

Knot Theory

T. V. H. Prathamesh

October 13, 2025

Abstract

This work contains a formalization of some topics in knot theory. The concepts that were formalized include definitions of tangles, links, framed links and link/tangle equivalence. The formalization is based on a formulation of links in terms of tangles. We further construct and prove the invariance of the Bracket polynomial. Bracket polynomial is an invariant of framed links closely linked to the Jones polynomial. This is perhaps the first attempt to formalize any aspect of knot theory in an interactive proof assistant.

For further reference, one can refer to the paper "Formalising Knot Theory in Isabelle/HOL" in Interactive Theorem Proving, 6th International Conference, ITP 2015, Nanjing, China, August 24-27, 2015, Proceedings.

Contents

1 Preliminaries: Definitions of tangles and links	1
2 Tangles: Definition as a type and basic functions on tangles	6
3 Tangle_Algebra: Tensor product of tangles and its properties	8
4 Definition of tensor product of walls	8
5 Properties of tensor product of tangles	9
6 Tangle_Moves: Defining moves on tangles	12
7 Link_Algebra: Defining equivalence of tangles and links	17
8 Showing equivalence of links: An example	19
9 Kauffman Matrix and Kauffman Bracket- Definitions and Properties	20

10 Rational Functions	20
11 Kauffman matrices	22
12 Computations: This section can be skipped	30
13 Tangle moves and Kauffman bracket	33
14 Kauffman_Invariance: Proving the invariance of Kauffman Bracket	37

1 Preliminaries: Definitions of tangles and links

```
theory Preliminaries
imports Main
begin
```

This theory contains the definition of a link. A link is defined as link diagrams upto equivalence moves. Link diagrams are defined in terms of the constituent tangles

each block is a horizontal block built by putting basic link bricks next to each other. (1) vert is the straight line (2) cup is the up facing cup (3) cap is the bottom facing (4) over is the positive cross (5) under is the negative cross

```
datatype brick = vert
              | cup
              | cap
              | over
              | under
```

block is obtained by putting bricks next to each other

```
type-synonym block = brick list
```

wall are link diagrams obtained by placing a horizontal blocks a top each other

```
datatype wall = basic block
              | prod block wall (infixr <*> 66)
```

Concatenate gives us the block obtained by putting two blocks next to each other

```
primrec concatenate :: block => block => block (infixr <⊗> 65) where
concatenates-Nil: [] ⊗ ys = ys |
concatenates-Cons: ((x#xs)⊗ys) = x#(xs⊗ys)
```

```
lemma empty-concatenate: xs ⊗ Nil = xs
<proof>
```

Associativity properties of Conscatenation

lemma *leftright-associativity*: $(x \otimes y) \otimes z = x \otimes (y \otimes z)$
 $\langle proof \rangle$

lemma *left-associativity*: $(x \otimes y) \otimes z = x \otimes y \otimes z$
 $\langle proof \rangle$

lemma *right-associativity*: $x \otimes (y \otimes z) = x \otimes y \otimes z$
 $\langle proof \rangle$

Compose gives us the wall obtained by putting a wall above another, perhaps in an invalid way.

primrec *compose* :: *wall* $\Rightarrow$ *wall* $\Rightarrow$ *wall* (**infixr** $\circ$ 66) **where**
compose-Nil: $(basic\ x) \circ ys = prod\ x\ ys \mid$
compose-Cons: $((prod\ x\ xs) \circ ys) = prod\ x\ (xs \circ ys)$

Associativity properties of composition

lemma *compose-leftassociativity*: $((x::wall) \circ y) \circ z = (x \circ y) \circ z$
 $\langle proof \rangle$

lemma *compose-rightassociativity*: $(x::wall) \circ (y \circ z) = (x \circ y) \circ z$
 $\langle proof \rangle$

block-length of a block is the number of bricks in a given block

primrec *block-length*::*block* $\Rightarrow$ *nat*
where
block-length [] = 0
block-length (Cons *x y*) = 1 + (*block-length y*)

primrec *domain*::*brick* $\Rightarrow$ *int*
where
domain vert = 1
domain cup = 0
domain cap = 2
domain over = 2
domain under = 2

lemma *domain-non-negative*: $\forall x. (domain\ x) \geq 0$
 $\langle proof \rangle$

primrec *codomain*::*brick* $\Rightarrow$ *int*

where

codomain vert = 1 |
codomain cup = 2 |
codomain cap = 0 |
codomain over = 2 |
codomain under = 2

primrec *domain-block::block* $\Rightarrow$ *int*

where

domain-block [] = 0
| *domain-block* (Cons *x y*) = (*domain x* + (*domain-block y*))

lemma *domain-block-non-negative:domain-block xs* $\geq$ 0

<proof>

primrec *codomain-block::block* $\Rightarrow$ *int*

where

codomain-block [] = 0
| *codomain-block* (Cons *x y*) = (*codomain x* + (*codomain-block y*))

primrec *domain-wall:: wall* $\Rightarrow$ *int* **where**

domain-wall (*basic x*) = *domain-block x*
| *domain-wall* (*x*ys*) = *domain-block x*

fun *codomain-wall:: wall* $\Rightarrow$ *int* **where**

codomain-wall (*basic x*) = *codomain-block x*
| *codomain-wall* (*x*ys*) = *codomain-wall ys*

lemma *domain-wall-compose: domain-wall (xs* $\circ$ *ys)* = *domain-wall xs*

<proof>

lemma *codomain-wall-compose: codomain-wall (xs* $\circ$ *ys)* = *codomain-wall ys*

<proof>

this lemma tells us the number of incoming and outgoing strands of a composition of two wall

absolute value

definition *abs::int* $\Rightarrow$ *int* **where**

$abs\ x \equiv if\ (x \geq 0)\ then\ x\ else\ (0 - x)$

theorems about abs

lemma *abs-zero*: **assumes** $abs\ x = 0$ **shows** $x = 0$
<proof>

lemma *abs-zero-equality*: **assumes** $abs\ (x - y) = 0$ **shows** $x = y$
<proof>

lemma *abs-non-negative*: $abs\ x \geq 0$
<proof>

lemma *abs-non-negative-sum*: **assumes** $abs\ x + abs\ y = 0$
shows $abs\ x = 0$ **and** $abs\ y = 0$
<proof>

The following lemmas tell us that the number of incoming and outgoing strands of every brick is a non negative integer

lemma *domain-nonnegative*: $(domain\ x) \geq 0$
<proof>

lemma *codomain-nonnegative*: $(codomain\ x) \geq 0$
<proof>

The following lemmas tell us that the number of incoming and outgoing strands of every block is a non negative integer

lemma *domain-block-nonnegative*: $domain\text{-}block\ x \geq 0$
<proof>

lemma *codomain-block-nonnegative*: $(codomain\text{-}block\ x) \geq 0$
<proof>

The following lemmas tell us that if a block is appended to a block with incoming strands, then the resultant block has incoming strands

lemma *domain-positive*: $((domain\text{-}block\ (x \# Nil)) > 0) \vee ((domain\text{-}block\ y) > 0)$

$\implies (domain\text{-}block\ (x \# y) > 0)$
<proof>

lemma *domain-additive*: $(domain\text{-}block\ (x \otimes y)) = (domain\text{-}block\ x) + (domain\text{-}block\ y)$
<proof>

lemma *codomain-additive*: $(codomain\text{-}block\ (x \otimes y)) = (codomain\text{-}block\ x) + (codomain\text{-}block\ y)$

$\langle proof \rangle$

lemma *domain-zero-sum*: **assumes** $(domain\text{-}block\ x) + (domain\text{-}block\ y) = 0$
shows $domain\text{-}block\ x = 0$ **and** $domain\text{-}block\ y = 0$
 $\langle proof \rangle$

lemma *domain-block-positive*: **fixes** or **assumes** $domain\text{-}block\ y > 0$ or $domain\text{-}block\ y > 0$
shows $(domain\text{-}block\ (x \otimes y)) > 0$
 $\langle proof \rangle$

lemma *codomain-block-positive*: **fixes** or **assumes** $codomain\text{-}block\ y > 0$ or $codomain\text{-}block\ y > 0$
shows $(codomain\text{-}block\ (x \otimes y)) > 0$
 $\langle proof \rangle$

We prove that if the first count of a block is zero, then it is composed of cups and empty bricks. In order to do that we define the functions *brick-is-cup* and *is-cup* which check if a given block is composed of cups or if the blocks are composed of blocks

primrec *brick-is-cup*:: $brick \Rightarrow bool$
where
brick-is-cup *vert* = *False* |
brick-is-cup *cup* = *True* |
brick-is-cup *cap* = *False* |
brick-is-cup *over* = *False* |
brick-is-cup *under* = *False*

primrec *is-cup*:: $block \Rightarrow bool$
where
is-cup [] = *True* |
is-cup $(x \# y)$ = (if $(x = cup)$ then $(is\text{-}cup\ y)$ else *False*)

lemma *brickcount-zero-implies-cup*: $(domain\ x = 0) \Longrightarrow (x = cup)$
 $\langle proof \rangle$

lemma *brickcount-zero-implies-brick-is-cup*: $(domain\ x = 0) \Longrightarrow (brick\text{-}is\text{-}cup\ x)$
 $\langle proof \rangle$

lemma *domain-zero-implies-is-cup*: $(domain\text{-}block\ x = 0) \Longrightarrow (is\text{-}cup\ x)$
 $\langle proof \rangle$

We need a function that checks if a wall represents a knot diagram.

primrec *is-tangle-diagram*:: $wall \Rightarrow bool$
where

```

is-tangle-diagram (basic x) = True
| is-tangle-diagram (x*xs) = (if is-tangle-diagram xs
                                then (codomain-block x = domain-wall xs)
                                else False)

```

```

definition is-link-diagram::wall  $\Rightarrow$  bool
where
is-link-diagram x  $\equiv$  (if (is-tangle-diagram x)
                        then (abs (domain-wall x) + abs(codomain-wall x) = 0)
                        else False)

end

```

2 Tangles: Definition as a type and basic functions on tangles

```

theory Tangles
imports Preliminaries
begin

```

well-defined wall as a type called diagram. The morphisms Abs_diagram maps a well defined wall to its diagram type and Rep_diagram maps the diagram back to the wall

```

typedef Tangle-Diagram = {(x::wall). is-tangle-diagram x}
  <proof>

