Knot Theory

T. V. H. Prathamesh

October 27, 2022

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 8
3 Tangle_Algebra: Tensor product of tangles and its properties 11
4 Definition of tensor product of walls 12
5 Properties of tensor product of tangles 12
6 Tangle_Moves: Defining moves on tangles 26
7 Link_Algebra: Defining equivalence of tangles and links 31
8 Showing equivalence of links: An example 34
9 Kauffman Matrix and Kauffman Bracket- Definitions and Properties 43
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 up to 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

A 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 o 65) where
concatenates-Nil: [] o ys = ys |
concatenates-Cons: ((x#xs)o ys) = x#(xs o ys)

lemma empty-concatenate: xs o Nil = xs
by (induction xs) (auto)
Associativity properties of Concatenation

**Lemma** left-right-associativity: $(x \otimes y) \otimes z = x \otimes (y \otimes z)$
by (induction $x$) (auto)

**Lemma** left-associativity: $(x \otimes y) \otimes z = x \otimes y \otimes z$
by (induction $x$) (auto)

**Lemma** right-associativity: $x \otimes (y \otimes z) = x \otimes y \otimes z$
by auto

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

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

Associativity properties of composition

**Lemma** compose-left-associativity: $((x \:: \text{wall}) \circ y) \circ z = (x \circ y) \circ z$
by (induction $x$) (auto)

**Lemma** compose-right-associativity: $(x \:: \text{wall}) \circ (y \circ z) = (x \circ y) \circ z$
by (induction $x$) (auto)

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

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

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

**Lemma** domain-non-negative: $\forall x. (\text{domain} \ x) \geq 0$
proof
have $\forall x. (x = \text{vert}) \lor (x = \text{over}) \lor (x = \text{under}) \lor (x = \text{cap}) \lor (x = \text{cup})$
by (metis brick.exhaust)
moreover have
∀ \( x. ((x = \text{vert}) \lor (x = \text{over}) \lor (x = \text{under}) \lor (x = \text{cap}) \lor (x = \text{cup})) \rightarrow (\text{domain } x) \geq 0 \)

using \( \text{domain.simps by (metis order-refl zero-le-numeral zero-le-one) }\)
ultimately show \( \text{thesis by auto} \)
qed

primrec \( \text{codomain::brick} \Rightarrow \text{int} \)
where
\( \text{codomain} \text{ vert} = 1 \mid \)
\( \text{codomain} \text{ cup} = 2 \mid \)
\( \text{codomain} \text{ cap} = 0 \mid \)
\( \text{codomain} \text{ over} = 2 \mid \)
\( \text{codomain} \text{ under} = 2 \)

primrec \( \text{domain-block::block} \Rightarrow \text{int} \)
where
\( \text{domain-block} [\text{[]} = 0 \mid \)
\( \text{domain-block} (\text{Cons } x y) = (\text{domain } x + (\text{domain-block } y)) \)

lemma \( \text{domain-block-non-negative} \text{::domain-block } xs \geq 0 \)
by (induction \( xs \)) (auto simp add: \( \text{domain-non-negative} \))

primrec \( \text{codomain-block::block} \Rightarrow \text{int} \)
where
\( \text{codomain-block} [\text{[]} = 0 \mid \)
\( \text{codomain-block} (\text{Cons } x y) = (\text{codomain } x + (\text{codomain-block } y)) \)

primrec \( \text{domain-wall:: wall} \Rightarrow \text{int} \text{ where} \)
\( \text{domain-wall} \text{ (basic } x) = \text{domain-block } x \mid \)
\( \text{domain-wall} (x\ast ys) = \text{domain-block } x \)

fun \( \text{codomain-wall:: wall} \Rightarrow \text{int} \text{ where} \)
\( \text{codomain-wall} \text{ (basic } x) = \text{codomain-block } x \mid \)
\( \text{codomain-wall} (x\ast ys) = \text{codomain-wall } ys \)

lemma \( \text{domain-wall-compose} \text{::domain-wall } (xs\circ ys) = \text{domain-wall } xs \)
by (induction \( xs \)) (auto)
lemma codomain-wall-compose: codomain-wall (xs◦ys) = codomain-wall ys
by (induction xs) (auto)

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

absolute value
definition abs::int ⇒ int where
abs x ≡ if (x≥0) then x else (0−x)

theorems about abs

lemma abs-zero: assumes abs x = 0 shows x = 0
using abs-def assms eq-iff-diff-eq-0
by metis

lemma abs-zero-equality: assumes abs (x−y) = 0 shows x = y
using assms abs-zero eq-iff-diff-eq-0
by blast

lemma abs-non-negative: abs x ≥ 0
using abs-def diff-0 le-cases neg-0-le-iff-le
by auto

lemma abs-non-negative-sum: assumes abs x + abs y = 0
shows abs x = 0 and abs y = 0
using abs-def diff-0 abs-non-negative neg-0-le-iff-le
add-nonneg-eq-0-iff assms
apply (metis)
by (metis abs-non-negative add-nonneg-eq-0-iff assms)

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) ≥ 0
using domain.simps brick.exhaust le-cases not-numeral-le-zero zero-le-one by (metis)

lemma codomain-nonnegative: (codomain x) ≥ 0
by (cases x)(auto)

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-block x ≥ 0
by (induction x)(auto simp add: domain-nonnegative)

lemma codomain-block-nonnegative: (codomain-block x) ≥ 0
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:** \(((\text{domain-block } (x\#\text{Nil})) > 0) \lor ((\text{domain-block } y) > 0)\)

\[\implies (\text{domain-block } (x\#y) > 0)\]

**Proof**
- **Have** \((\text{domain-block } (x\#y)) = (\text{domain } x) + (\text{domain-block } y)\) by auto
- **Also have** \((\text{domain } x) = (\text{domain-block } (x\#\text{Nil}))\) by auto
- **Then have** \((\text{domain-block } (x\#\text{Nil}) > 0) = (\text{domain } x > 0)\) by auto
- **Then have** \(((\text{domain } x > 0) \lor (\text{domain-block } y > 0)) \implies (\text{domain } x + \text{domain-block } y) > 0\)
  - Using **domain-nonnegative**, **add-nonneg-pos**, **add-pos-nonneg**, **domain-block-nonnegative**
    - By **metis**
  - From this
    - **Show** \(((\text{domain-block } (x\#\text{Nil})) > 0) \lor ((\text{domain-block } y) > 0)\)
      \[\implies (\text{domain-block } (x\#y) > 0)\]
    - By **auto**

**Lemma domain-additive:** \((\text{domain-block } (x\otimes y)) = (\text{domain-block } x) + (\text{domain-block } y)\)

**By** (induction x)(auto)

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

**By** (induction x)(auto)

**Lemma domain-zero-sum:** assumes \((\text{domain-block } x) + (\text{domain-block } y) = 0\)

**Shows** \((\text{domain-block } x = 0) \text{ and } \text{domain-block } y = 0\)

**Using** **domain-block-nonnegative**, **add-nonneg-eq-0-iff**, **assms**

**Apply** **metis**

**By** (metis **add-nonneg-eq-0-iff**, **assms** **domain-block-nonnegative**)

**Lemma domain-block-positive:** fixes or assumes \((\text{domain-block } y > 0) \text{ or domain-block } y \geq 0\)

**Shows** \((\text{domain-block } (x\otimes y)) > 0\)

**Apply** (simp add: **domain-additive**)

**By** (metis **assms**(1) **domain-additive**, **domain-block-nonnegative**, **domain-zero-sum**, **2**) (**less-le**)

**Lemma codomain-block-positive:** fixes or assumes \((\text{codomain-block } y > 0) \text{ or codomain-block } y \geq 0\)

**Shows** \((\text{codomain-block } (x\otimes y)) > 0\)

**Apply** (simp add: **codomain-additive**)

6
using assms(1) codomain-additive codomain-block-nonnegative eq-neg-iff-add-eq-0

le-less-trans less-le neg-less-0-iff-less
by (metis)

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 ⇒ 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 ⇒ bool
where
is-cup [] = True
is-cup (x#y) = (if (x = cup) then (is-cup y) else False)

lemma brickcount-zero-implies-cup:(domain x= 0) ⇒ (x = cup)
by (cases x) (auto)

lemma brickcount-zero-implies-brick-is-cup:(domain x= 0) ⇒ (brick-is-cup x)
by (cases x) (auto)

lemma domain-zero-implies-is-cup:(domain-block x= 0) ⇒ (is-cup x)
proof(induction x)
case Nil
  show ?case by auto
next
case (Cons a y)
  show ?case
  proof
    have step1: domain-block (a # y) = (domain a) + (domain-block y)
      by auto
    with domain-zero-sum havedomain-block y = 0
      by (metis (full-types) Cons.prems domain-block-nonnegative domain-positive leD neq-iff)
    then have step2: (is-cup y)
      using Cons.IH by (auto)
    with step1 and domain-zero-sum
    have domain a= 0
      using Cons.prems '(domain-block y = 0) by linarith
then have brick-is-cup a
  using brickcount-zero-implies-brick-is-cup by auto
then have a=cup
  using brick-is-cup-def by (metis ⟨domain a = 0⟩) brickcount-zero-implies-cup
with step2 have is-cup (a#y)
  using is-cup-def by auto
then show ?case by auto
qed
qed

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

primrec is-tangle-diagram::wall ⇒ 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 ⇒ bool
where
is-link-diagram x ≡ (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}  
by (rule-tac x = prod (cup#[])) (basic (cap#[])) in exI) (auto)

typedef Link-Diagram = {(x::wall), is-link-diagram x}  
by (rule-tac x = prod (cup#[])) (basic (cap#[])) in exI) (auto simp add:is-link-diagram-def abs-def)

