
  openin Y {y. Inr y  $\in$  U}
⟨proof⟩

lemma topspace-binary-sum-topology [simp]:
  topspace (binary-sum-topology X Y) = Inl ' topspace X  $\cup$  Inr ' topspace Y
⟨proof⟩

lemma subset-topspace-binary-sum-inl:
  (Inl :: 'a  $\Rightarrow$  'a + 'b) ' topspace X  $\subseteq$  topspace (binary-sum-topology X Y)
⟨proof⟩

lemma subset-topspace-binary-sum-inr:
  (Inr :: 'b  $\Rightarrow$  'a + 'b) ' topspace Y  $\subseteq$  topspace (binary-sum-topology X Y)
⟨proof⟩

lemma continuous-map-binary-sum-inl:
  continuous-map X (binary-sum-topology X Y) (Inl :: 'a  $\Rightarrow$  'a + 'b)
⟨proof⟩

lemma continuous-map-binary-sum-inr:
  continuous-map Y (binary-sum-topology X Y) (Inr :: 'b  $\Rightarrow$  'a + 'b)
⟨proof⟩

lemma open-map-binary-sum-inl:
  open-map X (binary-sum-topology X Y) (Inl :: 'a  $\Rightarrow$  'a + 'b)
⟨proof⟩

lemma open-map-binary-sum-inr:
  open-map Y (binary-sum-topology X Y) (Inr :: 'b  $\Rightarrow$  'a + 'b)
⟨proof⟩

lemma continuous-map-from-binary-sum-topology:
  assumes f: continuous-map X Z f
  and g: continuous-map Y Z g
  shows continuous-map (binary-sum-topology X Y) Z (case-sum f g)
⟨proof⟩