```

```

typedef Link-Diagram = {(x::wall). is-link-diagram x}
  <proof>

```

The next few lemmas list the properties of well defined diagrams

For a well defined diagram, the morphism Rep_diagram acts as an inverse of Abs_diagram the morphism which maps a well defined wall to its representative in the type diagram

```

lemma Abs-Rep-well-defined:
assumes is-tangle-diagram x
shows Rep-Tangle-Diagram (Abs-Tangle-Diagram x) = x
  <proof>

```

The map Abs_diagram is injective

```

lemma Rep-Abs-well-defined:
assumes is-tangle-diagram x
and is-tangle-diagram y
and (Abs-Tangle-Diagram x) = (Abs-Tangle-Diagram y)
shows x = y

```

$\langle proof \rangle$

restating the property of well-defined wall in terms of diagram

In order to locally defined moves, it helps to prove that if composition of two wall is a well defined wall then the number of outgoing strands of the wall below are equal to the number of incoming strands of the wall above. The following lemmas prove that for a well defined wall, the number of incoming and outgoing strands are zero

lemma *is-tangle-left-compose:*

is-tangle-diagram $(x \circ y) \implies is-tangle-diagram\ x$
 $\langle proof \rangle$

lemma *is-tangle-right-compose:*

is-tangle-diagram $(x \circ y) \implies is-tangle-diagram\ y$
 $\langle proof \rangle$

lemma *comp-of-tangle-dgms:*

assumes *is-tangle-diagram* y
shows $((is-tangle-diagram\ x) \wedge (codomain-wall\ x = domain-wall\ y)) \implies is-tangle-diagram\ (x \circ y)$
 $\langle proof \rangle$

lemma *composition-of-tangle-diagrams:*

assumes *is-tangle-diagram* x
and *is-tangle-diagram* y
and $(domain-wall\ y = codomain-wall\ x)$
shows *is-tangle-diagram* $(x \circ y)$
 $\langle proof \rangle$

lemma *converse-composition-of-tangle-diagrams:*

is-tangle-diagram $(x \circ y) \implies (domain-wall\ y) = (codomain-wall\ x)$
 $\langle proof \rangle$

definition *compose-Tangle::Tangle-Diagram $\Rightarrow$ Tangle-Diagram $\Rightarrow$ Tangle-Diagram*

(infixl $\langle \circ \rangle$ 65)

where

compose-Tangle $x\ y = Abs-Tangle-Diagram$
 $((Rep-Tangle-Diagram\ x) \circ (Rep-Tangle-Diagram\ y))$

theorem *well-defined-compose:*

assumes *is-tangle-diagram* x

```

    and is-tangle-diagram y
    and domain-wall x = codomain-wall y
  shows (Abs-Tangle-Diagram x) ∘ (Abs-Tangle-Diagram y)
        = (Abs-Tangle-Diagram (x ∘ y))
  ⟨proof⟩

```

```

definition domain-Tangle::Tangle-Diagram ⇒ int
where
  domain-Tangle x = domain-wall(Rep-Tangle-Diagram x)

```

```

definition codomain-Tangle::Tangle-Diagram ⇒ int
where
  codomain-Tangle x = codomain-wall(Rep-Tangle-Diagram x)

```

```

end

```

3 Tangle_Algebra: Tensor product of tangles and its properties

```

theory Tangle-Algebra
imports Tangles
begin

```

4 Definition of tensor product of walls

the following definition is used to construct a block of n vert strands

```

primrec make-vert-block:: nat ⇒ block
where
  make-vert-block 0 = []
  |make-vert-block (Suc n) = vert#(make-vert-block n)

```

```

lemma domain-make-vert:domain-block (make-vert-block n) = int n
  ⟨proof⟩

```

```

lemma codomain-make-vert:codomain-block (make-vert-block n) = int n
  ⟨proof⟩

```

```

fun tensor::wall => wall => wall (infixr ⟨⊗⟩ 65)
where
  1:tensor (basic x) (basic y) = (basic (x ⊗ y))
  |2:tensor (x*xs) (basic y) = (
    if (codomain-block y = 0)

```

```

      then (x ⊗ y)*xs
      else
        (x ⊗ y)
        *(xs⊗(basic (make-vert-block (nat (codomain-block y))))))
|3:tensor (basic x) (y*ys) = (
  if (codomain-block x = 0)
  then (x ⊗ y)*ys
  else
    (x ⊗ y)
    *((basic (make-vert-block (nat (codomain-block x))))⊗ ys))
|4:tensor (x*xs) (y*ys) = (x ⊗ y)* (xs ⊗ ys)

```

5 Properties of tensor product of tangles

lemma *Nil-left-tensor:* $xs \otimes (\text{basic } []) = xs$
 ⟨proof⟩

lemma *Nil-right-tensor:* $(\text{basic } []) \otimes xs = xs$
 ⟨proof⟩

The definition of tensors is extended to diagrams by using the following function

definition *tensor-Tangle* :: *Tangle-Diagram* $\Rightarrow$ *Tangle-Diagram* $\Rightarrow$ *Tangle-Diagram*
 (infixl <⊗> 65)

where

tensor-Tangle $x\ y = \text{Abs-Tangle-Diagram } ((\text{Rep-Tangle-Diagram } x) \otimes (\text{Rep-Tangle-Diagram } y))$

lemma *tensor* (basic [vert]) (basic ([vert])) = (basic (([vert]) ⊗ ([vert])))
 ⟨proof⟩

domain_wall of a tensor product of two walls is the sum of the domain_wall of each of the tensor products

lemma *tensor-domain-wall-additivity:*

domain-wall (xs ⊗ ys) = *domain-wall* xs + *domain-wall* ys
 ⟨proof⟩

codomain of tensor of two walls is the sum of the respective codomain's is shown by the following theorem

lemma *tensor-codomain-wall-additivity:*

codomain-wall (xs ⊗ ys) = *codomain-wall* xs + *codomain-wall* ys
 ⟨proof⟩

theorem *is-tangle-make-vert-right:*

$(is-tangle-diagram\ xs)$
 $\implies is-tangle-diagram\ (xs \otimes (basic\ (make-vert-block\ n)))$
 $\langle proof \rangle$

theorem *is-tangle-make-vert-left*:
 $(is-tangle-diagram\ xs) \implies is-tangle-diagram\ ((basic\ (make-vert-block\ n)) \otimes xs)$
 $\langle proof \rangle$

lemma *simp1*: $(codomain-block\ y) \neq 0 \implies$
 $is-tangle-diagram\ (xs)$
 $\wedge is-tangle-diagram\ ((basic\ (make-vert-block\ (nat\ (codomain-block\ y)))) \longrightarrow$
 $is-tangle-diagram\ (xs \otimes ((basic\ (make-vert-block\ (nat\ (codomain-block\ y)))))) \implies$
 $(is-tangle-diagram\ (x * xs) \wedge is-tangle-diagram\ (basic\ y) \longrightarrow is-tangle-diagram\ (x$
 $* xs \otimes basic\ y))$
 $\langle proof \rangle$

lemma *simp2*:
 $(codomain-block\ x) \neq 0$
 $\implies$
 $is-tangle-diagram\ (basic\ (make-vert-block\ (nat\ (codomain-block\ x))))$
 $\wedge is-tangle-diagram\ (ys)$
 $\longrightarrow$
 $is-tangle-diagram\ ((basic\ (make-vert-block\ (nat\ (codomain-block\ x)))) \otimes ys)$
 $\implies$
 $(is-tangle-diagram\ (basic\ x)$
 $\wedge is-tangle-diagram\ (y * ys)$
 $\longrightarrow is-tangle-diagram\ ((basic\ x) \otimes (y * ys)))$
 $\langle proof \rangle$

lemma *simp3*:
 $is-tangle-diagram\ xs \wedge is-tangle-diagram\ ys \longrightarrow is-tangle-diagram\ (xs \otimes ys)$
 $\implies$
 $is-tangle-diagram\ (x * xs) \wedge is-tangle-diagram\ (y * ys)$
 $\longrightarrow is-tangle-diagram\ (x * xs \otimes y * ys)$
 $\langle proof \rangle$

theorem *is-tangle-diagramness*:
shows $(is-tangle-diagram\ x) \wedge (is-tangle-diagram\ y) \longrightarrow is-tangle-diagram\ (tensor\ x$
 $y)$
 $\langle proof \rangle$

theorem *tensor-preserves-is-tangle:*

assumes *is-tangle-diagram* x
and *is-tangle-diagram* y
shows *is-tangle-diagram* $(x \otimes y)$
 $\langle \text{proof} \rangle$

definition *Tensor-Tangle::Tangle-Diagram $\Rightarrow$ Tangle-Diagram $\Rightarrow$ Tangle-Diagram*

(**infixl** $\langle \circ \rangle$ 65)

where

Tensor-Tangle $x \ y =$
Abs-Tangle-Diagram $((\text{Rep-Tangle-Diagram } x) \otimes (\text{Rep-Tangle-Diagram } y))$

theorem *well-defined-compose:*

assumes *is-tangle-diagram* x
and *is-tangle-diagram* y
shows $(\text{Abs-Tangle-Diagram } x) \otimes (\text{Abs-Tangle-Diagram } y) = (\text{Abs-Tangle-Diagram } (x \otimes y))$
 $\langle \text{proof} \rangle$

end

theory *Tangle-Relation*

imports *Main*

begin

lemma *symmetry1: assumes* *symp* R

shows $\forall x \ y. (x, y) \in \{(x, y). R \ x \ y\}^* \longrightarrow (y, x) \in \{(x, y). R \ x \ y\}^*$
 $\langle \text{proof} \rangle$

lemma *symmetry2: assumes* $\forall x \ y. (x, y) \in \{(x, y). R \ x \ y\}^* \longrightarrow (y, x) \in \{(x, y). R \ x \ y\}^*$

shows *symp* $R^{\hat{**}}$
 $\langle \text{proof} \rangle$

lemma *symmetry3: assumes* *symp* R **shows** *symp* $R^{\hat{**}}$ $\langle \text{proof} \rangle$

lemma *symp-trans: assumes* *symp* R **shows** *symp* $R^{\hat{++}}$ $\langle \text{proof} \rangle$

end

6 Tangle_Moves: Defining moves on tangles

theory *Tangle-Moves*

imports *Tangles Tangle-Algebra Tangle-Relation*

begin

Two Links diagrams represent the same link if and only if the diagrams can be related by a set of moves called the Reidemeister moves. For links defined through Tangles, additional set of moves are needed to account for different tangle representations of the same link diagram.