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 $x$) = $x$

**Using** Rep-Tangle-Diagram Abs-Tangle-Diagram-inverse assms mem-Collect-eq **by** auto

The map Abs_diagram is injective

**Lemma Rep-Abs-well-defined:**

**Assumes** is-tangle-diagram $x$

**Shows** Rep-Abs-well-defined $x$

**Using** Rep-Tangle-Diagram Abs-Tangle-Diagram-inverse assms mem-Collect-eq **by** metis

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 $x$

**Proof** (induct $x$)

**Case** (basic $z$)

have is-tangle-diagram (basic $z$) **using** is-tangle-diagram.simps(1) **by** auto

then show $?case$ **using** basic **by** auto

next

**Case** (prod $z$ $zs$)

have $(z \ast zs) \circ y = (z \ast (zs \circ y))$ **by** auto

then have is-tangle-diagram $(z \ast (zs \circ y))$ **using** prod **by** auto

moreover then have 1: is-tangle-diagram $zs$

**Using** is-tangle-diagram.simps(2) prod.hyps prod.prems **by** metis

ultimately have domain-wall $(zs \circ y) =$ domain-wall $z$

**By** (metis is-tangle-diagram.simps(2))

moreover have domain-wall $(zs \circ y) =$ domain-wall $zs$

**Using** domain-wall-def domain-wall-compose **by** auto

ultimately have domain-wall $zs =$ domain-wall $z$

**By** (metis 1 is-tangle-diagram.simps(2))

then have is-tangle-diagram $(z \ast zs)$

**By** (metis 1 is-tangle-diagram.simps(2))

then show $?case$ **by** auto

**Qed**

**Lemma is-tangle-right-compose:**

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

9
proof (induct x)
case (basic z)
  have (basic z) o y = (z * y) using basic by auto
  then have is-tangle-diagram y
    unfolding is-tangle-diagram.simps(2) using basic.prems by (metis is-tangle-diagram.simps(2))
  then show ?case using basic.prems by auto
next
case (prod z zs)
  have ((z * zs) o y) = (z * (zs o y)) by auto
  then have is-tangle-diagram (z * (zs o y)) using prod by auto
  then have is-tangle-diagram (zs o y) using is-tangle-diagram.simps(2) by metis
  then have is-tangle-diagram y using prod.hyps by auto
  then show ?case by auto
qed

lemma comp-of-tangle-dgms:
assumes is-tangle-diagram y
shows ((is-tangle-diagram x)
\land(codomain-wall z = domain-wall y))
\implies is-tangle-diagram (z o y)
proof(induct x)
case (basic z)
  have codomain-block z = codomain-wall (basic z)
    using domain-wall-def by auto
  moreover have (basic z) o y = z * y
    using compose-def by auto
  ultimately have codomain-block z = domain-wall y
    using basic.prems by auto
  moreover have is-tangle-diagram y
    using assms by auto
  ultimately have is-tangle-diagram (z * y)
    unfolding is-tangle-diagram-def by auto
  then show ?case by auto
next
case (prod z zs)
  have is-tangle-diagram (z * zs)
    using prod.prems by metis
  then have codomain-block z = domain-wall zs
    using is-tangle-diagram.simps(2) prod.prems by metis
  then have codomain-block z = domain-wall (zs o y)
    using domain-wall.simps domain-wall-compose by auto
  moreover have is-tangle-diagram (zs o y)
    using prod.hyps by (metis codomain-wall.simps(2) is-tangle-diagram.simps(2)
    prod.prems)
  ultimately have is-tangle-diagram (z * (zs o y))
    unfolding is-tangle-diagram-def by auto
  then show ?case by auto

10
**qed**

**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 ◦ y)  
using comp-of-tangle-dgms using assms by auto

**lemma** converse-composition-of-tangle-diagrams:  
is-tangle-diagram (x ◦ y) ⇒ (domain-wall y) = (codomain-wall x)  
**proof**(induct x)  
**case** (basic z)  
**have** (basic z) ◦ y = z * y  
using compose-def basic by auto  
then **have**  
is-tangle-diagram ((basic z) ◦ y) ⇒  
(is-tangle-diagram y)∧ (codomain-block z = domain-wall y)  
using is-tangle-diagram.simps(2) by (metis)  
then **have** (codomain-block z) = (domain-wall y)  
using basic.prems by auto  
**moreover** **have** codomain-wall (basic z) = codomain-block z  
using domain-wall-compose by auto  
ultimately **have** (codomain-wall (basic z)) = (domain-wall y)  
by auto  
then **show** ?case by simp  
next  
**case** (prod z zs)  
**have** codomain-wall zs = domain-wall y  
using prod.hyps prod.prems  
by (metis compose-Nil compose-leftassociativity is-tangle-right-compose)  
**moreover** **have** codomain-wall zs = codomain-wall (z*zs)  
using domain-wall-compose by auto  
ultimately **show** ?case by metis  
**qed**

**definition** compose-Tangle:: Tangle-Diagram ⇒ Tangle-Diagram ⇒ Tangle-Diagram  
where  
compose-Tangle x y = Abs-Tangle-Diagram  
((Rep-Tangle-Diagram x) ◦ (Rep-Tangle-Diagram y))

**theorem** well-defined-compose:  
**assumes** is-tangle-diagram x  
and is-tangle-diagram y
and domain-wall \( x = \text{codomain-wall} \ y \)

shows \((\text{Abs-Tangle-Diagram} \ x) \circ (\text{Abs-Tangle-Diagram} \ y)\)

\[= (\text{Abs-Tangle-Diagram} \ (x \circ y))\]

using \(\text{Abs-Tangle-Diagram-inverse}\), \text{assms(1)} \text{assms(2)} \text{compose-Tangle-def mem-Collect-eq}

by auto

definition domain-Tangle::\text{Tangle-Diagram} \Rightarrow \text{int}
where
domain-Tangle \ x = \text{domain-wall}(\text{Rep-Tangle-Diagram} \ x)

definition codomain-Tangle::\text{Tangle-Diagram} \Rightarrow \text{int}
where
codomain-Tangle \ x = \text{codomain-wall}(\text{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:: \text{nat} \Rightarrow \text{block}
where
make-vert-block \ 0 = []
\text{|make-vert-block} \ (\text{Suc} \ n) = \text{vert#}(\text{make-vert-block} \ n)

lemma domain-make-vert: \text{domain-block} \ (\text{make-vert-block} \ n) = \text{int} \ n
by (induction \ n) (auto)

lemma codomain-make-vert: \text{codomain-block} \ (\text{make-vert-block} \ n) = \text{int} \ n
by (induction \ n) (auto)

fun tensor::\text{wall} => \text{wall} => \text{wall} \ (\text{infixr} \otimes 65)
where
1:tensor \ (\text{basic} \ x) \ (\text{basic} \ y) = (\text{basic} \ (x \otimes y))
\text{|2:tensor} \ (\text{x} \otimes \text{xs}) \ (\text{basic} \ y) = (}
if (codomain-block y = 0) 
then (x ⊗ y)∗xs 
else 
(x ⊗ y) 
ﬁrst(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 ⊗ (ys⊗(abs(make-vert-block(nat (codomain-block x))))) 
))

|4: tensor (x∗xs) (y*ys) = (x ⊗ (y∗ys))

5 Properties of tensor product of tangles

lemma Nil-left-tensor:xs ⊗ (basic ([])) = xs 
by (cases xs) (auto simp add:empty-concatenate)

lemma Nil-right-tensor:(basic ([])) ⊗ xs = xs 
by (cases xs) (auto)

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

definition tensor-Tangle :: Tangle-Diagram ⇒ Tangle-Diagram ⇒ Tangle-Diagram
(infixl ⊗ 65)
where

tensor-Tangle x y = Abs-Tangle-Diagram ((Rep-Tangle-Diagram x) ⊗ (Rep-Tangle-Diagram y))

lemma tensor (basic [vert]) (basic ([vert])) = (basic (([vert]) ⊗ ([vert])))
by simp

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(cases xs)
ﬁx x
assume A:xs = basic x
then have domain-wall (xs ⊗ ys) = domain-wall xs + domain-wall ys
proof(cases ys)
ﬁx y
assume B:ys = basic y
have domain-block (x ⊗ y) = domain-block x + domain-block y
using domain-additive by auto
then have domain-wall (xs ⊗ ys) = domain-wall xs + domain-wall ys
using \texttt{tensor.simps(1)} A B by auto
thus \texttt{thesis} by auto

next

fix \(z\) \(zs\)
assume \(C\) : \(ys = (z * zs)\)

have \(\text{domain-wall} (xs \otimes ys) = \text{domain-wall} xs + \text{domain-wall} ys\)
proof(cases (\text{codomain-block} x) = 0)
assume \(\text{codomain-block} x = 0\)
then have \((xs \otimes ys) = (x \otimes z) * zs\)
using \(A\ C\ \text{tensor.simps(4)}\) by auto
then have \(\text{domain-wall} (xs \otimes ys) = \text{domain-block} (x \otimes z)\)
by auto
moreover have \(\text{domain-wall} ys = \text{domain-block} z\)
unfolding \(\text{domain-wall-def} C\) by auto
moreover have \(\text{domain-wall} xs = \text{domain-block} x\)
unfolding \(\text{domain-wall-def} A\) by auto
moreover have \(\text{domain-block} (x \otimes z) = \text{domain-block} x + \text{domain-block} z\)
using \(\text{domain-additive}\) by auto
ultimately show \(\text{thesis}\) by auto

next

assume \(\text{codomain-block} x \neq 0\)

have \((xs \otimes ys) = (x \otimes z)\)
\*\*\*((\text{basic} (\text{make-vert-block} (\text{nat} (\text{codomain-block} x)))) \otimes zs)\)
using \(\text{tensor.simps(3)} A\ C\ \text{\langle codomain-block} x \neq 0\text{\rangle}\) by auto
then have \(\text{domain-wall} (xs \otimes ys) = \text{domain-block} (x \otimes z)\)
by auto
moreover have \(\text{domain-wall} ys = \text{domain-block} z\)
unfolding \(\text{domain-wall-def} C\) by auto
moreover have \(\text{domain-wall} xs = \text{domain-block} x\)
unfolding \(\text{domain-wall-def} A\) by auto
moreover have \(\text{domain-block} (x \otimes z) = \text{domain-block} x + \text{domain-block} z\)
using \(\text{domain-additive}\) by auto
ultimately show \(\text{thesis}\) by auto
qed
then show \(\text{thesis}\) by auto
qed
then show \(\text{thesis}\) by auto

next

fix \(z\) \(zs\)
assume \(D\) : \(xs = z * zs\)