end
theory Topological-Pushout-Scaffold
  imports Binary-Sum-Topology Pushout-Scaffold
begin

```

## 17 Topological pushouts

The quotient-level pushout relation from *Pushout-Scaffold* becomes a genuine topological pushout here. The theory defines the quotient topology on the glued disjoint sum and proves the universal maps from the two summands into that topological pushout.

**definition** *pushout-topospace* ::

$'a \text{ topology} \Rightarrow 'b \text{ topology} \Rightarrow ('c \Rightarrow 'a) \Rightarrow ('c \Rightarrow 'b) \Rightarrow ('a + 'b) \text{ set set}$

**where**

$\text{pushout-topospace } X \ Y \ f \ g = \text{pushout-class } f \ g \text{ ' } \text{topospace } (\text{binary-sum-topology } X \ Y)$

**definition** *pushout-topology* ::

$'a \text{ topology} \Rightarrow 'b \text{ topology} \Rightarrow ('c \Rightarrow 'a) \Rightarrow ('c \Rightarrow 'b) \Rightarrow (('a + 'b) \text{ set}) \text{ topology}$

**where**

$\text{pushout-topology } X \ Y \ f \ g = \text{topology } (\lambda U.$

$U \subseteq \text{pushout-topospace } X \ Y \ f \ g \wedge$

$\text{openin } (\text{binary-sum-topology } X \ Y)$

$\{z \in \text{topospace } (\text{binary-sum-topology } X \ Y). \text{pushout-class } f \ g \ z \in U\})$

**lemma** *pushout-topospace-alt*:

$\text{pushout-topospace } X \ Y \ f \ g = \text{pushout-inl } f \ g \text{ ' } \text{topospace } X \cup \text{pushout-inr } f \ g \text{ ' } \text{topospace } Y$

$\langle \text{proof} \rangle$

**lemma** *pushout-topospace-subset-pushout-space*:

$\text{pushout-topospace } X \ Y \ f \ g \subseteq \text{pushout-space } f \ g$

$\langle \text{proof} \rangle$

**lemma** *is-pushout-topology*:

$\text{istopology } (\lambda U.$

$U \subseteq \text{pushout-topospace } X \ Y \ f \ g \wedge$

$\text{openin } (\text{binary-sum-topology } X \ Y)$

$\{z \in \text{topospace } (\text{binary-sum-topology } X \ Y). \text{pushout-class } f \ g \ z \in U\})$

$\langle \text{proof} \rangle$

**lemma** *openin-pushout-topology*:

$\text{openin } (\text{pushout-topology } X \ Y \ f \ g) \ U \longleftrightarrow$

$U \subseteq \text{pushout-topospace } X \ Y \ f \ g \wedge$

$\text{openin } (\text{binary-sum-topology } X \ Y)$

$\{z \in \text{topospace } (\text{binary-sum-topology } X \ Y). \text{pushout-class } f \ g \ z \in U\}$

$\langle \text{proof} \rangle$

**lemma** *topospace-pushout-topology* [*simp*]:

$\text{topospace } (\text{pushout-topology } X \ Y \ f \ g) = \text{pushout-topospace } X \ Y \ f \ g$

$\langle \text{proof} \rangle$

**lemma** *quotient-map-pushout-class*:

*quotient-map* (*binary-sum-topology*  $X$   $Y$ ) (*pushout-topology*  $X$   $Y$   $f$   $g$ )  
(*pushout-class*  $f$   $g$ )  
{*proof*}

**lemma** *continuous-map-pushout-class*:

*continuous-map* (*binary-sum-topology*  $X$   $Y$ ) (*pushout-topology*  $X$   $Y$   $f$   $g$ )  
(*pushout-class*  $f$   $g$ )  
{*proof*}

**lemma** *pushout-inl-in-topospace*:

**assumes**  $a \in \text{topspace } X$   
**shows** *pushout-inl*  $f$   $g$   $a \in \text{pushout-topospace } X$   $Y$   $f$   $g$   
{*proof*}

**lemma** *pushout-inr-in-topospace*:

**assumes**  $b \in \text{topspace } Y$   
**shows** *pushout-inr*  $f$   $g$   $b \in \text{pushout-topospace } X$   $Y$   $f$   $g$   
{*proof*}

**lemma** *continuous-map-pushout-inl* [*continuous-intros*]:

*continuous-map*  $X$  (*pushout-topology*  $X$   $Y$   $f$   $g$ ) (*pushout-inl*  $f$   $g$ )  
{*proof*}

**lemma** *continuous-map-pushout-inr* [*continuous-intros*]:

*continuous-map*  $Y$  (*pushout-topology*  $X$   $Y$   $f$   $g$ ) (*pushout-inr*  $f$   $g$ )  
{*proof*}

**lemma** *pushout-rec-pushout-class*:

**assumes** *compat*: *pushout-case-compatible*  $f$   $g$  *left* *right*  
**and** *x-in*:  $x \in \text{topspace } (\text{binary-sum-topology } X$   $Y)$   
**shows** *pushout-rec*  $f$   $g$  *left* *right* (*pushout-class*  $f$   $g$   $x$ ) = *case-sum* *left* *right*  $x$   
{*proof*}

**lemma** *continuous-map-pushout-rec*:

**assumes** *compat*: *pushout-case-compatible*  $f$   $g$  *left* *right*  
**and** *left-cont*: *continuous-map*  $X$   $Z$  *left*  
**and** *right-cont*: *continuous-map*  $Y$   $Z$  *right*  
**shows** *continuous-map* (*pushout-topology*  $X$   $Y$   $f$   $g$ )  $Z$  (*pushout-rec*  $f$   $g$  *left* *right*)  
{*proof*}

**lemma** *pushout-rec-universal*:

**assumes** *compat*: *pushout-case-compatible*  $f$   $g$  *left* *right*  
**and** *left-cont*: *continuous-map*  $X$   $Z$  *left*  
**and** *right-cont*: *continuous-map*  $Y$   $Z$  *right*  
**shows** *continuous-map* (*pushout-topology*  $X$   $Y$   $f$   $g$ )  $Z$  (*pushout-rec*  $f$   $g$  *left* *right*)  
**and**  $\bigwedge a$ . *pushout-rec*  $f$   $g$  *left* *right* (*pushout-inl*  $f$   $g$   $a$ ) = *left*  $a$   
**and**  $\bigwedge b$ . *pushout-rec*  $f$   $g$  *left* *right* (*pushout-inr*  $f$   $g$   $b$ ) = *right*  $b$   
{*proof*}