We formalise these 'moves' in terms of relations. Each move is defined as a relation on diagrams. Two diagrams are then stated to be equivalent if the reflexive-symmetric-transitive closure of the disjunction of above relations holds true. A Link is defined as an element of the quotient type of diagrams modulo equivalence relations. We formalise the definition of framed links on similar lines.

In terms of formalising the moves, there is a trade off between choosing a small number of moves from which all other moves can be obtained, which is conducive to probe invariance of a function on diagrams. However, such an approach might not be conducive to establish equivalence of two diagrams. We opt for the former approach of minimising the number of tangle moves. However, the moves that would be useful in practice are proved as theorems in

type-synonym *relation = wall $\Rightarrow$ wall $\Rightarrow$ bool*

Link uncross

abbreviation *right-over::wall*

where

right-over $\equiv ((\text{basic } [vert, cup]) \circ (\text{basic } [over, vert]) \circ (\text{basic } [vert, cap]))$

abbreviation *left-over::wall*

where

left-over $\equiv ((\text{basic } (cup \# vert \# [])) \circ (\text{basic } (vert \# over \# [])) \circ (\text{basic } (cap \# vert \# [])))$

abbreviation *right-under::wall*

where

right-under $\equiv ((\text{basic } (vert \# cup \# [])) \circ (\text{basic } (under \# vert \# [])) \circ (\text{basic } (vert \# cap \# [])))$

abbreviation *left-under::wall*

where

left-under $\equiv ((\text{basic } (cup \# vert \# [])) \circ (\text{basic } (vert \# under \# [])) \circ (\text{basic } (cap \# vert \# [])))$

abbreviation *straight-line::wall*

where

straight-line $\equiv (\text{basic } (vert \# [])) \circ (\text{basic } (vert \# [])) \circ (\text{basic } (vert \# []))$

definition *uncross-positive-flip::relation*

where

uncross-positive-flip $x\ y \equiv ((x = \text{right-over}) \wedge (y = \text{left-over}))$

definition *uncross-positive-straighten::relation*

where

uncross-positive-straighten $x\ y \equiv ((x = \text{right-over}) \wedge (y = \text{straight-line}))$

definition *uncross-negative-flip::relation*

where

uncross-negative-flip $x\ y \equiv ((x = \text{right-under}) \wedge (y = \text{left-under}))$

definition *uncross-negative-straighten::relation*

where

uncross-negative-straighten $x\ y \equiv ((x = \text{left-under}) \wedge (y = \text{straight-line}))$

definition *uncross*

where

uncross $x\ y \equiv ((\text{uncross-positive-straighten } x\ y) \vee (\text{uncross-positive-flip } x\ y) \vee (\text{uncross-negative-straighten } x\ y) \vee (\text{uncross-negative-flip } x\ y))$

swing begins

abbreviation *r-over-braid::wall*

where

r-over-braid $\equiv ((\text{basic } ((\text{over}\#\text{vert}\#\square)) \circ (\text{basic } ((\text{vert}\#\text{over}\#\square)))) \circ (\text{basic } (\text{over}\#\text{vert}\#\square)))$

abbreviation *l-over-braid::wall*

where

l-over-braid $\equiv (\text{basic } (\text{vert}\#\text{over}\#\square)) \circ (\text{basic } (\text{over}\#\text{vert}\#\square)) \circ (\text{basic } (\text{vert}\#\text{over}\#\square))$

abbreviation *r-under-braid::wall*

where

r-under-braid $\equiv ((\text{basic } ((\text{under}\#\text{vert}\#\square)) \circ (\text{basic } ((\text{vert}\#\text{under}\#\square)))) \circ (\text{basic } (\text{under}\#\text{vert}\#\square)))$

abbreviation *l-under-braid::wall*

where

l-under-braid $\equiv (\text{basic } (\text{vert}\#\text{under}\#\square)) \circ (\text{basic } (\text{under}\#\text{vert}\#\square)) \circ (\text{basic } (\text{vert}\#\text{under}\#\square))$

definition *swing-pos::wall $\Rightarrow$ wall $\Rightarrow$ bool*

where

$swing-pos\ x\ y \equiv (x = r-over-braid) \wedge (y = l-over-braid)$

definition $swing-neg::wall \Rightarrow wall \Rightarrow bool$

where

$swing-neg\ x\ y \equiv (x = r-under-braid) \wedge (y = l-under-braid)$

definition $swing::relation$

where

$swing\ x\ y \equiv (swing-pos\ x\ y) \vee (swing-neg\ x\ y)$

pull begins

definition $pull-posneg::relation$

where

$pull-posneg\ x\ y \equiv ((x = ((basic\ (over\ \#[])) \circ (basic\ (under\ \#[]))))$
 $\wedge (y = ((basic\ (vert\ \#vert\ \#[]))$
 $\circ (basic\ ((vert\ \#vert\ \#[]))))))$

definition $pull-negpos::relation$

where

$pull-negpos\ x\ y \equiv ((x = ((basic\ (under\ \#[])) \circ (basic\ (over\ \#[]))))$
 $\wedge (y = ((basic\ (vert\ \#vert\ \#[]))$
 $\circ (basic\ ((vert\ \#vert\ \#[]))))))$

pull definition

definition $pull::relation$

where

$pull\ x\ y \equiv ((pull-posneg\ x\ y) \vee (pull-negpos\ x\ y))$

linkrel-pull ends

linkrel-straighten

definition $straighten-topdown::relation$

where

$straighten-topdown\ x\ y \equiv ((x = ((basic\ ((vert\ \#cup\ \#[]))$
 $\circ (basic\ ((cap\ \#vert\ \#[]))))$
 $\wedge (y = ((basic\ (vert\ \#[])) \circ (basic\ (vert\ \#[]))))))$

definition $straighten-downtop::relation$

where

$straighten-downtop\ x\ y \equiv ((x = ((basic\ ((cup\ \#vert\ \#[]))$
 $\circ (basic\ ((vert\ \#cap\ \#[]))))$
 $\wedge (y = ((basic\ (vert\ \#[])) \circ (basic\ (vert\ \#[]))))))$

definition straighten

definition $straighten::relation$

where

$straighten\ x\ y \equiv ((straighten\text{-}topdown\ x\ y) \vee (straighten\text{-}downtop\ x\ y))$

straighten ends

rotate moves

definition $rotate\text{-}toppos::relation$

where

$rotate\text{-}toppos\ x\ y \equiv ((x = ((basic\ ((vert\ \#over\ \#[]))$
 $\quad \circ(basic\ ((cap\ \#vert\ \#[]))))$
 $\wedge (y = ((basic\ ((under\ \#vert\ \#[]))$
 $\quad \circ(basic\ ((vert\ \#cap\ \#[]))))))$

definition $rotate\text{-}topneg::wall \Rightarrow wall \Rightarrow bool$

where

$rotate\text{-}topneg\ x\ y \equiv ((x = ((basic\ ((vert\ \#under\ \#[]))$
 $\quad \circ(basic\ ((cap\ \#vert\ \#[]))))$
 $\wedge (y = ((basic\ ((over\ \#vert\ \#[]))$
 $\quad \circ(basic\ ((vert\ \#cap\ \#[]))))))$

definition $rotate\text{-}downpos::wall \Rightarrow wall \Rightarrow bool$

where

$rotate\text{-}downpos\ x\ y \equiv ((x = ((basic\ (cup\ \#vert\ \#[]))$
 $\quad \circ(basic\ (vert\ \#over\ \#[]))))$
 $\wedge (y = ((basic\ ((vert\ \#cup\ \#[]))$
 $\quad \circ(basic\ ((under\ \#vert\ \#[]))))))$

definition $rotate\text{-}downneg::wall \Rightarrow wall \Rightarrow bool$

where

$rotate\text{-}downneg\ x\ y \equiv ((x = ((basic\ (cup\ \#vert\ \#[]))$
 $\quad \circ(basic\ (vert\ \#under\ \#[]))))$
 $\wedge (y = ((basic\ ((vert\ \#cup\ \#[]))$
 $\quad \circ(basic\ ((over\ \#vert\ \#[]))))))$

rotate definition

definition $rotate::wall \Rightarrow wall \Rightarrow bool$

where

$rotate\ x\ y \equiv ((rotate\text{-}toppos\ x\ y) \vee (rotate\text{-}topneg\ x\ y)$
 $\vee (rotate\text{-}downpos\ x\ y) \vee (rotate\text{-}downneg\ x\ y))$

rotate ends

Compress - Compress has two levels of equivalences. It is a composition of Compress-null, compbelow and compabove. compbelow and compabove are further written as disjunction of many other relations. Compbelow refers to when the bottom row is extended or compressed. Compabove refers to when the row above is extended or compressed

definition $compress-top1::wall \Rightarrow wall \Rightarrow bool$

where

$compress-top1\ x\ y \equiv \exists B.((x = (basic\ (make-vert-block\ (nat\ (domain-wall\ B)))) \circ B)$

$$\wedge(y = B) \wedge (codomain-wall\ B \neq 0) \\ \wedge(is-tangle-diagram\ B))$$

definition $compress-bottom1::wall \Rightarrow wall \Rightarrow bool$

where

$compress-bottom1\ x\ y \equiv \exists B.((x = B \circ (basic\ (make-vert-block\ (nat\ (codomain-wall\ B))))))$

$$\wedge(y = B) \wedge (domain-wall\ B \neq 0) \\ \wedge(is-tangle-diagram\ B))$$

definition $compress-bottom::wall \Rightarrow wall \Rightarrow bool$

where

$compress-bottom\ x\ y \equiv \exists B.((x = B \circ (basic\ (make-vert-block\ (nat\ (codomain-wall\ B))))))$

$$\wedge(y = ((basic\ ([\])) \circ B)) \wedge (domain-wall\ B = 0) \\ \wedge(is-tangle-diagram\ B))$$

definition $compress-top::wall \Rightarrow wall \Rightarrow bool$

where

$compress-top\ x\ y \equiv \exists B.((x = (basic\ (make-vert-block\ (nat\ (domain-wall\ B)))) \circ B)$