then have \(\text{domain-wall} (xs \otimes ys) = \text{domain-wall} xs + \text{domain-wall} ys\)
proof(cases \(ys\))
fix \(y\)
assume \(E\) : \(ys = \text{basic} y\)
then have \(\text{domain-wall} (xs \otimes ys) = \text{domain-wall} xs + \text{domain-wall} ys\)
proof(cases \(\text{codomain-block} y = 0\))
assume \(\text{codomain-block} y = 0\)
have \((xs \otimes ys) = (z \otimes y) * zs\)
using \texttt{tensor.simps(2)} \( \text{D E \langle codomain-block y = 0\rangle by auto} \)
then have \texttt{domain-wall (xs \* ys) = domain-block (z \* y)} by auto
moreover have \texttt{domain-wall xs = domain-block z unfolding \texttt{domain-wall-def D by auto}}
moreover have \texttt{domain-wall ys = domain-block y unfolding \texttt{domain-wall-def E by auto}}
moreover have \texttt{domain-block (z \* y) = domain-block z + domain-block y using \texttt{domain-additive by auto}}
ultimately show \texttt{?thesis by auto} next
assume \texttt{codomain-block y} \( \neq \) \texttt{0}
have \texttt{(xs} \* \texttt{ys)} = 
\( (z \* y) \)
* \texttt{(zs} \* \texttt{(make-vert-block (nat (codomain-block y))))})
using \texttt{tensor.simps(3)} \( \text{D E \langle codomain-block y \neq 0\rangle by auto} \)
then have \texttt{domain-wall (xs \* ys) = domain-block (z \* y)} by auto
moreover have \texttt{domain-wall ys = domain-block y unfolding \texttt{domain-wall-def E by auto}}
moreover have \texttt{domain-wall xs = domain-block z unfolding \texttt{domain-wall-def D by auto}}
moreover have \texttt{domain-block (z \* y) = domain-block z + domain-block y using \texttt{domain-additive by auto}}
ultimately show \texttt{?thesis by auto} qed
then show \texttt{?thesis by auto} qed
next
fix \texttt{w ws}
assume \texttt{F:ys = w*ws}
have \texttt{(xs} \* \texttt{ys)} = \( (z \* w) \) * \texttt{(zs} \* \texttt{ws})
using \texttt{D F by auto}
then have \texttt{domain-wall (xs} \* \texttt{ys)} = \texttt{domain-block (z} \* \texttt{w)} by auto
moreover have \texttt{domain-wall ys = domain-block w unfolding \texttt{domain-wall-def F by auto}}
moreover have \texttt{domain-wall xs = domain-block z unfolding \texttt{domain-wall-def D by auto}}
moreover have \texttt{domain-block (z} \* \texttt{w)} = \texttt{domain-block z} + \texttt{domain-block w using \texttt{domain-additive by auto}}
ultimately show \texttt{?thesis by auto} qed
then show \texttt{?thesis by auto} qed

\text{codomain of tensor of two walls is the sum of the respective codomain’s is shown by the following theorem}

\textbf{lemma} \texttt{tensor-codomain-wall-additivity}: 

15
\begin{proof} \textbf{induction} $xs$ $ys$ \textbf{rule:} tensor.induct \\
\textbf{fix} $xs$ $ys$ \\
\textbf{let} $?case = (\text{codomain-wall } ((\text{basic } xs) \otimes (\text{basic } ys)))$

\begin{align*}
&= (\text{codomain-wall } (\text{basic } (xs))) \\
&+ (\text{codomain-wall } (\text{basic } (ys)))
\end{align*}

\textbf{show} $?case$ \textbf{using} codomain-wall.simps codomain-block.simps tensor.simps \\
\textbf{by} (metis codomain-additive) \\
\end{proof}

next \\
\textbf{fix} $x$ $xs$ $y$ \\
\textbf{assume} case-2: \\
\textbf{case} True \\
\textbf{have} $(x \cdot xs) \otimes (\text{basic } y) = (x \otimes y) \cdot xs$

\textbf{using} Tangle-Algebra.2 \textbf{True by} auto \\
\textbf{then have} \text{codomain-wall } ((x \cdot xs) \otimes (\text{basic } y)) = \text{codomain-wall } ((x \otimes y) \cdot xs)$

\textbf{by} auto \\
\textbf{then have} ...

\textbf{using} codomain-wall.simps \textbf{by} auto \\
\textbf{then have} ...

\textbf{using} True codomain-wall.simps(1) \textbf{by} auto \\
\textbf{then show} $?thesis$ \textbf{by} auto \\
next \\
\textbf{case} False \\
\textbf{have} $(x \cdot xs) \otimes (\text{basic } y)$

\begin{align*}
&= (x \otimes y) \\
&\cdot (x \otimes \cdot (\text{basic } (\text{make-vert-block } (\text{nat } (\text{codomain-block } y))))))
\end{align*}

\textbf{using} False \textbf{by} (metis Tangle-Algebra.2) \\
\textbf{moreover then have} \text{codomain-wall } ((x \cdot xs) \otimes (\text{basic } y)) = \text{codomain-wall}(...)$

\textbf{by} auto \\
\textbf{moreover have} ...

\begin{align*}
&= \text{codomain-wall} \\
&\cdot (x \otimes \cdot (\text{basic } (\text{make-vert-block } (\text{nat } (\text{codomain-block } y))))))
\end{align*}

\textbf{using} domain-wall.simps \textbf{by} auto \\
\textbf{moreover have} ...

\begin{align*}
&= \text{codomain-wall } xs \\
&+ \text{codomain-wall}
\end{align*}
(basic (make-vert-block (nat (codomain-block y))))

using case-2 False by auto

moreover have ... = codomain-wall (x*xs) + codomain-block y

using codomain-wall.simps

by (metis codomain-block-nonnegative codomain-make-vert int-nat-eq)

moreover have ... = codomain-wall (x*xs) + codomain-wall (basic y)

using codomain-wall.simps(1) by auto

ultimately show ?thesis by auto

qed

next

fix x y ys

assume case-3:(codomain-block x ≠ 0 →

codomain-wall

(basic (make-vert-block (nat (codomain-block x)))) ⊗ ys)

= codomain-wall

(basic (make-vert-block (nat (codomain-block x))))

+ codomain-wall ys)

let ?case = codomain-wall ((basic x) ⊗ (y*ys))

= codomain-wall (basic x) + codomain-wall (y*ys)

show ?case

proof(cases codomain-block x = 0)

case True

have (basic x)⊗(y*ys) = (x ⊗ y)*ys

using True 3 by auto

then have codomain-wall (...) = codomain-wall (...)

by auto

then have ... = codomain-wall ys

by auto

then have ... = codomain-wall ys + codomain-wall (basic x)

using codomain-wall.simps(1) True by auto

then show ?thesis by auto

next

case False

have (basic x) ⊗ (y*ys)

= (x ⊗ y)

*(((basic (make-vert-block (nat (codomain-block x))))⊗ ys)

using False 3 by auto

then have codomain-wall (...) = codomain-wall (...)

by auto

then have ...

= codomain-wall

(((basic (make-vert-block (nat (codomain-block x))))⊗ ys)

using codomain-wall.simps(2) by auto

then have ... = codomain-block x + codomain-wall ys

using codomain-wall.simps case-3 False

codomain-block-nonnegative codomain-make-vert int-nat-eq

by auto

then have ... = codomain-wall (basic x) + codomain-wall (y*ys)
using codomain-wall.simps by auto

then show ?thesis by (metis \langle basic \ x \otimes y \ast y \rangle \ast \langle basic (make-vert-block (nat (codomain-block x))) \otimes ys \rangle \ast \langle codomain-wall ((x \otimes y) \ast (basic (make-vert-block (nat (codomain-block x))) \otimes ys)) \ast \langle codomain-wall (basic (make-vert-block (nat (codomain-block x))) \otimes ys)) \ast \langle codomain-wall (basic (make-vert-block (nat (codomain-block x))) \otimes ys) = codomain-block x + codomain-wall ys \rangle \ast \langle basic \ x \otimes y \ast y \rangle \ast \langle basic \ x \otimes y \ast y \rangle = codomain-block x + codomain-wall ys \rangle
qed

next

fix \ x \ y \ x s \ y s

assume case-4 : codomain-wall \langle basic (make-vert-block (nat (codomain-block x))) \otimes y s \rangle = codomain-wall \langle basic (make-vert-block (nat (codomain-block x))) \otimes y s \rangle + codomain-wall \langle basic (make-vert-block (nat (codomain-block x))) \otimes y s \rangle

let ?case = codomain-wall \langle \langle x \ast x \rangle \otimes (y \ast y) \rangle

have \langle \langle x \ast x \rangle \otimes (y \ast y) \rangle = \langle x \ast y \rangle \ast \langle x \otimes y \rangle
using 4 by auto

then have codomain-wall (\langle \langle x \ast x \rangle \otimes (y \ast y) \rangle = codomain-wall (\langle x \ast x \rangle)
by auto

then have \langle \langle x \ast x \rangle \otimes (y \ast y) \rangle = codomain-wall (\langle x \ast x \rangle + codomain-wall (\langle y \ast y \rangle
using codomain-wall.simps(2) by auto

then have \langle \langle x \ast x \rangle \otimes (y \ast y) \rangle = codomain-wall (\langle x \ast x \rangle + codomain-wall (\langle y \ast y \rangle
using codomain-wall.simps(2) by auto

then show ?case by (metis \langle codomain-wall ((x \otimes y) \ast (xs \otimes ys)) = codomain-wall (xs \otimes ys) \ast \langle x \ast x \otimes y \ast y \rangle = (x \ast y) \ast (x \otimes y) \rangle \ast \langle basic (make-vert-block n) \rangle \ast \langle case-4 \rangle
qed

theorem is-tangle-make-vert-right:
(is-tangle-diagram xs) \implies is-tangle-diagram (\langle xs \otimes (basic (make-vert-block n)) \rangle

proof (induct xs)

case (basic xs)

show ?case by auto

next

case (prod x xs)

have ?case

proof (cases n)

case 0

have codomain-block (x \otimes (make-vert-block 0)) = \langle \langle x \otimes y \rangle \ast \langle basic (make-vert-block n) \rangle \ast \langle case-4 \rangle
using codomain-additive by auto

moreover have codomain-block (\langle make-vert-block 0 \rangle = 0
by auto

ultimately have codomain-block (x \otimes \langle make-vert-block 0 \rangle) = codomain-block (x)
by auto