```

end
theory Topological-Seifert-Van-Kampen
  imports
    Carrier-Amalgamated-Free-Product-Eval
    Explicit-Fundamental-Group-Scaffold
    Topological-Pushout-Scaffold
    Seifert-Van-Kampen-Scaffold
begin

```

## 18 Concrete decode data for topological pushouts

Once the pushout topology and the carrier-side amalgamation evaluator are available, the remaining task is to package a concrete decoding map back into the fundamental group of the pushout. This theory shows how such decode data yields the abstract carrier-level Seifert–van Kampen bijection in the topological pushout setting.

**lemma** *loopin-image-compose* [*simp*]:  
 $\text{loopin-image } g (\text{loopin-image } f p) = \text{loopin-image } (g \circ f) p$   
*<proof>*

**lemma** *fundamental-groupin-map-eq*:  
**assumes** *A-in*:  $A \in \text{fundamental-groupin-space } X x$   
**and** *f-cont*: *continuous-map*  $X Y f$   
**and** *g-cont*: *continuous-map*  $X Y g$   
**and** *fx*:  $f x = y$   
**and** *gx*:  $g x = y$   
**and** *fg*:  $\bigwedge u. f u = g u$   
**shows**  $\text{fundamental-groupin-map } X x Y y f A = \text{fundamental-groupin-map } X x Y y g A$   
*<proof>*

**lemma** *fundamental-groupin-map-compose*:  
**assumes** *A-in*:  $A \in \text{fundamental-groupin-space } X x$   
**and** *f-cont*: *continuous-map*  $X Y f$   
**and** *g-cont*: *continuous-map*  $Y Z g$   
**and** *fx*:  $f x = y$   
**and** *gy*:  $g y = z$   
**shows**  $\text{fundamental-groupin-map } Y y Z z g (\text{fundamental-groupin-map } X x Y y f A) =$   
 $\text{fundamental-groupin-map } X x Z z (g \circ f) A$   
*<proof>*

**locale** *topological-svk-setup* =  
**fixes**  $X :: 'a \text{ topology}$   
**and**  $Y :: 'b \text{ topology}$   
**and**  $Z :: 'c \text{ topology}$   
**and**  $f :: 'c \Rightarrow 'a$

**and**  $g :: 'c \Rightarrow 'b$   
**and**  $z0 :: 'c$   
**assumes**  $z0\text{-in}: z0 \in \text{topspace } Z$   
**and**  $f\text{-cont}: \text{continuous-map } Z \ X \ f$   
**and**  $g\text{-cont}: \text{continuous-map } Z \ Y \ g$   
**begin**

**abbreviation**  $x0$  **where**  $x0 \equiv f \ z0$   
**abbreviation**  $y0$  **where**  $y0 \equiv g \ z0$   
**abbreviation**  $P$  **where**  $P \equiv \text{pushout-topology } X \ Y \ f \ g$   
**abbreviation**  $p0$  **where**  $p0 \equiv \text{pushout-inl } f \ g \ x0$

**abbreviation**  $G1$  **where**  $G1 \equiv \text{fundamental-groupin-space } X \ x0$   
**abbreviation**  $G2$  **where**  $G2 \equiv \text{fundamental-groupin-space } Y \ y0$   
**abbreviation**  $H$  **where**  $H \equiv \text{fundamental-groupin-space } Z \ z0$   
**abbreviation**  $\text{mult1}$  **where**  $\text{mult1} \equiv \text{fundamental-groupin-mult } X \ x0$   
**abbreviation**  $\text{one1}$  **where**  $\text{one1} \equiv \text{fundamental-groupin-one } X \ x0$   
**abbreviation**  $\text{mult2}$  **where**  $\text{mult2} \equiv \text{fundamental-groupin-mult } Y \ y0$   
**abbreviation**  $\text{one2}$  **where**  $\text{one2} \equiv \text{fundamental-groupin-one } Y \ y0$   
**abbreviation**  $\text{multP}$  **where**  $\text{multP} \equiv \text{fundamental-groupin-mult } P \ p0$   
**abbreviation**  $\text{oneP}$  **where**  $\text{oneP} \equiv \text{fundamental-groupin-one } P \ p0$

**abbreviation**  $i1$  **where**  $i1 \equiv \text{fundamental-groupin-map } Z \ z0 \ X \ x0 \ f$   
**abbreviation**  $i2$  **where**  $i2 \equiv \text{fundamental-groupin-map } Z \ z0 \ Y \ y0 \ g$   
**abbreviation**  $j1$  **where**  $j1 \equiv \text{fundamental-groupin-map } X \ x0 \ P \ p0 \ (\text{pushout-inl } f \ g)$   
**abbreviation**  $j2$  **where**  $j2 \equiv \text{fundamental-groupin-map } Y \ y0 \ P \ p0 \ (\text{pushout-inr } f \ g)$

**lemma**  $x0\text{-in}$  [*simp*]:  $x0 \in \text{topspace } X$   
*<proof>*

**lemma**  $y0\text{-in}$  [*simp*]:  $y0 \in \text{topspace } Y$   
*<proof>*

**lemma**  $\text{pushout-basepoint-eq}$  [*simp*]:  
 $\text{pushout-inr } f \ g \ y0 = p0$   
*<proof>*

**lemma**  $\text{pushout-basepoint-in}$  [*simp*]:  
 $p0 \in \text{topspace } P$   
*<proof>*

**lemma**  $\text{pushout-fundamental-group-maps-agree}$ :  
**assumes**  $h\text{-in}: h \in H$   
**shows**  $j1 \ (i1 \ h) = j2 \ (i2 \ h)$   
*<proof>*

**lemma**  $i1\text{-in-}G1$ :

**assumes**  $h \in H$   
**shows**  $i1 \ h \in G1$   
 $\langle proof \rangle$

**lemma** *i2-in-G2*:  
**assumes**  $h \in H$   
**shows**  $i2 \ h \in G2$   
 $\langle proof \rangle$

**lemma** *decode-locale*:  
*carrier-full-amalg-word-eval*  
 $G1 \ mult1 \ one1 \ (fundamental-groupin-inv \ X \ x0)$   
 $G2 \ mult2 \ one2 \ (fundamental-groupin-inv \ Y \ y0)$   
 $H \ i1 \ i2$   
 $(fundamental-groupin-space \ P \ p0) \ multP \ oneP \ (fundamental-groupin-inv \ P \ p0)$   
 $j1 \ j2$   
 $\langle proof \rangle$

**interpretation** *decode*:  
*carrier-full-amalg-word-eval*  
 $G1 \ mult1 \ one1 \ fundamental-groupin-inv \ X \ x0$   
 $G2 \ mult2 \ one2 \ fundamental-groupin-inv \ Y \ y0$   
 $H \ i1 \ i2$   
 $fundamental-groupin-space \ P \ p0 \ multP \ oneP \ fundamental-groupin-inv \ P \ p0$   
 $j1 \ j2$   
 $\langle proof \rangle$

**abbreviation** *svk-word-eval* **where**  $svk-word-eval \equiv decode.carrier-full-amalg-eval$   
**abbreviation** *svk-decode* **where**  $svk-decode \equiv decode.carrier-full-amalg-decode$

**lemma** *svk-decode-in-space*:  
 $svk-decode \ w \in fundamental-groupin-space \ P \ p0$   
 $\langle proof \rangle$

**lemma** *svk-decode-respects*:  
**assumes** *carrier-full-amalg-equiv*  $G1 \ G2 \ H \ i1 \ i2 \ mult1 \ one1 \ mult2 \ one2 \ u \ v$   
**shows**  $svk-decode \ u = svk-decode \ v$   
 $\langle proof \rangle$

**lemma** *svk-decode-eq-eval*:  
**assumes** *fpw-in-space*  $G1 \ G2 \ w$   
**shows**  $svk-decode \ w = svk-word-eval \ w$   
 $\langle proof \rangle$

**end**

**locale** *topological-svk-open-cover* =  
*topological-svk-setup*  $X \ Y \ Z \ f \ g \ z0$   
**for**  $X :: 'a \ topology$