$$\wedge(y = (B \circ (basic\ ([\])))) \wedge (codomain-wall\ B = 0) \\ \wedge(is-tangle-diagram\ B))$$

definition $compress::wall \Rightarrow wall \Rightarrow bool$

where

$compress\ x\ y = ((compress-top\ x\ y) \vee (compress-bottom\ x\ y))$

slide relation refer to the relation where a crossing is slided over a vertical strand

definition $slide::wall \Rightarrow wall \Rightarrow bool$

where

$slide\ x\ y \equiv \exists B.((x = ((basic\ (make-vert-block\ (nat\ (domain-block\ B)))) \circ (basic\ B)))$

$$\wedge(y = ((basic\ B) \circ (basic\ (make-vert-block\ (nat\ (codomain-block\ B)))))) \\ \wedge ((domain-block\ B) \neq 0))$$

linkrel-definition

definition $linkrel::wall \Rightarrow wall \Rightarrow bool$

where

$linkrel\ x\ y = ((uncross\ x\ y) \vee (pull\ x\ y) \vee (straighten\ x\ y) \\ \vee (swing\ x\ y) \vee (rotate\ x\ y) \vee (compress\ x\ y) \vee (slide\ x\ y))$

definition *framed-uncross::wall* $\Rightarrow$ *wall* $\Rightarrow$ *bool*
where
framed-uncross *x y* $\equiv ((\text{uncross-positive-flip } x y) \vee (\text{uncross-negative-flip } x y))$

definition *framed-linkrel::wall* $\Rightarrow$ *wall* $\Rightarrow$ *bool*
where
framed-linkrel *x y* $= ((\text{framed-uncross } x y) \vee (\text{pull } x y) \vee (\text{straighten } x y) \vee (\text{swing } x y) \vee (\text{rotate } x y) \vee (\text{compress } x y) \vee (\text{slide } x y))$

lemma *framed-uncross-implies-uncross*: (*framed-uncross* *x y*) $\implies$ (*uncross* *x y*)
 $\langle \text{proof} \rangle$

end

7 Link_Algebra: Defining equivalence of tangles and links

theory *Link-Algebra*
imports *Tangles Tangle-Algebra Tangle-Moves*
begin

inductive *Tangle-Equivalence* :: *wall* $\Rightarrow$ *wall* $\Rightarrow$ *bool* (**infixl** $\sim$ 64)
where
 $\text{refl } [\text{intro!}, \text{Pure.intro!}, \text{simp}]: a \sim a$
 $\text{equality } [\text{Pure.intro}]: \text{linkrel } a b \implies a \sim b$
 $\text{domain-compose}: (\text{domain-wall } a = 0) \wedge (\text{is-tangle-diagram } a) \implies a \sim ((\text{basic } []) \circ a)$
 $\text{codomain-compose}: (\text{codomain-wall } a = 0) \wedge (\text{is-tangle-diagram } a) \implies a \sim (a \circ (\text{basic } []))$
 $\text{compose-eq}: ((B::\text{wall}) \sim D) \wedge ((A::\text{wall}) \sim C) \wedge (\text{is-tangle-diagram } A) \wedge (\text{is-tangle-diagram } B) \wedge (\text{is-tangle-diagram } C) \wedge (\text{is-tangle-diagram } D) \wedge (\text{domain-wall } B) = (\text{codomain-wall } A) \wedge (\text{domain-wall } D) = (\text{codomain-wall } C) \implies ((A::\text{wall}) \circ B) \sim (C \circ D)$
 $\text{trans}: A \sim B \implies B \sim C \implies A \sim C$
 $\text{sym}: A \sim B \implies B \sim A$
 $\text{tensor-eq}: ((B::\text{wall}) \sim D) \wedge ((A::\text{wall}) \sim C) \wedge (\text{is-tangle-diagram } A) \wedge (\text{is-tangle-diagram } B) \wedge (\text{is-tangle-diagram } C) \wedge (\text{is-tangle-diagram } D) \implies ((A::\text{wall}) \otimes B) \sim (C \otimes D)$

inductive *Framed-Tangle-Equivalence* :: *wall* $\Rightarrow$ *wall* $\Rightarrow$ *bool* (**infixl** $\sim_f$ 64)
where

$refl [intro!, Pure.intro!, simp]: a \sim_f a$
 $|equality [Pure.intro]: framed-linkrel\ a\ b \implies a \sim_f b$
 $|domain-compose:(domain-wall\ a = 0) \wedge (is-tangle-diagram\ a) \implies a \sim_f ((basic\ []) \circ a)$
 $|codomain-compose:(codomain-wall\ a = 0) \wedge (is-tangle-diagram\ a) \implies a \sim_f (a \circ (basic\ []))$
 $|compose-eq:((B::wall) \sim_f D) \wedge ((A::wall) \sim_f C)$
 $\quad \wedge (is-tangle-diagram\ A) \wedge (is-tangle-diagram\ B)$
 $\quad \wedge (is-tangle-diagram\ C) \wedge (is-tangle-diagram\ D)$
 $\quad \wedge (domain-wall\ B) = (codomain-wall\ A)$
 $\quad \wedge (domain-wall\ D) = (codomain-wall\ C)$
 $\quad \implies ((A::wall) \circ B) \sim_f (C \circ D)$
 $|trans: A \sim_f B \implies B \sim_f C \implies A \sim_f C$
 $|sym: A \sim_f B \implies B \sim_f A$
 $|tensor-eq: ((B::wall) \sim_f D) \wedge ((A::wall) \sim_f C) \wedge (is-tangle-diagram\ A) \wedge (is-tangle-diagram\ B)$
 $\quad \wedge (is-tangle-diagram\ C) \wedge (is-tangle-diagram\ D) \implies ((A::wall) \otimes B) \sim_f (C \otimes D)$

definition *Tangle-Diagram-Equivalence*::*Tangle-Diagram* $\Rightarrow$ *Tangle-Diagram* $\Rightarrow$ *bool*

(infixl $\langle \sim T \rangle$ 64)

where

Tangle-Diagram-Equivalence *T1 T2* $\equiv$

(*Rep-Tangle-Diagram* *T1*) $\sim$ (*Rep-Tangle-Diagram* *T2*)

definition *Link-Diagram-Equivalence*::*Link-Diagram* $\Rightarrow$ *Link-Diagram* $\Rightarrow$ *bool*

(infixl $\langle \sim L \rangle$ 64)

where

Link-Diagram-Equivalence *T1 T2* $\equiv$ (*Rep-Link-Diagram* *T1*) $\sim$ (*Rep-Link-Diagram* *T2*)

quotient-type *Tangle* = *Tangle-Diagram* / *Tangle-Diagram-Equivalence*

morphisms *Rep-Tangles* *Abs-Tangles*

$\langle proof \rangle$

quotient-type *Link* = *Link-Diagram* / *Link-Diagram-Equivalence*

morphisms *Rep-Links* *Abs-Links*

$\langle proof \rangle$

definition *Framed-Tangle-Diagram-Equivalence*::*Tangle-Diagram* $\Rightarrow$ *Tangle-Diagram*

$\Rightarrow$ *bool*

(infixl $\langle \sim T \rangle$ 64)

where

Framed-Tangle-Diagram-Equivalence *T1 T2*

$\equiv$ (*Rep-Tangle-Diagram* *T1*) $\sim$ (*Rep-Tangle-Diagram* *T2*)

definition *Framed-Link-Diagram-Equivalence*::*Link-Diagram* $\Rightarrow$ *Link-Diagram* $\Rightarrow$

```

bool
(infixl <~L> 64)
where
  Framed-Link-Diagram-Equivalence T1 T2
    ≡ (Rep-Link-Diagram T1) ~ (Rep-Link-Diagram T2)

```

```

quotient-type Framed-Tangle = Tangle-Diagram
  /Framed-Tangle-Diagram-Equivalence
morphisms Rep-Framed-Tangles Abs-Framed-Tangles
<proof>

```

```

quotient-type Framed-Link = Link-Diagram/Framed-Link-Diagram-Equivalence
morphisms Rep-Framed-Links Abs-Framed-Links
<proof>

```

```

end

```

8 Showing equivalence of links: An example

```

theory Example
imports Link-Algebra
begin

```

We prove that a link diagram with a single crossing is equivalent to the unknot

```

lemma transitive: assumes a~b and b~c shows a~c
  <proof>

```

```

lemma prelim-cup-compress:
  ((basic (cup#[])) o (basic (vert # vert # []))) ~
    ((basic [] o (basic (cup#[])))
  <proof>

```

```

lemma cup-compress:
  (basic (cup#[])) o (basic (vert # vert # [])) ~ (basic (cup#[]))
  <proof>

```

```

abbreviation x::wall
where
  x ≡ (basic [cup,cup]) o (basic [vert,over,vert]) o (basic [cap,cap])

```

```

abbreviation y::wall
where
  y ≡ (basic [cup]) o (basic [cap])