18
moreover have is-tangle-diagram xs  
  using prod.prems by (metis is-tangle-diagram.simps(2))  
then have is-tangle-diagram ((x ⊗ (make-vert-block 0)) * xs)  
  using is-tangle-diagram.simps(2) by (metis calculation prod.prems)  
then have is-tangle-diagram ((x * xs) ⊗ (basic (make-vert-block 0)))  
  by auto  
then show ?thesis using 0 by (metis)  
next  
  case (Suc k)  
  have codomain-block (make-vert-block (k+1)) = int (k+1)  
    using codomain-make-vert by auto  
  then have (nat (codomain-block (make-vert-block (k+1)))) = k + 1  
    by auto  
  then have make-vert-block (nat (codomain-block (make-vert-block (k+1)))) = make-vert-block (k+1)  
    by auto  
  moreover have codomain-wall (basic (make-vert-block (k+1))) > 0  
    using make-vert-block.simps codomain-wall.simps Suc-eq-plus1 codomain-make-vert of-nat-0-less-iff zero-less-Suc by metis  
  ultimately have 1: (x * xs) ⊗ (basic (make-vert-block (k+1)))  
    = (x ⊗ (make-vert-block (k+1))) * (xs ⊗ (basic (make-vert-block (k+1))))  
    using tensor.simps(2) by simp  
  have domain-wall (xs ⊗ (basic (make-vert-block (k+1)))) = domain-wall xs + domain-wall (basic (make-vert-block (k+1)))  
    using tensor-domain-wall-additivity by auto  
  then have 2:  
    domain-wall (xs ⊗ (basic (make-vert-block (k+1)))) = (domain-wall xs) + int (k+1)  
    using domain-make-vert domain-wall.simps(1) by auto  
  moreover have 3: codomain-block (x ⊗ (make-vert-block (k+1))) = codomain-block x + int (k+1)  
    using codomain-additive codomain-make-vert by (metis)  
  have is-tangle-diagram (x * xs)  
    using prod.prems by auto  
  then have 4: codomain-block x = domain-wall xs  
    using is-tangle-diagram.simps(2) by metis  
from 2 3 4 have  
  domain-wall (xs ⊗ (basic (make-vert-block (k+1)))) = codomain-block (x ⊗ (make-vert-block (k+1)))  
  by auto  
  moreover have is-tangle-diagram (xs ⊗ (basic (make-vert-block (k+1))))  
    using prod.hyps prod.prems by (metis SucSuc-eq-plus1 is-tangle-diagram.simps(2))  
ultimately have is-tangle-diagram ((x * xs) ⊗ (basic (make-vert-block (k+1))))  
    using 1 by auto  
  then show ?thesis using Suc SucSuc-eq-plus1 by metis  
qed  
then show ?case by auto  
qed
**Theorem** \( \text{is-tangle-make-vert-left} : \)
\[
(\text{is-tangle-diagram } xs) \implies \text{is-tangle-diagram} \ ((\text{basic} \ (\text{make-vert-block} \ n)) \otimes xs)
\]
**Proof**
\[
\text{induct } xs
\]
\[
\text{case } (\text{basic} \ xs)
\]
\[
\text{show } ?\text{case} \text{ by auto}
\]
\[
\text{next}
\]
\[
\text{case } (\text{prod} \ x \ xs)
\]
\[
\text{have } ?\text{case}
\]
\[
\text{proof(} \text{cases } n \text{)}
\]
\[
\text{have } 0
\]
\[
\text{have codomain-block} \ ((\text{make-vert-block} \ 0) \otimes x)
\]
\[
= (\text{codomain-block} \ x) + \text{codomain-block}(\text{make-vert-block} \ 0)
\]
\[
\text{using codomain-additive by auto}
\]
\[
\text{moreover have} \ \text{codomain-block} \ (\text{make-vert-block} \ 0) = 0
\]
\[
\text{by auto}
\]
\[
\text{ultimately have codomain-block} \ ((\text{make-vert-block} \ 0) \otimes x) = \text{codomain-block} \ x
\]
\[
\text{by auto}
\]
\[
\text{moreover have} \ \text{is-tangle-diagram} \ xs
\]
\[
\text{using prod.prems by (metis is-tangle-diagram.simps(2))}
\]
\[
\text{then have} \ \text{is-tangle-diagram} \ ((\text{make-vert-block} \ 0) \otimes x)*xs
\]
\[
\text{using is-tangle-diagram.simps(2) by (metis calculation prod.prems)}
\]
\[
\text{then have} \ \text{is-tangle-diagram} \ ((\text{basic} \ (\text{make-vert-block} \ 0)) \otimes (x*xs))
\]
\[
\text{by auto}
\]
\[
\text{then show } ?\text{thesis using } 0 \text{ by (metis)}
\]
\[
\text{next}
\]
\[
\text{case } (\text{Suc} \ k)
\]
\[
\text{have codomain-block} \ ((\text{make-vert-block} \ (k+1)) \otimes x)
\]
\[
= (\text{nat}(\text{codomain-block} \ (\text{make-vert-block} \ (k+1))))
\]
\[
\text{by auto}
\]
\[
\text{then have} \ \text{make-vert-block} \ ((\text{nat}(\text{codomain-block} \ (\text{make-vert-block} \ (k+1)))))
\]
\[
= \text{make-vert-block} \ (k+1)
\]
\[
\text{by auto}
\]
\[
\text{moreover have} \ \text{codomain-wall} \ ((\text{basic} \ (\text{make-vert-block} \ (k+1)))*0
\]
\[
\text{using make-vert-block.simps codomain-wall.simps Suc-eq-plus1}
\]
\[
\text{codomain-make-vert of-nat-0-less-iff zero-less-Suc}
\]
\[
\text{by metis}
\]
\[
\text{ultimately have } 1: \ ((\text{basic} \ (\text{make-vert-block} \ (k+1)))*x*xs)
\]
\[
= ((\text{make-vert-block} \ (k+1)) \otimes x)*((\text{basic} \ (\text{make-vert-block} \ (k+1)))*xs)
\]
\[
\text{using tensor.simps(3) by simp}
\]
\[
\text{have domain-wall} \ ((\text{basic} \ (\text{make-vert-block} \ (k+1)))*xs)
\]
\[
= \text{domain-wall} \ xs + \text{domain-wall} \ ((\text{basic} \ (\text{make-vert-block} \ (k+1)))*xs)
\]
\[
\text{using tensor-domain-wall-additivity by auto}
\]
then have 2:
  \( \text{domain-wall} \left( (\text{basic} \ \text{make-vert-block} \ (k+1)) \otimes xs \right) \)
  \( = (\text{domain-wall} \ \text{xs}) + \text{int} \ (k+1) \)

  using \text{domain-make-vert} \ \text{domain-wall} \ \text{simps}(1) \ \text{by} \ \text{auto}

moreover have 3: \( \text{codomain-block} \left( \left( \text{make-vert-block} \ (k+1) \right) \otimes x \right) \)
  \( = \text{codomain-block} \ \text{xs} + \text{int} \ (k+1) \)

  using \text{codomain-additive} \ \text{codomain-make-vert}
  \ \text{by} \ (\text{simp add: codomain-additive})

have \text{is-tangle-diagram} \ (x*xs)
  \ \text{using} \ \text{prod}
  \ \text{prems} \ \text{by} \ \text{auto}

then have 4: \( \text{codomain-block} \ \text{xs} = \text{domain-wall} \ \text{xs} \)
  \ \text{using} \ \text{is-tangle-diagram}
  \ \text{simps} \ (2) \ \text{by} \ \text{metis}

from 2 3 4 have \( \text{domain-wall} \ ((\text{basic} \ \text{make-vert-block} \ (k+1)) \otimes xs) \)
  \( = \text{codomain-block} \ ((\text{make-vert-block} \ (k+1)) \otimes x) \)

  by \ \text{auto}

moreover have \text{is-tangle-diagram} \ ((\text{basic} \ \text{make-vert-block} \ (k+1)) \otimes xs)
  \ \text{using} \ \text{prod} \ \text{hyps} \ \text{prod} \ \text{prems} \ \text{by} \ (\text{metis Suc Suc-eq-plus1 is-tangle-diagram simps(2)})

ultimately have \text{is-tangle-diagram} \ ((\text{basic} \ \text{make-vert-block} \ (k+1)) \otimes (x*xs))
  \ \text{using} \ 1 \ \text{by} \ \text{auto}

then show ?thesis using \ \text{Suc Suc-eq-plus1} \ \text{by} \ \text{metis}

qed

then show ?case by \text{auto}

qed

lemma \text{simp1}: \( (\text{codomain-block} \ y) \neq 0 \implies \)
  \text{is-tangle-diagram} \ (xs)
  \ \land \ \text{is-tangle-diagram} \ ((\text{basic} \ \text{make-vert-block} \ (\text{nat} \ (\text{codomain-block} \ y)))) \implies 
  \text{is-tangle-diagram} \ (xs \otimes ((\text{basic} \ \text{make-vert-block} \ (\text{nat} \ (\text{codomain-block} \ y))))) \implies 
  (\text{is-tangle-diagram} \ (x * xs) \land \text{is-tangle-diagram} \ (\text{basic} \ y) \implies \text{is-tangle-diagram} \ (x * xs \otimes \text{basic} \ y))

proof
  assume \( A: \ (\text{codomain-block} \ y) \neq 0 \)

  assume \( B: \)
  \text{is-tangle-diagram} \ (xs)
  \ \land \ \text{is-tangle-diagram} \ ((\text{basic} \ \text{make-vert-block} \ (\text{nat} \ (\text{codomain-block} \ y))))
  \implies 
  \text{is-tangle-diagram} \ (xs \otimes ((\text{basic} \ \text{make-vert-block} \ (\text{nat} \ (\text{codomain-block} \ y)))))

  have \text{is-tangle-diagram} \ (x * xs) \land \text{is-tangle-diagram} \ (\text{basic} \ y) \implies \text{is-tangle-diagram} \ xs
    \ \by \ (\text{metis is-tangle-diagram simps(2)})

moreover have \text{is-tangle-diagram} \ ((\text{basic} \ \text{make-vert-block} \ (\text{nat} \ (\text{codomain-block} \ y))))
  \ \using \ \text{is-tangle-diagram simps(1) by auto}

ultimately have
\[(\text{is-tangle-diagram } xs) \wedge (\text{is-tangle-diagram } (\text{basic } (\text{make-vert-block } (\text{nat } (\text{codomain-block } y)))))) \rightarrow (\text{is-tangle-diagram } (xs \otimes \text{basic } (\text{make-vert-block } (\text{nat } (\text{codomain-block } y)))))) \rightarrow (\text{is-tangle-diagram } (xs \otimes \text{basic } (\text{make-vert-block } (\text{nat } (\text{codomain-block } y)))))) \rightarrow (\text{is-tangle-diagram } ((x \otimes xs) \otimes (\text{basic } ((y \otimes x) \otimes (xs \otimes (\text{basic } (\text{make-vert-block } (\text{nat } (\text{codomain-block } y)))))))))) \rightarrow (\text{is-tangle-diagram } (\text{basic } (\text{make-vert-block } (\text{nat } (\text{codomain-block } x)))))) \wedge (\text{is-tangle-diagram } (ys))\]