```

    and Y :: 'b topology
    and Z :: 'c topology
    and f :: 'c ⇒ 'a
    and g :: 'c ⇒ 'b
    and z0 :: 'c +
  assumes left-open: openin P (pushout-inl f g ' topspace X)
    and right-open: openin P (pushout-inr f g ' topspace Y)
begin

lemma loopin-subdivision-by-summands:
  assumes p-loop: p ∈ loopin-space P p0
  shows ∃ n::nat. 0 < n ∧
    (∀ i < n.
      subpathin (real i / real n) (real (Suc i) / real n) p ' {0..1}
        ⊆ pushout-inl f g ' topspace X ∨
      subpathin (real i / real n) (real (Suc i) / real n) p ' {0..1}
        ⊆ pushout-inr f g ' topspace Y)
⟨proof⟩

end

locale topological-svk =
  topological-svk-setup X Y Z f g z0
  for X :: 'a topology
    and Y :: 'b topology
    and Z :: 'c topology
    and f :: 'c ⇒ 'a
    and g :: 'c ⇒ 'b
    and z0 :: 'c +
  fixes encode
  assumes encode-in-space: fpw-in-space G1 G2 (encode A)
    and decode-encode: svk-decode (encode A) = A
    and encode-decode:
      carrier-full-amalg-equiv G1 G2 H i1 i2 mult1 one1 mult2 one2 (encode
(svk-decode w)) w
begin

lemma svk-locale:
  carrier-svk-encode-decode-interface
  G1 G2 H i1 i2 mult1 one1 mult2 one2 encode svk-decode
⟨proof⟩

interpretation svk: carrier-svk-encode-decode-interface
  G1 G2 H i1 i2 mult1 one1 mult2 one2 encode svk-decode
⟨proof⟩

```

The theorem below isolates the topological conclusion once an encoding map has been supplied and the usual round-trip laws have been verified. The classical open-union theorem later instantiates this interface by constructing

that encoding from loop partitions subordinate to  $U$  and  $V$ .

**theorem** *topological-seifert-van-kampen-bij-betw:*

*bij-betw svk.carrier-svk-quotient-map UNIV*  
*(carrier-full-amalgamated-free-product-space G1 G2 H i1 i2 mult1 one1 mult2*  
*one2)*  
*<proof>*

**end**

**end**

**theory** *Seifert-Van-Kampen*

**imports**

*Classical-Seifert-Van-Kampen*

*Topological-Seifert-Van-Kampen*

**begin**

## 19 Top-level entry point

This session formalizes a quotient-oriented version of the classical Seifert–van Kampen theorem in Isabelle/HOL. It develops reusable infrastructure for pushout glue relations, free-product words, carrier-based amalgamated free products, and explicit path/homotopy quotients. The unconditional classical open-union theorem is proved in *Classical-Seifert-Van-Kampen*.

Auxiliary theories package the abstract encode/decode interface and the pushout-level constructions used internally by the proof. The headline theorem exported by this entry is the classical result for open unions.

**end**

## References

- [1] R. Brown. *Topology and Groupoids*. BookSurge LLC, Charleston, SC, 3 edition, 2006.
- [2] A. Hatcher. *Algebraic Topology*. Cambridge University Press, Cambridge, 2002.
- [3] H. Seifert. Konstruktion dreidimensionaler geschlossener räume. *Berichte über die Verhandlungen der Sächsischen Akademie der Wissenschaften zu Leipzig, Mathematisch-Physische Klasse*, 83:26–66, 1931.
- [4] E. R. van Kampen. On the connection between the fundamental groups of some related spaces. *American Journal of Mathematics*, 55:261–267, 1933. JSTOR stable URL: <https://www.jstor.org/stable/51000091>.