```

```

lemma uncross-straighten-left-over:left-over ~ straight-line
  <proof>

```

theorem *Example:*

$x \sim y$
<proof>

end

9 Kauffman Matrix and Kauffman Bracket- Definitions and Properties

theory *Kauffman-Matrix*

imports

Matrix-Tensor.Matrix-Tensor

Link-Algebra

HOL-Computational-Algebra.Polynomial

HOL-Computational-Algebra.Fraction-Field

begin

10 Rational Functions

intpoly is the type of integer polynomials

type-synonym *intpoly* = *int poly*

lemma *eval-pCons: poly (pCons 0 1) x = x*
<proof>

lemma *pCons2: (pCons 0 1) ≠ (1::int poly)*
<proof>

definition *var-def: x = (pCons 0 1)*

lemma *non-zero:x ≠ 0*
<proof>

rat_poly is the fraction field of integer polynomials. In other words, it is the type of rational functions

type-synonym *rat-poly* = *intpoly fract*

A is defined to be x/1, while B is defined to be 1/x

definition *var-def1:A = Fract x 1*

definition *var-def2: B = Fract 1 x*

lemma *assumes* $b \neq 0$ **and** $d \neq 0$
shows $\text{Fract } a \ b = \text{Fract } c \ d \longleftrightarrow a * d = c * b$
 $\langle \text{proof} \rangle$

lemma *A-non-zero*: $A \neq (0::\text{rat-poly})$
 $\langle \text{proof} \rangle$

lemma *mult-inv-non-zero*:
assumes $(p::\text{rat-poly}) \neq 0$
and $p*q = (1::\text{rat-poly})$
shows $q \neq 0$
 $\langle \text{proof} \rangle$

abbreviation *rat-poly-times*:: $\text{rat-poly} \Rightarrow \text{rat-poly} \Rightarrow \text{rat-poly}$
where
rat-poly-times $p \ q \equiv p*q$

abbreviation *rat-poly-plus*:: $\text{rat-poly} \Rightarrow \text{rat-poly} \Rightarrow \text{rat-poly}$
where
rat-poly-plus $p \ q \equiv p+q$

abbreviation *rat-poly-inv*:: $\text{rat-poly} \Rightarrow \text{rat-poly}$
where
rat-poly-inv $p \equiv (- \ p)$

interpretation *rat-poly:semiring-0* *rat-poly-plus 0* *rat-poly-times*
 $\langle \text{proof} \rangle$

interpretation *rat-poly:semiring-1* *1* *rat-poly-times* *rat-poly-plus 0*
 $\langle \text{proof} \rangle$

lemma *mat1-equiv*: $\text{mat1 } (1::\text{nat}) = [[(1::\text{rat-poly})]]$
 $\langle \text{proof} \rangle$

rat_poly is an interpretation of the locale *plus_mult*

interpretation *rat-poly:plus-mult 1* *rat-poly-times 0* *rat-poly-plus*
rat-poly-inv
 $\langle \text{proof} \rangle$

lemma *rat-poly.matrix-mult* $[[A, 1], [0, A]] \ [[A, 0], [0, A]] = [[A*A, A], [0, A*A]]$
 $\langle \text{proof} \rangle$

abbreviation

$rat-polymat-tensor::rat-poly\ mat \Rightarrow rat-poly\ mat \Rightarrow rat-poly\ mat$
 (infixl $\langle \otimes \rangle$ 65)

where

$rat-polymat-tensor\ p\ q \equiv rat-poly.Tensor\ p\ q$

lemma assumes $(j::nat)\ div\ a = i\ div\ a$

and $j\ mod\ a = i\ mod\ a$

shows $j = i$

$\langle proof \rangle$

lemma $[[1]] \otimes M = M$

$\langle proof \rangle$

lemma $M \otimes [[1]] = M$

$\langle proof \rangle$

11 Kauffman matrices

We assign every brick to a matrix of rational polynomials

primrec $brickmat::brick \Rightarrow rat-poly\ mat$

where

$brickmat\ vert = [[1,0],[0,1]]$

$|brickmat\ cup = [[0],[A],[-B],[0]]$

$|brickmat\ cap = [[0,-A,B,0]]$

$|brickmat\ over = [[A,0,0,0],$
 $[0,0,B,0],$
 $[0,B,A-(B*B*B),0],$
 $[0,0,0,A]]$

$|brickmat\ under = [[B,0,0,0],$
 $[0,B-(A*A*A),A,0],$
 $[0,A,0,0],$
 $[0,0,0,B]]$

lemma $inverse1:rat-poly-times\ A\ B = 1$

$\langle proof \rangle$

lemma $inverse2:rat-poly-times\ B\ A = 1$

$\langle proof \rangle$

lemma $B-non-zero:B \neq 0$

$\langle proof \rangle$

lemma $rat-poly-times\ p\ (q + r)$

$= (rat-poly-times\ p\ q) + (rat-poly-times\ p\ r)$

$\langle proof \rangle$

lemma *minus-left-distributivity*:

$$\begin{aligned} \text{rat-poly-times } p \ (q - r) \\ = (\text{rat-poly-times } p \ q) - (\text{rat-poly-times } p \ r) \end{aligned}$$

$\langle \text{proof} \rangle$

lemma *minus-right-distributivity*:

$$\text{rat-poly-times } (p - q) \ r = (\text{rat-poly-times } p \ r) - (\text{rat-poly-times } q \ r)$$

$\langle \text{proof} \rangle$

lemma *equation*:

$$\begin{aligned} \text{rat-poly-plus} \\ (\text{rat-poly-times } B \ (B - \text{rat-poly-times } (\text{rat-poly-times } A \ A) \ A)) \\ (\text{rat-poly-times } (A - \text{rat-poly-times } (\text{rat-poly-times } B \ B) \ B) \ A) \\ = 0 \end{aligned}$$

$\langle \text{proof} \rangle$

lemma *rat-poly.matrix-mult (brickmat over) (brickmat under)*

$$= [[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]]$$

$\langle \text{proof} \rangle$

lemma *rat-poly-inv* $A = -A$

$\langle \text{proof} \rangle$

lemma *vert-dim:rat-poly.row-length (brickmat vert) = 2* $\wedge$ *length (brickmat vert)*
= 2

$\langle \text{proof} \rangle$

lemma *cup-dim:rat-poly.row-length (brickmat cup) = 1* **and** *length (brickmat cup)*
= 4

$\langle \text{proof} \rangle$

lemma *cap-dim:rat-poly.row-length (brickmat cap) = 4* **and** *length (brickmat cap)*
= 1

$\langle \text{proof} \rangle$

lemma *over-dim:rat-poly.row-length (brickmat over) = 4* **and** *length (brickmat over)* = 4

$\langle \text{proof} \rangle$

lemma *under-dim:rat-poly.row-length (brickmat under) = 4* **and** *length (brickmat under)* = 4

$\langle \text{proof} \rangle$

lemma *mat-vert:mat 2 2 (brickmat vert)*

$\langle \text{proof} \rangle$

lemma *mat-cup:mat 1 4 (brickmat cup)*

$\langle \text{proof} \rangle$

lemma *mat-cap:mat 4 1 (brickmat cap)*

$\langle \text{proof} \rangle$

lemma *mat-over:mat 4 4 (brickmat over)*

$\langle \text{proof} \rangle$

lemma *mat-under:mat 4 4 (brickmat under)*

$\langle proof \rangle$

primrec $rowlength::nat \Rightarrow nat$

where

$rowlength\ 0 = 1$

$|rowlength\ (Suc\ k) = 2*(Suc\ k)$

lemma $(rat-poly.row-length\ (brickmat\ d)) = (2^{nat\ (domain\ d)})$

$\langle proof \rangle$

lemma $rat-poly.row-length\ (brickmat\ cup) = 1$

$\langle proof \rangle$

lemma $two:(Suc\ (Suc\ 0)) = 2$

$\langle proof \rangle$

we assign every block to a matrix of rational function as follows

primrec $blockmat::block \Rightarrow rat-poly\ mat$

where

$blockmat\ [] = [[1]]$

$|blockmat\ (l\#ls) = (brickmat\ l) \otimes (blockmat\ ls)$

lemma $blockmat\ [a] = brickmat\ a$

$\langle proof \rangle$

lemma $nat-sum:$

assumes $a \geq 0$ **and** $b \geq 0$

shows $nat\ (a+b) = (nat\ a) + (nat\ b)$

$\langle proof \rangle$

lemma $rat-poly.row-length\ (blockmat\ ls) = (2^{nat\ ((domain-block\ ls))})$

$\langle proof \rangle$

lemma $row-length-domain-block:$

$rat-poly.row-length\ (blockmat\ ls) = (2^{nat\ ((domain-block\ ls))})$

$\langle proof \rangle$

lemma $length-codomain-block:length\ (blockmat\ ls)$

$= (2^{nat\ ((codomain-block\ ls))})$

$\langle proof \rangle$

lemma $matrix-blockmat:$

mat

$(rat-poly.row-length\ (blockmat\ ls))$

$(length\ (blockmat\ ls))$

(*blockmat ls*)

⟨*proof*⟩

The function `kauff_mat` below associates every wall to a matrix. We call this the Kauffman matrix. When the wall represents a well defined tangle diagram, the Kauffman matrix is a 1×1 matrix whose entry is the Kauffman bracket.

primrec *kauff_mat::wall* $\Rightarrow$ *rat-poly mat*

where

kauff_mat (*basic w*) = (*blockmat w*)

|*kauff_mat* (*w*ws*) = *rat-poly.matrix-mult* (*blockmat w*) (*kauff_mat ws*)

The following theorem tells us that if a wall represents a tangle diagram, then its Kauffman matrix is a ‘valid’ matrix.

theorem *matrix-kauff_mat:*

((*is-tangle-diagram ws*)

$\Rightarrow$ (*rat-poly.row-length* (*kauff_mat ws*)) = $2^{\neg(\text{nat } (\text{domain-wall } ws))}$

$\wedge$ (*length* (*kauff_mat ws*)) = $2^{\neg(\text{nat } (\text{codomain-wall } ws))}$

$\wedge$ (*mat*

(*rat-poly.row-length* (*kauff_mat ws*))

(*length* (*kauff_mat ws*))

(*kauff_mat ws*)))

⟨*proof*⟩

theorem *effective-matrix-kauff_mat:*

assumes *is-tangle-diagram ws*

shows (*rat-poly.row-length* (*kauff_mat ws*)) = $2^{\neg(\text{nat } (\text{domain-wall } ws))}$

and *length* (*kauff_mat ws*) = $2^{\neg(\text{nat } (\text{codomain-wall } ws))}$

and *mat* (*rat-poly.row-length* (*kauff_mat ws*)) (*length* (*kauff_mat ws*))

(*kauff_mat ws*)

⟨*proof*⟩

lemma *mat-mult-equiv:*

rat-poly.matrix-mult m1 m2 = *mat-mult* (*rat-poly.row-length m1*) *m1 m2*

⟨*proof*⟩

theorem *associative-rat-poly-mat:*

assumes *mat* (*rat-poly.row-length m1*) (*rat-poly.row-length m2*) *m1*

and *mat* (*rat-poly.row-length m2*) (*rat-poly.row-length m3*) *m2*

and *mat* (*rat-poly.row-length m3*) *nc m3*

shows *rat-poly.matrix-mult m1* (*rat-poly.matrix-mult m2 m3*)

= *rat-poly.matrix-mult* (*rat-poly.matrix-mult m1 m2*) *m3*

⟨*proof*⟩

It follows from this result that the Kauffman Matrix of a wall representing a link diagram, is a 1×1 matrix. Thus it establishes a correspondence between links and rational functions.