**Lemma simp2**: 
\[(\text{codomain-block } x) \neq 0 \Rightarrow (\text{is-tangle-diagram } (\text{basic } (\text{make-vert-block } (\text{nat } (\text{codomain-block } x)))))) \wedge (\text{is-tangle-diagram } (ys))\]
is-tangle-diagram \(((\text{basic } (\text{make-vert-block } (\text{nat } (\text{codomain-block } x)))) \otimes \text{ys})\)

\[ (\text{is-tangle-diagram } (\text{basic } x) \land \text{is-tangle-diagram } (\text{ys}) \rightarrow \text{is-tangle-diagram } ((\text{basic } x) \otimes (\text{ys}))) \]

**Proof**

Assume \( A: (\text{codomain-block } x) \neq 0 \)

Assume \( B: \text{is-tangle-diagram } (\text{basic } (\text{make-vert-block } (\text{nat } (\text{codomain-block } x)))) \land \text{is-tangle-diagram } (\text{ys}) \rightarrow \text{is-tangle-diagram } ((\text{basic } (\text{make-vert-block } (\text{nat } (\text{codomain-block } x)))) \otimes \text{ys}) \)

**Have** \( \text{is-tangle-diagram } (\text{basic } x) \land \text{is-tangle-diagram } (\text{ys}) \rightarrow \text{is-tangle-diagram } \text{ys} \)

By (metis is-tangle-diagram.simps(2))

Moreover have \( (\text{is-tangle-diagram } (\text{basic } (\text{make-vert-block } (\text{nat } (\text{codomain-block } x)))))) \)

Using is-tangle-diagram.simps(1) by auto

Ultimately have \( ((\text{is-tangle-diagram } \text{ys}) \land (\text{is-tangle-diagram } (\text{basic } (\text{make-vert-block } (\text{nat } (\text{codomain-block } x)))))) \)

\[ \rightarrow \text{is-tangle-diagram } ((\text{basic } (\text{make-vert-block } (\text{nat } (\text{codomain-block } x)))) \otimes \text{ys}) \]

\[ \rightarrow \text{is-tangle-diagram } (\text{basic } x) \land \text{is-tangle-diagram } (\text{ys} \rightarrow \text{is-tangle-diagram } ((\text{basic } (\text{make-vert-block } (\text{nat } (\text{codomain-block } x)))) \otimes \text{ys})) \]

By metis

Moreover have 1:codomain-block \( x \)

\[ = \text{domain-wall } (\text{basic } (\text{make-vert-block } (\text{nat } (\text{codomain-block } x)))) \]

Using codomain-block-nonnegative domain-make-vert domain-wall.simps(1)

int-nat-eq by auto

Moreover have 2:is-tangle-diagram \( (\text{basic } x) \land \text{is-tangle-diagram } (\text{ys} \rightarrow \text{domain-wall } \text{ys} \rightarrow \text{codomain-block } x = \text{domain-wall } \text{ys} \)

Using is-tangle-diagram.simps(2) by metis

Moreover have codomain-block \( (x \otimes y) = \text{codomain-block } x + \text{codomain-block } y \)

Using codomain-additive by auto

Moreover have domain-wall \(((\text{basic } (\text{make-vert-block } (\text{nat } (\text{codomain-block } x)))) \otimes \text{ys}) \)

\[ = \text{domain-wall } (\text{basic } (\text{make-vert-block } (\text{nat } (\text{codomain-block } x)))) + \text{domain-wall } \text{ys} \]

Using tensor-domain-wall-additivity by auto

Moreover then have is-tangle-diagram \( (\text{basic } x) \land \text{is-tangle-diagram } (\text{ys} \rightarrow \text{domain-wall } ((\text{basic } (\text{make-vert-block } (\text{nat } (\text{codomain-block } x)))) \otimes \text{ys}) \rightarrow \text{codomain-block } x \otimes y) \)

By (metis 1 2 calculation(4))

Ultimately have
\[(\text{is-tangle-diagram } ys) \land (\text{is-tangle-diagram } (\text{basic } (\text{make-vert-block } (\text{nat } (\text{codomain-block } x)))))) \rightarrow \text{is-tangle-diagram } ((\text{basic } (\text{make-vert-block } (\text{nat } (\text{codomain-block } x)))) \otimes ys)\]

\[\Rightarrow \text{is-tangle-diagram } (\text{basic } x) \land \text{is-tangle-diagram } (y \cdot ys)\]

\[\rightarrow \text{is-tangle-diagram } ((x \otimes y) \cdot ((\text{basic } (\text{make-vert-block } (\text{nat } (\text{codomain-block } x)))))) \otimes ys)\]

using \text{is-tangle-diagram.simps}(2) by auto

then have
\[\text{is-tangle-diagram } (\text{basic } x) \land \text{is-tangle-diagram } (y \cdot ys) \rightarrow \text{is-tangle-diagram } ((\text{basic } x) \otimes (y \cdot ys))\]

by (metis Tangle-Algebra.3 A B)

then show ?thesis by auto

qed

lemma simp3:
\[\text{is-tangle-diagram } xs \land \text{is-tangle-diagram } ys \rightarrow \text{is-tangle-diagram } (xs \otimes ys)\]

\[\Rightarrow \text{is-tangle-diagram } (x \cdot xs) \land \text{is-tangle-diagram } (y \cdot ys)\]

\[\rightarrow \text{is-tangle-diagram } (x \cdot xs \otimes y \cdot ys)\]

proof

assume A: \[\text{is-tangle-diagram } xs \land \text{is-tangle-diagram } ys \rightarrow \text{is-tangle-diagram } (xs \otimes ys)\]

have \[\text{is-tangle-diagram } (x \cdot xs) \rightarrow (\text{codomain-block } x = \text{domain-wall } xs)\]

using \text{is-tangle-diagram.simps}(2) by auto

moreover have \[\text{is-tangle-diagram } (y \cdot ys) \rightarrow (\text{codomain-block } y = \text{domain-wall } ys)\]

using \text{is-tangle-diagram.simps}(2) by auto

ultimately have \[\text{is-tangle-diagram } (x \cdot xs) \land \text{is-tangle-diagram } (y \cdot ys) \rightarrow \text{is-tangle-diagram } (xs \otimes ys)\]

using \text{codomain-additive tensor-domain-wall-additivity} by auto

moreover have \[\text{is-tangle-diagram } (x \cdot xs) \land \text{is-tangle-diagram } (y \cdot ys) \rightarrow \text{is-tangle-diagram } (xs \otimes ys)\]

using A \text{is-tangle-diagram.simps}(2) by auto

moreover have \[\text{is-tangle-diagram } (x \cdot xs) \otimes (y \cdot ys) = (x \otimes y) \cdot (xs \otimes ys)\]

using \text{tensor.simps}(4) by auto

ultimately have \[\text{is-tangle-diagram } (x \cdot xs) \land \text{is-tangle-diagram } (y \cdot ys) \rightarrow \text{is-tangle-diagram } ((x \cdot xs) \otimes (y \cdot ys))\]

by auto

then show ?thesis by simp

qed

theorem is-tangle-diagramness:

shows \((\text{is-tangle-diagram } x) \land (\text{is-tangle-diagram } y) \rightarrow \text{is-tangle-diagram } (\text{tensor } x \ y)\)
proof (induction x y rule: tensor.induct)

fix z w
let ?case = (is-tangle-diagram (basic z)) ∧ (is-tangle-diagram (basic w)) → is-tangle-diagram ((basic z) ⊗ (basic w))
show ?case by auto

next
fix x xs y
let ?case = (is-tangle-diagram (x * xs)) ∧ (is-tangle-diagram (basic y)) → is-tangle-diagram ((x * xs) ⊗ (basic y))
from simp1 show ?case
  by (metis Tangle-Algebra.2 add.commute codomain-additive comm-monoid-add-class.add-0
  is-tangle-diagram.simps(2) is-tangle-make-vert-right)

next
fix x y xs ys
assume A: is-tangle-diagram xs ∧ is-tangle-diagram ys → is-tangle-diagram (xs ⊗ ys)
let ?case = (is-tangle-diagram (x * xs)) ∧ (is-tangle-diagram (y * ys)) → is-tangle-diagram ((x * xs) ⊗ (y * ys))
from simp3 show ?case using A by auto
qed

theorem tensor-preserves-is-tangle:
assumes is-tangle-diagram x
and is-tangle-diagram y
shows is-tangle-diagram (x ⊗ y)
using assms is-tangle-diagramness by auto

definition Tensor-Tangle:: Tangle-Diagram ⇒ Tangle-Diagram ⇒ Tangle-Diagram

(infixl ◦ 65)
where
Tensor-Tangle x y = Abs-Tangle-Diagram ((Rep-Tangle-Diagram x) ⊗ (Rep-Tangle-Diagram y))

theorem well-defined-compose:
assumes is-tangle-diagram x
and is-tangle-diagram y
shows (Abs-Tangle-Diagram x) ⊗ (Abs-Tangle-Diagram y) = (Abs-Tangle-Diagram x y)
(x ⊗ y))
  using Abs-Tangle-Diagram-inverse assms(1) assms(2)
  mem-Collect-eq tensor-preserves-is-tangle
  tensor-Tangle-def
  by auto

end
theory Tangle-Relation
imports Main
begin

lemma symmetry1: assumes symp R
shows ∀ x y. (x, y) ∈ ∪{(x, y). R x y}∗ → (y, x) ∈ ∪{(x, y). R x y}∗
proof –
  have R x y → R y x by (metis assms sympD)
  then have (x, y) ∈ ∪{(x, y). R x y} → (y, x) ∈ ∪{(x, y). R x y} by auto
  then have 2: ∀ x y. (x, y) ∈ ∪{(x, y). R x y} → (y, x) ∈ ∪{(x, y). R x y}
  by (metis (full-types) assms mem-Collect-eq split-conv sympE)
  then have sym ∪{(x, y). R x y} unfolding sym-def by auto
  then have 3: sym (rtrancl ∪{(x, y). R x y}) using sym-rtrancl by auto
  then show ?thesis by (metis symE)
qed

lemma symmetry2: assumes ∀ x y. (x, y) ∈ ∪{(x, y). R x y}∗ → (y, x) ∈ ∪{(x, y). R x y}∗
shows symp R^∗∗
unfolding symp-def Enum.rtranclp-rtrancl-eq assms by (metis assms)

lemma symmetry3: assumes symp R shows symp R^∗∗ using assms symmetry1 symmetry2 by metis

lemma symm-trans: assumes symp R shows symp R^++ by (metis assms rtranclpD symmetry3 symp-def trancp-into-rtrancp)