theorem *link-diagram-matrix*:
assumes *is-link-diagram ws*
shows *mat 1 1 (kauff-mat ws)*
 $\langle \text{proof} \rangle$

theorem *tangle-compose-matrix*:
 $((\text{is-tangle-diagram } ws1) \wedge (\text{is-tangle-diagram } ws2))$
 $\wedge (\text{domain-wall } ws2 = \text{codomain-wall } ws1)) \implies$
 $\text{kauff-mat } (ws1 \circ ws2) = \text{rat-poly.matrix-mult } (\text{kauff-mat } ws1) (\text{kauff-mat } ws2)$
 $\langle \text{proof} \rangle$

theorem *left-mat-compose*:
assumes *is-tangle-diagram ws*
and *codomain-wall ws = 0*
shows $\text{kauff-mat } ws = (\text{kauff-mat } (ws \circ (\text{basic } [])))$
 $\langle \text{proof} \rangle$

theorem *right-mat-compose*:
assumes *is-tangle-diagram ws* **and** *domain-wall ws = 0*
shows $\text{kauff-mat } ws = (\text{kauff-mat } ((\text{basic } []) \circ ws))$
 $\langle \text{proof} \rangle$

lemma *left-id-blockmat*: $\text{blockmat } [] \otimes \text{blockmat } b = \text{blockmat } b$
 $\langle \text{proof} \rangle$

lemma *tens-assoc*:
 $\forall a \text{ } xs \text{ } ys. (\text{brickmat } a \otimes (\text{blockmat } xs \otimes \text{blockmat } ys))$
 $= (\text{brickmat } a \otimes \text{blockmat } xs) \otimes \text{blockmat } ys$
 $\langle \text{proof} \rangle$

lemma *kauff-mat-tensor-distrib*:
 $\forall xs. \forall ys. (\text{kauff-mat } (\text{basic } xs \otimes \text{basic } ys))$
 $= \text{kauff-mat } (\text{basic } xs) \otimes \text{kauff-mat } (\text{basic } ys)$
 $\langle \text{proof} \rangle$

lemma *blockmat-tensor-distrib*:
 $(\text{blockmat } (a \otimes b)) = (\text{blockmat } a) \otimes (\text{blockmat } b)$
 $\langle \text{proof} \rangle$

lemma *blockmat-non-empty*: $\forall bs. (\text{blockmat } bs \neq [])$
 $\langle \text{proof} \rangle$

The kauffman matrix of a wall representing a tangle diagram is non empty

lemma *kauff-mat-non-empty*:
fixes *ws*
assumes *is-tangle-diagram ws*
shows $\text{kauff-mat } ws \neq []$

⟨proof⟩

lemma *is-tangle-diagram-length-rowlength*:
assumes *is-tangle-diagram* ($w * ws$)
shows $\text{length } (\text{blockmat } w) = \text{rat-poly.row-length } (\text{kauff-mat } ws)$
 ⟨proof⟩

lemma *is-tangle-diagram-matrix-match*:
assumes *is-tangle-diagram* ($w1 * ws1$)
and *is-tangle-diagram* ($w2 * ws2$)
shows $\text{rat-poly.matrix-match } (\text{blockmat } w1)$
 $\quad (\text{kauff-mat } ws1) (\text{blockmat } w2) (\text{kauff-mat } ws2)$
 ⟨proof⟩

The following function constructs a $2^n \times 2^n$ identity matrix for a given n

primrec *make-vert-equiv::nat $\Rightarrow$ rat-poly mat*
where
 $\text{make-vert-equiv } 0 = [[1]]$
 $|\text{make-vert-equiv } (\text{Suc } k) = ((\text{mat1 } 2) \otimes (\text{make-vert-equiv } k))$

lemma *mve1:make-vert-equiv 1 = (mat1 2)*
 ⟨proof⟩

lemma
assumes $i < 2$ **and** $j < 2$
shows $(\text{make-vert-equiv } 1)!i!j = (\text{if } i = j \text{ then } 1 \text{ else } 0)$
 ⟨proof⟩

lemma *mat1-vert-equiv:(mat1 2) = (brickmat vert) (is ?l = ?r)*
 ⟨proof⟩

lemma *blockmat-make-vert*:
 $\text{blockmat } (\text{make-vert-block } n) = (\text{make-vert-equiv } n)$
 ⟨proof⟩

lemma *prop-make-vert-equiv*:
shows $\text{rat-poly.row-length } (\text{make-vert-equiv } n) = 2^n$
and $\text{length } (\text{make-vert-equiv } n) = 2^n$
and mat
 $\quad (\text{rat-poly.row-length } (\text{make-vert-equiv } n))$
 $\quad (\text{length } (\text{make-vert-equiv } n))$
 $\quad (\text{make-vert-equiv } n)$
 ⟨proof⟩

abbreviation *nat-mult::nat $\Rightarrow$ nat $\Rightarrow$ nat* (**infixl** $\langle * \rangle$ 65)
where
 $\text{nat-mult } a \ b \equiv ((a::\text{nat}) * b)$

lemma *equal-div-mod*:**assumes** $((j::nat) \text{ div } a) = (i \text{ div } a)$
and $(j \text{ mod } a) = (i \text{ mod } a)$
shows $j = i$
 $\langle proof \rangle$

lemma *equal-div-mod2*: $((j::nat) \text{ div } a) = (i \text{ div } a)$
 $\wedge ((j \text{ mod } a) = (i \text{ mod } a)) = (j = i)$
 $\langle proof \rangle$

lemma *impl-rule*:
assumes $(\forall i < m. \forall j < n. (P \ i) \wedge (Q \ j))$
and $\forall i \ j. (P \ i) \wedge (Q \ j) \longrightarrow R \ i \ j$
shows $(\forall i < m. \forall j < n. R \ i \ j)$
 $\langle proof \rangle$

lemma *implic*:
assumes $\forall i \ j. ((P \ i \ j) \longrightarrow (Q \ i \ j))$
and $\forall i \ j. ((Q \ i \ j) \longrightarrow (R \ i \ j))$
shows $\forall i \ j. ((P \ i \ j) \longrightarrow (R \ i \ j))$
 $\langle proof \rangle$

lemma *assumes* $a < (b * c)$
shows $((a::nat) \text{ div } b) < c$
 $\langle proof \rangle$

lemma *mult-if-then*: $((v = (\text{if } P \text{ then } 1 \text{ else } 0))$
 $\wedge (w = (\text{if } Q \text{ then } 1 \text{ else } 0)))$
 $\implies (\text{rat-poly-times } v \ w = (\text{if } (P \wedge Q) \text{ then } 1 \text{ else } 0))$
 $\langle proof \rangle$

lemma *rat-poly-unity*:*rat-poly-times* 1 1 = 1
 $\langle proof \rangle$

lemma $((P \wedge Q) \longrightarrow R) \implies (P \longrightarrow Q \longrightarrow R)$
 $\langle proof \rangle$

lemma *length* (*mat1* 2) = 2
 $\langle proof \rangle$

theorem *make-vert-equiv-mat*:
make-vert-equiv n = (*mat1* (2ⁿ))
 $\langle proof \rangle$

theorem *make-vert-block-map-blockmat*:
blockmat (*make-vert-block* n) = (*mat1* (2ⁿ))
 $\langle proof \rangle$

lemma *mat1-rt-mult*:**assumes** *mat* nr nc m1

shows $\text{rat-poly.matrix-mult } m1 \ (\text{mat1 } (nc)) = m1$
 $\langle \text{proof} \rangle$

lemma mat1-vert-block :
 $\text{rat-poly.matrix-mult}$
 $\quad (\text{blockmat } b)$
 $\quad (\text{blockmat } (\text{make-vert-block } (\text{nat } (\text{codomain-block } b))))$
 $\quad = (\text{blockmat } b)$
 $\langle \text{proof} \rangle$

The following list of theorems deal with distributivity properties of tensor product of matrices (with entries as rational functions) and composition

definition weak-matrix-match ::
 $\text{rat-poly mat} \Rightarrow \text{rat-poly mat} \Rightarrow \text{rat-poly mat} \Rightarrow \text{bool}$

where
 $\text{weak-matrix-match } A1 \ A2 \ B1 \equiv (\text{mat } (\text{rat-poly.row-length } A1) \ (\text{length } A1) \ A1)$
 $\quad \wedge (\text{mat } (\text{rat-poly.row-length } A2) \ (\text{length } A2) \ A2)$
 $\quad \wedge (\text{mat } (\text{rat-poly.row-length } B1) \ 1 \ B1)$
 $\wedge (A1 \neq []) \wedge (A2 \neq []) \wedge (B1 \neq [])$
 $\quad \wedge (\text{length } A1 = \text{rat-poly.row-length } A2)$

theorem $\text{weak-distributivity1}$:
 $\text{weak-matrix-match } A1 \ A2 \ B1$
 $\implies ((\text{rat-poly.matrix-mult } A1 \ A2) \otimes B1)$
 $\quad = (\text{rat-poly.matrix-mult } (A1 \otimes B1) \ (A2))$
 $\langle \text{proof} \rangle$

definition $\text{weak-matrix-match2}$::
 $\text{rat-poly mat} \Rightarrow \text{rat-poly mat} \Rightarrow \text{rat-poly mat} \Rightarrow \text{bool}$
where
 $\text{weak-matrix-match2 } A1 \ B1 \ B2 \equiv (\text{mat } (\text{rat-poly.row-length } A1) \ 1 \ A1)$
 $\quad \wedge (\text{mat } (\text{rat-poly.row-length } B1) \ (\text{length } B1) \ B1)$
 $\quad \wedge (\text{mat } (\text{rat-poly.row-length } B2) \ (\text{length } B2) \ B2)$
 $\wedge (A1 \neq []) \wedge (B1 \neq []) \wedge (B2 \neq [])$
 $\quad \wedge (\text{length } B1 = \text{rat-poly.row-length } B2)$

theorem $\text{weak-distributivity2}$:
 $\text{weak-matrix-match2 } A1 \ B1 \ B2$
 $\implies (A1 \otimes (\text{rat-poly.matrix-mult } B1 \ B2))$
 $\quad = (\text{rat-poly.matrix-mult } (A1 \otimes B1) \ (B2))$
 $\langle \text{proof} \rangle$

lemma $\text{is-tangle-diagram-weak-matrix-match}$:
assumes $\text{is-tangle-diagram } (w1 * ws1)$
and $\text{codomain-block } w2 = 0$
shows $\text{weak-matrix-match } (\text{blockmat } w1) \ (\text{kauff-mat } ws1) \ (\text{blockmat } w2)$
 $\langle \text{proof} \rangle$

lemma *is-tangle-diagram-weak-matrix-match2*:
assumes *is-tangle-diagram* ($w2 * ws2$)
and *codomain-block* $w1 = 0$
shows *weak-matrix-match2* (*blockmat* $w1$) (*blockmat* $w2$) (*kauff-mat* $ws2$)
 $\langle proof \rangle$

lemma *is-tangle-diagram-vert-block*:
 $is-tangle-diagram\ (b * (basic\ (make-vert-block\ (nat\ (codomain-block\ b))))))$
 $\langle proof \rangle$