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, addition 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 practise are proved as theorems in

\textbf{type-synonym} \begin{center} \textit{relation} = \textit{wall} \Rightarrow \textit{wall} \Rightarrow \textit{bool} \end{center}

\textit{Link uncross}

\textbf{abbreviation} \texttt{right-over::wall} where \begin{align*}
\text{right-over} & \equiv ((\text{basic} \ [\text{vert},\text{cup}]) \circ (\text{basic} \ [\text{over},\text{vert}]) \circ (\text{basic} \ [\text{vert},\text{cap}]))
\end{align*}

\textbf{abbreviation} \texttt{left-over::wall} where \begin{align*}
\text{left-over} & \equiv ((\text{basic} \ (\text{cup}#\text{vert}#[])) \circ (\text{basic} \ (\text{vert}#\text{over}#[])) \circ (\text{basic} \ (\text{cap}#\text{vert}#[])))
\end{align*}

\textbf{abbreviation} \texttt{right-under::wall} where \begin{align*}
\text{right-under} & \equiv ((\text{basic} \ (\text{vert}#\text{cup}#[])) \circ (\text{basic} \ (\text{under}#\text{vert}#[])) \circ (\text{basic} \ (\text{vert}#\text{cap}#[])))
\end{align*}

\textbf{abbreviation} \texttt{left-under::wall} where \begin{align*}
\text{left-under} & \equiv ((\text{basic} \ (\text{cup}#\text{vert}#[])) \circ (\text{basic} \ (\text{vert}#\text{under}#[])) \circ (\text{basic} \ (\text{cap}#\text{vert}#[])))
\end{align*}

\textbf{abbreviation} \texttt{straight-line::wall} where \begin{align*}
\text{straight-line} & \equiv (\text{basic} \ (\text{vert}#[])) \circ (\text{basic} \ (\text{vert}#[])) \circ (\text{basic} \ (\text{vert}#[]))
\end{align*}

\textbf{definition} \texttt{uncross-positive-flip::relation} where \begin{align*}
\text{uncross-positive-flip} \ x \ y & \equiv (x = \text{right-over}) \land (y = \text{left-over})
\end{align*}

\textbf{definition} \texttt{uncross-positive-straighten::relation} where
uncross-positive-straighten \( x \) \( y \) \( \equiv ((x = \text{right-over}) \land (y = \text{straight-line})) \)

**Definition** \( \text{uncross-negative-flip}::\text{relation} \)
where
\( \text{uncross-negative-flip} \ x \ y \ \equiv ((x = \text{right-under}) \land (y = \text{left-under})) \)

**Definition** \( \text{uncross-negative-straighten}::\text{relation} \)
where
\( \text{uncross-negative-straighten} \ x \ y \ \equiv ((x = \text{left-under}) \land (y = \text{straight-line})) \)

**Definition** \( \text{uncross} \)
where
\( \text{uncross} \ x \ y \ \equiv ((\text{uncross-positive-straighten} \ x \ y) \lor (\text{uncross-positive-flip} \ x \ y) \lor (\text{uncross-negative-straighten} \ x \ y) \lor (\text{uncross-negative-flip} \ x \ y)) \)

**Swing Begins**

**Abbreviation** \( \text{r-over-braid}::\text{wall} \)
where
\( \text{r-over-braid} \ \equiv \ ((\text{basic} ((\text{over}\#\text{vert}\#[]) \circ (\text{basic} ((\text{over}\#\text{vert}\#[])))) \circ (\text{basic} (\text{over}\#\text{vert}\#[])))) \)

**Abbreviation** \( \text{l-over-braid}::\text{wall} \)
where
\( \text{l-over-braid} \ \equiv \ ((\text{basic} (\text{vert}\#\text{over}\#[]) \circ (\text{basic} (\text{over}\#\text{vert}\#[])))) \circ (\text{basic} (\text{over}\#\text{vert}\#[]))) \)

**Abbreviation** \( \text{r-under-braid}::\text{wall} \)
where
\( \text{r-under-braid} \ \equiv \ ((\text{basic} ((\text{under}\#\text{vert}\#[]) \circ (\text{basic} ((\text{vert}\#\text{under}\#[])))) \circ (\text{basic} (\text{vert}\#\text{under}\#[]))) \)

**Abbreviation** \( \text{l-under-braid}::\text{wall} \)
where
\( \text{l-under-braid} \ \equiv \ ((\text{basic} (\text{vert}\#\text{under}\#[]) \circ (\text{basic} (\text{under}\#\text{vert}\#[])))) \circ (\text{basic} (\text{vert}\#\text{under}\#[]))) \)

**Definition** \( \text{swing-pos}::\text{wall} \Rightarrow \text{wall} \Rightarrow \text{bool} \)
where
\( \text{swing-pos} \ x \ y \ \equiv (x = \text{r-over-braid}) \land (y = \text{l-over-braid}) \)

**Definition** \( \text{swing-neg}::\text{wall} \Rightarrow \text{wall} \Rightarrow \text{bool} \)
where
\( \text{swing-neg} \ x \ y \ \equiv (x = \text{r-under-braid}) \land (y = \text{l-under-braid}) \)
definition \text{swing}::\text{relation}
where
\text{swing } x y \equiv (\text{swing-pos } x y) \lor (\text{swing-neg } x y)

pull begins
definition \text{pull-posneg}::\text{relation}
where
\text{pull-posneg } x y \equiv ((x = ((\text{basic (over#[])}) \circ \text{basic (under#[]}))) \\
\land (y = ((\text{basic (vert#vert#[]})) \\
\circ (\text{basic (vert#vert#[]}))))

definition \text{pull-negpos}::\text{relation}
where
\text{pull-negpos } x y \equiv ((x = ((\text{basic (under#[]})) \circ \text{basic (over#[]}))) \\
\land (y = ((\text{basic (vert#vert#[]})) \\
\circ (\text{basic (vert#vert#[]}))))

pull definition
definition \text{pull}::\text{relation}
where
\text{pull } x y \equiv ((\text{pull-posneg } x y) \lor (\text{pull-negpos } x y))

linkrel-pull ends

linkrel-straighten
definition \text{straighten-topdown}::\text{relation}
where
\text{straighten-topdown } x y \equiv ((x = ((\text{basic (vert#cup#[]}))) \\
\circ (\text{basic (cap#vert#[]}))) \\
\land (y = ((\text{basic (vert#[]})) \circ \text{basic (vert#[]}))))

definition \text{straighten-downtop}::\text{relation}
where
\text{straighten-downtop } x y \equiv ((x = ((\text{cap#vert#[]}))) \\
\circ (\text{basic (cap#vert#[]}))) \\
\land (y = ((\text{basic (vert#[]})) \circ \text{basic (vert#[]}))))

definition \text{straighten}
where
\text{straighten } x y \equiv ((\text{straighten-topdown } x y) \lor (\text{straighten-downtop } x y))

straighten ends

rotate moves
definition \text{rotate-topp}::\text{relation}
where  
rotate-toppos \ x \ y \equiv ((x = ((\text{basic } ((\text{vert} \ #\ \text{over} \ #[])))) \\
    \circ (\text{basic } ((\text{cap} \ #\ \text{vert} \ #[])))))) \\
\land (y = ((\text{basic } ((\text{under} \ #\ \text{vert} \ #[])))) \\
    \circ (\text{basic } ((\text{vert} \ #\ \text{cap} \ #[]))))))

definition rotate-topneg::\text{wall} \Rightarrow \text{wall} \Rightarrow \text{bool} 
where  
rotate-topneg \ x \ y \equiv ((x = ((\text{basic } ((\text{vert} \ #\ \text{under} \ #[])))) \\
    \circ (\text{basic } ((\text{cap} \ #\ \text{vert} \ #[])))))) \\
\land (y = ((\text{basic } ((\text{over} \ #\ \text{vert} \ #[])))) \\
    \circ (\text{basic } ((\text{vert} \ #\ \text{cap} \ #[]))))))

definition rotate-downpos::\text{wall} \Rightarrow \text{wall} \Rightarrow \text{bool} 
where  
rotate-downpos \ x \ y \equiv ((x = ((\text{basic } (\text{cup} \ #\ \text{vert} \ #[])))) \\
    \circ (\text{basic } ((\text{vert} \ #\ \text{over} \ #[])))))) \\
\land (y = ((\text{basic } ((\text{vert} \ #\ \text{cup} \ #[])))) \\
    \circ (\text{basic } ((\text{vert} \ #\ \text{under} \ #[]))))))

definition rotate-downneg::\text{wall} \Rightarrow \text{wall} \Rightarrow \text{bool} 
where  
rotate-downneg \ x \ y \equiv ((x = ((\text{basic } (\text{cup} \ #\ \text{vert} \ #[])))) \\
    \circ (\text{basic } ((\text{vert} \ #\ \text{over} \ #[])))))) \\
\land (y = ((\text{basic } ((\text{vert} \ #\ \text{cup} \ #[])))) \\
    \circ (\text{basic } ((\text{over} \ #\ \text{vert} \ #[]))))))

definition \text{rotate}::\text{wall} \Rightarrow \text{wall} \Rightarrow \text{bool} 
where  
\text{rotate} \ x \ y \equiv ((\text{rotate-toppos} \ x \ y) \lor (\text{rotate-topneg} \ x \ y) \\
\lor (\text{rotate-downpos} \ x \ y) \lor (\text{rotate-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::\text{wall} \Rightarrow \text{wall} \Rightarrow \text{bool} 
where  
compress-top1 \ x \ y \equiv \exists B.( (x = (\text{basic } (\text{make-vert-block} \ (\text{nat} \ (\text{domain-wall} \ B)))) \circ B) \\
\land (y = B) \land (\text{codomain-wall} \ B \neq 0) \\
\land (\text{is-tangle-diagram} \ B))
definition compress-bottom1::wall ⇒ wall ⇒ bool
where
compress-bottom1 x y ≡ ∃ B.((x = B ◦ (basic (make-vert-block (nat (codomain-wall B)))))
∧ (y = B)) ∧ (domain-wall B ≠ 0)
∧ (is-tangle-diagram B)

definition compress-bottom::wall ⇒ wall ⇒ bool
where
compress-bottom x y ≡ ∃ B.((x = B ◦ (basic (make-vert-block (nat (codomain-wall B)))))
∧ (y = (basic ([])) ◦ B)) ∧ (domain-wall B = 0)
∧ (is-tangle-diagram B)

definition compress-top::wall ⇒ wall ⇒ bool
where
compress-top x y ≡ ∃ B.((x = (basic (make-vert-block (nat (domain-wall B))))) ◦
B)
∧ (y = (B ◦ (basic ([])))) ∧ (codomain-wall B = 0)
∧ (is-tangle-diagram B)

definition compress::wall ⇒ wall ⇒ bool
where
compress x y = ((compress-top x y) ∨ (compress-bottom x y))

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

definition slide::wall ⇒ wall ⇒ bool
where
slide x y ≡ ∃ B.((x = (basic (make-vert-block (nat (domain-block B))))) ◦ (basic B))
∧ (y = ((basic B) ◦ (basic (make-vert-block (nat (domain-block B)))))
∧ ((domain-block B) ≠ 0))

linkrel-definition

definition linkrel::wall =⇒ wall ⇒ bool
where
linkrel x y = (((uncross x y) ∨ (pull x y) ∨ (straighten x y)
∨ (swing x y) ∨ (rotate x y) ∨ (compress x y) ∨ (slide x y))

definition framed-uncross::wall ⇒ wall ⇒ bool
where
framed-uncross x y ≡ ((uncross-positive-flip x y) ∨ (uncross-negative-flip x y))

definition framed-linkrel::wall =⇒ wall ⇒ bool
where
framed-linkrel \(x\ y\) = ((framed-uncross \(x\ y\)) \lor (pull \(x\ y\)) \lor (straighten \(x\ y\))
\lor(swing \(x\ y\)) \lor (rotate \(x\ y\)) \lor (compress \(x\ y\)) \lor (slide \(x\ y\)))

lemma framed-uncross-implies-uncross: (framed-uncross \(x\ y\)) \implies (uncross \(x\ y\))
by (auto simp add: framed-uncross-def uncross-def)

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

32
compose-eq:\((B::\text{wall}) \sim f D) \land ((A::\text{wall}) \sim f C) \\
\land (\text{is-tangle-diagram } A) \land (\text{is-tangle-diagram } B) \\
\land (\text{is-tangle-diagram } C) \land (\text{is-tangle-diagram } D) \\
\land (\text{domain-wall } B) = (\text{codomain-wall } A) \\
\land (\text{domain-wall } D) = (\text{codomain-wall } C) \\
\implies (A::\text{wall}) \circ (B::\text{wall}) \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::\text{wall}) \sim f D) \land ((A::\text{wall}) \sim f C) \land (\text{is-tangle-diagram } A) \land (\text{is-tangle-diagram } B) \\
\land (\text{is-tangle-diagram } C) \land (\text{is-tangle-diagram } D) \implies ((A::\text{wall}) \otimes B) \sim f (C \otimes D) \\

\text{definition } \text{Tangle-Diagram-Equivalence}: \text{Tangle-Diagram} \Rightarrow \text{Tangle-Diagram} \Rightarrow \text{bool} \\
\text{infixl} \sim T 64 \\
\text{where} \\
\text{Tangle-Diagram-Equivalence } T1 T2 \equiv (\text{Rep-Tangle-Diagram } T1) \sim (\text{Rep-Tangle-Diagram } T2) \\

\text{definition } \text{Link-Diagram-Equivalence}: \text{Link-Diagram} \Rightarrow \text{Link-Diagram} \Rightarrow \text{bool} \\
\text{infixl} \sim L 64 \\
\text{where} \\
\text{Link-Diagram-Equivalence } T1 T2 \equiv (\text{Rep-Link-Diagram } T1) \sim (\text{Rep-Link-Diagram } T2) \\

\text{quotient-type } \text{Tangle} = \text{Tangle-Diagram}/\text{Tangle-Diagram-Equivalence} \\
\text{morphisms } \text{Rep-Tangles} \text{ Abs-Tangles} \\
\text{proof } (\text{rule equivpI}) \\
\text{show reflp } \text{Tangle-Diagram-Equivalence} \\
\text{unfolding } \text{reflp-def } \text{Tangle-Diagram-Equivalence-def} \\
\text{Tangle-Equivalence. refl} \\
\text{by auto} \\
\text{show symp } \text{Tangle-Diagram-Equivalence} \\
\text{unfolding } \text{Tangle-Diagram-Equivalence-def symp-def} \\
\text{using } \text{Tangle-Diagram-Equivalence-def Tangle-Equivalence.sym} \\
\text{by auto} \\
\text{show transp } \text{Tangle-Diagram-Equivalence} \\
\text{unfolding } \text{Tangle-Diagram-Equivalence-def transp-def} \\
\text{using } \text{Tangle-Diagram-Equivalence-def Tangle-Equivalence.trans} \\
\text{by metis} \\
\text{qed} \\

\text{quotient-type } \text{Link} = \text{Link-Diagram}/\text{Link-Diagram-Equivalence} \\
\text{morphisms } \text{Rep-Links} \text{ Abs-Links} \\
\text{proof } (\text{rule equivpI}) \\
\text{show reflp } \text{Link-Diagram-Equivalence} \\
\text{unfolding } \text{reflp-def } \text{Link-Diagram-Equivalence-def} \\
\text{Tangle-Equivalence. refl} \\
\text{by auto} \\
\text{show symp } \text{Link-Diagram-Equivalence} \\
\text{unfolding } \text{Link-Diagram-Equivalence-def symp-def} \\
\text{using } \text{Link-Diagram-Equivalence-def Tangle-Equivalence.sym} \\
\text{by auto} \\
\text{show transp } \text{Link-Diagram-Equivalence} \\
\text{unfolding } \text{Link-Diagram-Equivalence-def transp-def} \\
\text{using } \text{Link-Diagram-Equivalence-def Tangle-Equivalence.trans} \\
\text{by metis} \\
\text{qed}
by auto

show symp Link-Diagram-Equivalence
  unfolding Link-Diagram-Equivalence-def symp-def
  using Link-Diagram-Equivalence-def Tangle-Equivalence.sym
by auto

show transp Link-Diagram-Equivalence
  unfolding Link-Diagram-Equivalence-def transp-def
  using Link-Diagram-Equivalence-def Tangle-Equivalence.trans
by metis

qed

definition Framed-Tangle-Diagram-Equivalence:: Tangle-Diagram ⇒ Tangle-Diagram ⇒ bool
  (infixl ∼ T 64)
where
Framed-Tangle-Diagram-Equivalence T1 T2
  ≡ (Rep-Tangle-Diagram T1) ∼ (Rep-Tangle-Diagram T2)

definition Framed-Link-Diagram-Equivalence:: Link-Diagram ⇒ Link-Diagram ⇒ 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 (rule equivpI)
show reflp Framed-Tangle-Diagram-Equivalence
  unfolding reflp-def Framed-Tangle-Diagram-Equivalence-def
  Framed-Tangle-Equivalence.refl
by auto

show symp Framed-Tangle-Diagram-Equivalence
  unfolding symp-def Framed-Tangle-Diagram-Equivalence-def
  using Framed-Tangle-Diagram-Equivalence-def
  Framed-Tangle-Equivalence.sym
by (metis Tangle-Equivalence.sym)

show transp Framed-Tangle-Diagram-Equivalence
  unfolding transp-def Framed-Tangle-Diagram-Equivalence-def
  using Framed-Tangle-Diagram-Equivalence-def Framed-Tangle-Equivalence.trans
by (metis Tangle-Equivalence.trans)

qed

quotient-type Framed-Link = Link-Diagram / Framed-Link-Diagram-Equivalence
morphisms Rep-Framed-Links Abs-Framed-Links
proof (rule equivpI)
show reflp Framed-Link-Diagram-Equivalence
  unfolding reflp-def Framed-Link-Diagram-Equivalence-def
  Framed-Tangle-Equivalence.refl
  by auto
show symp Framed-Link-Diagram-Equivalence
  unfolding Framed-Link-Diagram-Equivalence-def symp-def
  using Framed-Link-Diagram-Equivalence-def Framed-Tangle-Equivalence.sym
  by (metis Tangle-Equivalence.sym)
show transp Framed-Link-Diagram-Equivalence
  unfolding Framed-Link-Diagram-Equivalence-def transp-def
  using Framed-Link-Diagram-Equivalence-def Framed-Tangle-Equivalence.trans
  by (metis Tangle-Equivalence.trans)
qed

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
  using Tangle-Equivalence.trans assms (1) assms (2) by metis

lemma prelim-cup-compress:
  \((\text{basic (cup #[])}) \circ (\text{basic (vert # vert # [])})\) ∼
  \((\text{basic []}) \circ (\text{basic (cup #[])})\)

proof
  have domain-wall (\text{basic (cup # [])}) = 0
    by auto
  moreover have codomain-wall (\text{basic (cup # [])}) = 2
    by auto
  moreover
    have make-vert-block (nat (codomain-wall (\text{basic (cup # [])})))
      = (vert # vert # [])
      unfolding make-vert-block-def
    by auto
  moreover have is-tangle-diagram \(((\text{basic (cup#[]}))) \circ (\text{basic (vert # vert # []}))\)
    using is-tangle-diagram.simps by auto
  ultimately
  have compress-bottom
    \(((\text{basic (cup#[]}))) \circ (\text{basic (vert # vert # []})))
    (\text{basic []}) \circ (\text{basic (cup#[]})))
using compress-bottom-def by (metis is-tangle-diagram.simps(1))
then have compress 

(((basic (cup#[])) ∘ (basic (vert # vert # [])