The following theorem tells us that the the map *kauff_mat* when restricted to walls representing tangles preserves the tensor product

theorem *Tensor-Invariance*:
 $(is-tangle-diagram\ ws1) \wedge (is-tangle-diagram\ ws2)$
 $\implies (kauff-mat\ (ws1 \otimes ws2) = (kauff-mat\ ws1) \otimes (kauff-mat\ ws2))$
 $\langle proof \rangle$

end

12 Computations: This section can be skipped

theory *Computations*
imports *Kauffman-Matrix*
begin

lemma *unlink-computation*:
 $rat-poly-plus\ (rat-poly-times\ (rat-poly-times\ A\ A)\ (rat-poly-times\ A\ A))$
 $(rat-poly-plus$
 $(rat-poly-times\ 2\ (rat-poly-times\ A\ (rat-poly-times\ A\ (rat-poly-times\ B\ B))))$
 $(rat-poly-times\ (rat-poly-times\ B\ B)\ (rat-poly-times\ B\ B))) =$
 $((A^4) + (B^4) + 2)$
 $\langle proof \rangle$

lemma *computation-swingpos*:
 $rat-poly-plus\ (rat-poly-times\ B\ (rat-poly-times\ (A - rat-poly-times\ (rat-poly-times\ B\ B)\ B)\ B))$
 $(rat-poly-times\ (A - rat-poly-times\ (rat-poly-times\ B\ B)\ B)$
 $(rat-poly-times\ A\ (A - rat-poly-times\ (rat-poly-times\ B\ B)\ B))) =$
 $rat-poly-times\ A\ (rat-poly-times\ (A - rat-poly-times\ (rat-poly-times\ B\ B)\ B)\ A)$
 $(is\ ?l = ?r)$
 $\langle proof \rangle$

lemma *computation2*:

$\text{rat-poly-plus } (\text{rat-poly-times } A \ (\text{rat-poly-times } (B - \text{rat-poly-times } (\text{rat-poly-times } A \ A) \ A)))$
 $(\text{rat-poly-times } (B - \text{rat-poly-times } (\text{rat-poly-times } A \ A) \ A))$
 $(\text{rat-poly-times } B \ (B - \text{rat-poly-times } (\text{rat-poly-times } A \ A) \ A))) =$
 $\text{rat-poly-times } B \ (\text{rat-poly-times } (B - \text{rat-poly-times } (\text{rat-poly-times } A \ A) \ A) \ B)$
 $(\text{is } ?l = ?r)$
 $\langle \text{proof} \rangle$

lemma *computation-swingneg:rat-poly-times* $B \ (\text{rat-poly-times } (B - \text{rat-poly-times } (\text{rat-poly-times } A \ A) \ A) \ B) =$
 rat-poly-plus
 $(\text{rat-poly-times } (B - \text{rat-poly-times } (\text{rat-poly-times } A \ A) \ A))$
 $(\text{rat-poly-times } B \ (B - \text{rat-poly-times } (\text{rat-poly-times } A \ A) \ A)))$
 $(\text{rat-poly-times } A \ (\text{rat-poly-times } (B - \text{rat-poly-times } (\text{rat-poly-times } A \ A) \ A) \ A))$
 $A))$
 $\langle \text{proof} \rangle$

lemma *computation-toppos:rat-poly-inv* $(\text{rat-poly-times } (A - \text{rat-poly-times } (\text{rat-poly-times } B \ B) \ B) \ A) =$
 $\text{rat-poly-times } (B - \text{rat-poly-times } (\text{rat-poly-times } A \ A) \ A) \ B (\text{is } ?l = ?r)$
 $\langle \text{proof} \rangle$

lemma *computation-downpos-prelim:*
 $\text{rat-poly-inv } (\text{rat-poly-times } (B - \text{rat-poly-times } (\text{rat-poly-times } A \ A) \ A) \ B) =$
 $\text{rat-poly-times } (A - \text{rat-poly-times } (\text{rat-poly-times } B \ B) \ B) \ A (\text{is } ?l = ?r)$
 $\langle \text{proof} \rangle$

lemma *computation-downpos:rat-poly-times* $A \ (A - \text{rat-poly-times } (\text{rat-poly-times } B \ B) \ B) =$
 $\text{rat-poly-inv } (\text{rat-poly-times } B \ (B - \text{rat-poly-times } (\text{rat-poly-times } A \ A) \ A))$
 $\langle \text{proof} \rangle$

lemma *computation-positive-flip:rat-poly-plus*
 $(\text{rat-poly-inv } (\text{rat-poly-times } A \ (\text{rat-poly-times } (A - \text{rat-poly-times } (\text{rat-poly-times } B \ B) \ B) \ A)))$
 $(\text{rat-poly-inv } (\text{rat-poly-times } B \ (\text{rat-poly-times } A \ B))) =$
 $\text{rat-poly-inv } (\text{rat-poly-times } A \ (\text{rat-poly-times } A \ A)) (\text{is } ?l = ?r)$
 $\langle \text{proof} \rangle$

lemma *computation-negative-flip:rat-poly-plus*
 $(\text{rat-poly-inv } (\text{rat-poly-times } B \ (\text{rat-poly-times } (B - \text{rat-poly-times } (\text{rat-poly-times } A \ A) \ A) \ B)))$
 $(\text{rat-poly-inv } (\text{rat-poly-times } A \ (\text{rat-poly-times } B \ A))) =$
 $\text{rat-poly-inv } (\text{rat-poly-times } B \ (\text{rat-poly-times } B \ B)) (\text{is } ?l = ?r)$
 $\langle \text{proof} \rangle$

lemma *computation-pull-pos-neg*:

rat-poly-plus (*rat-poly-times* *B* (*B* - *rat-poly-times* (*rat-poly-times* *A* *A*) *A*))
 (*rat-poly-times* (*A* - *rat-poly-times* (*rat-poly-times* *B* *B*) *B*) *A*) = 0
 ⟨*proof*⟩

lemma *aux1*: (*A* - *rat-poly-times* (*rat-poly-times* *B* *B*) *B*)
 = *A* - (*B*^3)
 ⟨*proof*⟩

lemma *square-subtract*: (((*p*::*rat-poly*) - (*q*::*rat-poly*))^2)
 = (*p*^2) - (2**p***q*) + (*q*^2)
 ⟨*proof*⟩

lemma *cube-minus*: ∀ *p q*. (((*p*::*rat-poly*) - (*q*::*rat-poly*))^3)
 = (*p*^3) - 3*(*p*^2)*(*q*) + 3*(*p*)*(*q*^2) - (*q*^3)
 ⟨*proof*⟩

lemma *power-mult*: ((*p*::*rat-poly*)^*m*)^*n* = (*p*)^(*m**(*n*::*nat*))
 ⟨*proof*⟩

lemma *cube-minus2*:

fixes *p q*

shows (((*p*::*rat-poly*) - (*q*::*rat-poly*))^3)
 = (*p*^3) - 3*(*p*^2)*(*q*) + 3*(*p*)*(*q*^2) - (*q*^3)
 ⟨*proof*⟩

lemma *subst-poly*: **assumes** *a* = *b* **shows** (*p*::*rat-poly*)**a* = *p***b*
 ⟨*proof*⟩

lemma *sub1*:

assumes *p***q* = 1

shows *r**(*p***q*) = *r**1

⟨*proof*⟩

lemma *n-distrib*: (*A*^(*n*::*nat*))*(*B*^*n*) = (*A***B*)^*n*
 ⟨*proof*⟩

lemma *rat-poly-id-pow*: (1::*rat-poly*)^*n* = 1
 ⟨*proof*⟩

lemma *power-prod*: (*A*^(*n*::*nat*))*(*B*^*n*) = (1::*rat-poly*)
 ⟨*proof*⟩

lemma (*pCons* 0 1) ≠ 0
 ⟨*proof*⟩

end

13 Tangle moves and Kauffman bracket

theory *Linkrel-Kauffman*
imports *Computations*
begin

lemma *mat1-vert-wall-left:*
assumes *is-tangle-diagram b*
shows
 $\text{rat-poly.matrix-mult } (\text{blockmat } (\text{make-vert-block } (\text{nat } (\text{domain-wall } b)))) (\text{kauff-mat } b)$
 $= (\text{kauff-mat } b)$
 $\langle \text{proof} \rangle$

lemma *mat1-vert-wall-right:*
assumes *is-tangle-diagram b*
shows
 $\text{rat-poly.matrix-mult } (\text{kauff-mat } b) (\text{blockmat } (\text{make-vert-block } (\text{nat } (\text{codomain-wall } b))))$
 $= (\text{kauff-mat } b)$
 $\langle \text{proof} \rangle$

lemma *compress-top-inv:* $(\text{compress-top } w1 \ w2) \implies \text{kauff-mat } w1 = \text{kauff-mat } w2$
 $\langle \text{proof} \rangle$

lemma *domain-make-vert-int:* $(n \geq 0) \implies (\text{domain-block } (\text{make-vert-block } (\text{nat } n)))$
 $= n$
 $\langle \text{proof} \rangle$

lemma *compress-bottom-inv:* $(\text{compress-bottom } w1 \ w2) \implies \text{kauff-mat } w1 = \text{kauff-mat } w2$
 $\langle \text{proof} \rangle$

theorem *compress-inv:* $\text{compress } w1 \ w2 \implies (\text{kauff-mat } w1 = \text{kauff-mat } w2)$
 $\langle \text{proof} \rangle$

lemma *straghten-topdown-computation:* $\text{kauff-mat } ((\text{basic } ([\text{vert}, \text{cup}])) \circ (\text{basic } ([\text{cap}, \text{vert}])))$
 $= \text{kauff-mat } ((\text{basic } ([\text{vert}])) \circ (\text{basic } ([\text{vert}])))$
 $\langle \text{proof} \rangle$

theorem *straighten-topdown-inv:* $\text{straighten-topdown } w1 \ w2 \implies (\text{kauff-mat } w1 = \text{kauff-mat } w2)$
 $\langle \text{proof} \rangle$

lemma *straihten-downtop-computation*: $\text{kauff-mat } ((\text{basic } ([\text{cup}, \text{vert}])) \circ (\text{basic } ([\text{vert}, \text{cap}])))$
 $= \text{kauff-mat } ((\text{basic } ([\text{vert}])) \circ (\text{basic } ([\text{vert}])))$
 $\langle \text{proof} \rangle$

theorem *straighten-downtop-inv*: $\text{straighten-downtop } w1 \ w2 \implies (\text{kauff-mat } w1) =$
 $(\text{kauff-mat } w2)$
 $\langle \text{proof} \rangle$

theorem *straighten-inv*: $\text{straighten } w1 \ w2 \implies (\text{kauff-mat } w1) = (\text{kauff-mat } w2)$
 $\langle \text{proof} \rangle$

lemma *kauff-mat-swingpos*:
 $\text{kauff-mat } (r\text{-over-braid}) = \text{kauff-mat } (l\text{-over-braid})$
 $\langle \text{proof} \rangle$

lemma *swing-pos-inv*: $\text{swing-pos } w1 \ w2 \implies (\text{kauff-mat } w1) = (\text{kauff-mat } w2)$
 $\langle \text{proof} \rangle$

lemma *kauff-mat-swingneg*:
 $\text{kauff-mat } (r\text{-under-braid}) = \text{kauff-mat } (l\text{-under-braid})$
 $\langle \text{proof} \rangle$

lemma *swing-neg-inv*: $\text{swing-neg } w1 \ w2 \implies (\text{kauff-mat } w1) = (\text{kauff-mat } w2)$
 $\langle \text{proof} \rangle$

theorem *swing-inv*:
 $\text{swing } w1 \ w2 \implies (\text{kauff-mat } w1) = (\text{kauff-mat } w2)$
 $\langle \text{proof} \rangle$

lemma *rotate-toppos-kauff-mat*: $\text{kauff-mat } ((\text{basic } [\text{vert}, \text{over}]) \circ (\text{basic } [\text{cap}, \text{vert}])))$
 $= \text{kauff-mat } ((\text{basic } [\text{under}, \text{vert}]) \circ (\text{basic } [\text{vert}, \text{cap}])))$
 $\langle \text{proof} \rangle$

lemma *rotate-toppos-inv*: $\text{rotate-toppos } w1 \ w2 \implies (\text{kauff-mat } w1) = (\text{kauff-mat } w2)$
 $\langle \text{proof} \rangle$

lemma *rotate-topneg-kauff-mat*: $\text{kauff-mat } ((\text{basic } [\text{vert}, \text{under}]) \circ (\text{basic } [\text{cap}, \text{vert}])))$
 $= \text{kauff-mat } ((\text{basic } [\text{over}, \text{vert}]) \circ (\text{basic } [\text{vert}, \text{cap}])))$
 $\langle \text{proof} \rangle$

lemma *rotate-topneg-inv*: $\text{rotate-topneg } w1 \ w2 \implies (\text{kauff-mat } w1) = (\text{kauff-mat } w2)$
 $\langle \text{proof} \rangle$

lemma *rotate-downpos-kauff-mat:*

$\text{kauff-mat } ((\text{basic } [\text{cup}, \text{vert}]) \circ (\text{basic } [\text{vert}, \text{over}])) = \text{kauff-mat } ((\text{basic } [\text{vert}, \text{cup}]) \circ (\text{basic } [\text{under}, \text{vert}]))$
 $\langle \text{proof} \rangle$

lemma *rotate-downpos-inv: rotate-downpos w1 w2 $\implies$ (kauff-mat w1) = (kauff-mat w2)*

$\langle \text{proof} \rangle$

lemma *rotate-downneg-kauff-mat:*

$\text{kauff-mat } ((\text{basic } [\text{cup}, \text{vert}]) \circ (\text{basic } [\text{vert}, \text{under}])) = \text{kauff-mat } ((\text{basic } [\text{vert}, \text{cup}]) \circ (\text{basic } [\text{over}, \text{vert}]))$
 $\langle \text{proof} \rangle$

lemma *rotate-downneg-inv: rotate-downneg w1 w2 $\implies$ (kauff-mat w1) = (kauff-mat w2)*

$\langle \text{proof} \rangle$

theorem *rotate-inv: rotate w1 w2 $\implies$ (kauff-mat w1) = (kauff-mat w2)*

$\langle \text{proof} \rangle$

lemma *positive-flip-kauff-mat:*

$\text{kauff-mat } (\text{left-over}) = \text{kauff-mat } (\text{right-over})$
 $\langle \text{proof} \rangle$

lemma *uncross-positive-flip-inv: uncross-positive-flip w1 w2 $\implies$ (kauff-mat w1) = (kauff-mat w2)*

$\langle \text{proof} \rangle$

lemma *negative-flip-kauff-mat: kauff-mat (left-under) = kauff-mat (right-under)*

$\langle \text{proof} \rangle$

lemma *uncross-negative-flip-inv: uncross-negative-flip w1 w2 $\implies$ (kauff-mat w1) = (kauff-mat w2)*

$\langle \text{proof} \rangle$

theorem *framed-uncross-inv: (framed-uncross w1 w2) $\implies$ (kauff-mat w1) = (kauff-mat w2)*

$\langle \text{proof} \rangle$

lemma *pos-neg-kauff-mat*:
 $\text{kauff-mat } ((\text{basic } [\text{over}]) \circ (\text{basic } [\text{under}])))$
 $\quad = \text{kauff-mat } ((\text{basic } [\text{vert}, \text{vert}]) \circ (\text{basic } [\text{vert}, \text{vert}])))$
 $\langle \text{proof} \rangle$

lemma *pull-posneg-inv*: $\text{pull-posneg } w1 \ w2 \implies (\text{kauff-mat } w1) = (\text{kauff-mat } w2)$
 $\langle \text{proof} \rangle$

lemma *neg-pos-kauff-mat*: $\text{kauff-mat } ((\text{basic } [\text{under}]) \circ (\text{basic } [\text{over}])))$
 $\quad = \text{kauff-mat } ((\text{basic } [\text{vert}, \text{vert}]) \circ (\text{basic } [\text{vert}, \text{vert}])))$
 $\langle \text{proof} \rangle$

lemma *pull-negpos-inv*: $\text{pull-negpos } w1 \ w2 \implies (\text{kauff-mat } w1) = (\text{kauff-mat } w2)$
 $\langle \text{proof} \rangle$

theorem *pull-inv*: $\text{pull } w1 \ w2 \implies (\text{kauff-mat } w1) = (\text{kauff-mat } w2)$
 $\langle \text{proof} \rangle$

theorem *slide-inv*: $\text{slide } w1 \ w2 \implies (\text{kauff-mat } w1 = \text{kauff-mat } w2)$
 $\langle \text{proof} \rangle$

theorem *framed-linkrel-inv*: $\text{framed-linkrel } w1 \ w2 \implies (\text{kauff-mat } w1) = (\text{kauff-mat } w2)$
 $\langle \text{proof} \rangle$

end

14 Kauffman_Invariance: Proving the invariance of Kauffman Bracket

theory *Kauffman-Invariance*
imports *Link-Algebra Linkrel-Kauffman*
begin

In the following theorem, we prove that the kauffman matrix is invariant of framed link invariance

theorem *kauffman-invariance*: $(w1 :: \text{wall}) \sim_f w2 \implies \text{kauff-mat } w1 = \text{kauff-mat } w2$
 $\langle \text{proof} \rangle$

lemma *rat-poly-times* $A \ B = 1$
 $\langle \text{proof} \rangle$

we calculate kauffman bracket of a few links

kauffman bracket of an unknot with zero crossings

lemma *kauff-mat* (*[cup]* (basic [cap])*) = *[[-(A^2) - (B^2)]]*
 ⟨*proof*⟩

kauffman bracket of an a two component unlinked unknot with zero crossings

lemma *kauff-mat* (*[cup,cup]* (basic [cap,cap])*) = *[[((A^4)+(B^4)) + 2]]*
 ⟨*proof*⟩

definition *trefoil-polynomial::rat-poly*

where

trefoil-polynomial ≡

rat-poly-plus

(*rat-poly-times* (*rat-poly-times* A A)
 (*rat-poly-plus*
 (*rat-poly-times* B
 (*rat-poly-times* B
 (*rat-poly-times* (A - *rat-poly-times* (*rat-poly-times* B B) B)
 (*rat-poly-times* A A))))
 (*rat-poly-times* (A - *rat-poly-times* (*rat-poly-times* B B) B)
 (*rat-poly-plus* (*rat-poly-times* B (*rat-poly-times* B (*rat-poly-times* A A)))
 (*rat-poly-times* (A - *rat-poly-times* (*rat-poly-times* B B) B)
 (*rat-poly-times* (A - *rat-poly-times* (*rat-poly-times* B B) B)
 (*rat-poly-times* A A))))))
 (*rat-poly-plus*
 (*rat-poly-times* 2
 (*rat-poly-times* A
 (*rat-poly-times* A
 (*rat-poly-times* A
 (*rat-poly-times* A (*rat-poly-times* A (*rat-poly-times* B B))))))
 (*rat-poly-times* (*rat-poly-times* B B)
 (*rat-poly-times* B
 (*rat-poly-times* (A - *rat-poly-times* (*rat-poly-times* B B) B)
 (*rat-poly-times* B (*rat-poly-times* B B))))))

kauffman bracket of trefoil

lemma *trefoil*:

kauff-mat (*[cup,cup]*[vert,over,vert]*[vert,over,vert]*[vert,over,vert]*
**(basic [cap,cap])*)
 = *[[trefoil-polynomial]]*
 ⟨*proof*⟩

end

theory *Knot-Theory*

imports *Kauffman-Invariance Example*

begin

end