(((basic []) ∘ (basic (cup#[]))))

using compress-def by auto
then have linkrel 

(((basic (cup#[])) ∘ (basic (vert # vert # []))

(((basic []) ∘ (basic (cup#[]))))

unfolding linkrel-def by auto
then show thesis

using Tangle-Equivalence.equality compress-bottom-def
Tangle-Moves.compress-bottom-def Tangle-Moves.compress-def
Tangle-Moves.linkrel-def

by auto
qed

lemma cup-compress:

(basic (cup#[])) ∘ (basic (vert # vert # [])) ∼ (basic (cup#[]))

proof −

have 

(((basic (cup#[])) ∘ (basic (vert # vert # []))

(((basic []) ∘ (basic (cup#[]))))

using prelim-cup-compress by auto
moreover have 

(((basic []) ∘ (basic (cup#[]))) ∼ (basic (cup#[]))

using domain-compose refl sym Tangle-Equivalence.domain-compose
Tangle-Equivalence.sym domain.simps(2) domain-block.simps
domain-wall.simps(1)
is-tangle-diagram.simps(1) monoid-add-class.add.right-neutral
by auto
ultimately show thesis using trans by (metis Example.transitive)
qed

abbreviation x::wall
where

x ≡ (basic [cup,cup]) ∘ (basic [vert,over,vert]) ∘ (basic [cap,cap])

abbreviation y::wall
where

y ≡ (basic [cup]) ∘ (basic [cap])

lemma uncross-straighten-left-over:left-over ∼ straight-line

proof −

have uncross right-over left-over

using uncross-positive-flip-def uncross-def by auto
then have linkrel right-over left-over

using linkrel-def by auto
then have right-over ∼ left-over

using Tangle-Equivalence.equality by auto
then have 1:left-over ∼ right-over

using Tangle-Equivalence.sym by auto
have uncross right-over straight-line

using uncross-positive-straighten-def uncross-def by auto
then have \( \text{linkrel right-over straight-line} \)
using \( \text{linkrel-def by auto} \)
then have \( 2:\text{right-over} \sim \text{straight-line} \)
using \( \text{Tangle-Equivalence.equality by auto} \)
have \( (\text{left-over} \sim \text{straight-line}) \land (\text{right-over} \sim \text{straight-line}) \)
\[\Longrightarrow \) thesis
using \( \text{transitive by auto} \)
then show \( ?\text{thesis using 1 2 transitive by blast} \)
qed

\text{example}
\begin{align*}
x \sim y
\end{align*}
\text{proof} –

have \( 1:\text{left-over} \sim \text{straight-line} \)
using \( \text{Tangle-Equivalence.equality uncross-straighten-left-over by auto} \)
moreover have \( 2:\text{straight-line} \sim \text{straight-line} \)
using \( \text{refl by auto} \)
have \( 3:(\text{left-over} \otimes \text{straight-line}) \sim (\text{straight-line} \otimes \text{straight-line}) \)
\text{proof} –

have \( \text{is-tangle-diagram (left-over)} \)
unfolding \( \text{is-tangle-diagram-def by auto} \)
moreover have \( \text{is-tangle-diagram (straight-line)} \)
unfolding \( \text{is-tangle-diagram-def by auto} \)
ultimately show \( ?\text{thesis using 1 2 by (metis Tangle-Equivalence.tensor-eq)} \)
qed

then have \( 4;\)
\[((\text{basic (cup#[])}) \circ (\text{left-over} \otimes \text{straight-line})) \sim ((\text{basic (cup#[])}) \circ (\text{straight-line} \otimes \text{straight-line}))\]
\text{proof} –

have \( \text{is-tangle-diagram (left-over} \otimes \text{straight-line)} \)
by \( \text{auto} \)
moreover have \( \text{is-tangle-diagram (straight-line} \otimes \text{straight-line)} \)
by \( \text{auto} \)
moreover have \( \text{is-tangle-diagram (basic (cup#[])}) \)
by \( \text{auto} \)
moreover have \( \text{domain-wall (left-over} \otimes \text{straight-line}) = (\text{codomain-wall (basic (cup#[])}) \)
unfolding \( \text{domain-wall-def by auto} \)
moreover have \( \text{domain-wall (straight-line} \otimes \text{straight-line}) = (\text{codomain-wall (basic (cup#[])}) \)
unfolding \( \text{domain-wall-def by auto} \)
moreover have \( (\text{basic (cup#[])}) \sim (\text{basic (cup#[])}) \)
using \( \text{refl by auto} \)
ultimately show \( ?\text{thesis using compose-eq 3 by (metis Tangle-Equivalence.compose-eq)} \)
qed

moreover have \( 5;\)
\[\text{(basic [cup])}\circ (\text{straight-line} \otimes \text{straight-line})\]
\[ \sim (\text{basic } [\text{cup}]) \]

**proof –**

**have 0:**

\[
\left( \text{basic } ([\text{cup}]) \right) \circ (\text{straight-line } \otimes \text{straight-line}) = \left( \text{basic } [\text{cup}] \right) \circ (\text{basic } [\text{vert}, \text{vert}])
\]

\[
\circ (\text{basic } [\text{vert}, \text{vert}]) \circ (\text{basic } [\text{vert}, \text{vert}])
\]

**by auto**

**let \( ?x = (\text{basic } (\text{cup}#[])) \)

\[
\circ (\text{basic } (\text{vert}#\text{vert}#[])) \circ (\text{basic } (\text{vert}#\text{vert}#[]))
\]

**let \( ?x1 = (\text{basic } (\text{vert}#\text{vert}#[])) \circ (\text{basic } (\text{vert}#\text{vert}#[])) \)

**have 1:** \( ?x \sim ((\text{basic } (\text{cup}#[])) \circ ?x1) \)

**proof –**

**have (\text{basic } (\text{cup}#[])) \circ (\text{basic } (\text{vert} \# \text{ vert} \# [])) \sim (\text{basic } (\text{cup}#[]))**

**using cup-compress by auto**

**moreover have is-tangle-diagram (\text{basic } (\text{cup}#[]))**

**using is-tangle-diagram-def by auto**

**moreover have is-tangle-diagram ((\text{basic } (\text{cup}#[])) \circ (\text{basic } (\text{vert} \# \text{ vert} \# [])))**

**using is-tangle-diagram-def by auto**

**moreover have is-tangle-diagram (?x1)**

**by auto**

**moreover have ?x1 \sim ?x1**

**using refl by auto**

**moreover have codomain-wall (\text{basic } (\text{cup}#[])) = domain-wall (\text{basic } (\text{vert}#\text{vert}#[]))**

**by auto**

**moreover have (\text{basic } (\text{cup}#[])) \sim (\text{basic } (\text{cup}#[]))**

**using refl by auto**

**ultimately show ?thesis**

**using compose-eq codomain-wall-compose compose-leftassociativity converse-composition-of-tangle-diagrams domain-wall-compose**

**by (metis Tangle-Equivalence.compose-eq is-tangle-diagram.simps(1))**

**qed**

**have 2:** \( ((\text{basic } (\text{cup}#[])) \circ ?x1) \sim (\text{basic } (\text{cup}#[])) \)

**proof –**

**have ((\text{basic } (\text{cup} \# [])) \circ (\text{basic } (\text{vert} \# \text{ vert} \# []))) \circ (\text{basic } (\text{vert} \# \text{ vert} \# [])) \sim ((\text{basic } (\text{cup}#[])) \circ (\text{basic } (\text{vert}#\text{vert}#[])))**

**proof –**

**have (\text{basic } (\text{cup}#[])) \circ (\text{basic } (\text{vert} \# \text{ vert} \# [])) \sim (\text{basic } (\text{cup}#[]))**

**using cup-compress by auto**

**moreover have (\text{basic } (\text{vert}#\text{vert}#[])) \sim (\text{basic } (\text{vert}#\text{vert}#[]))**

**using refl by auto**

**moreover have is-tangle-diagram (\text{basic } (\text{cup}#[]))**

**using is-tangle-diagram-def by auto**

**moreover have is-tangle-diagram ((\text{basic } (\text{cup}#[])) \circ (\text{basic } (\text{vert} \# \text{ vert} \# [])))**

**using is-tangle-diagram-def by auto**

**moreover have is-tangle-diagram ((\text{basic } (\text{vert}#\text{vert}#[])))**

**by auto**

38
moreover have
\[
\text{codomain-wall } (\text{basic } (\text{cup#[]})) \circ (\text{basic}(\text{vert#vert#[]})) = \text{domain-wall } (\text{basic}(\text{vert#vert#[]}))
\]
by auto
moreover
have \(\text{codomain-wall } (\text{basic } (\text{cup#[]})) = \text{domain-wall } (\text{basic}(\text{vert#vert#[]}))\)
by auto
ultimately show \(\text{thesis}\)
using compose-eq
by (metis \text{Tangle-Equivalence.compose-eq})
qed
then have \((\text{basic } (\text{cup#[]})) \circ ?x1 ~
\((\text{basic}(\text{cup#[]})) \circ (\text{basic}(\text{vert#vert#[]})))\)
by auto
then show \(\text{thesis}\) using cup-compress trans
by (metis (full-types) \text{Example.transitive})
qed
from 0 1 2 show \(\text{thesis}\) using trans transp-def trans compose-Nil
by (metis (opaque-lifting, no-types) \text{Example.transitive})
qed
let \(?y = (\text{basic } (\text{[]})) \circ (\text{basic } (\text{cup#[]})))\)
let \(?\text{temp} = (\text{basic}(\text{vert#over#vert#[]}))\circ (\text{basic}(\text{cap#vert#vert#[]})))\)
have 45: \((\text{left-over } \otimes \text{ straight-line}) =
\((\text{basic } (\text{cup#vert#vert#[]})) \circ ?\text{temp})\)
using tensor.simps by auto
then have \(55: (\text{basic } (\text{cup#[]})) \circ (\text{left-over } \otimes \text{ straight-line})
= (\text{basic } (\text{cup#[]})) \circ (\text{basic } (\text{cup#vert#vert#[]})) \circ ?\text{temp}\)
by auto
then have \(6:\)
\((\text{basic } (\text{cup#[]})) \circ (\text{basic } (\text{cup#vert#vert#[]}))
= (\text{basic } (\text{[]}) \circ (\text{cup#[]})) \circ (\text{basic } (\text{cap#vert#vert#[]})))\)
using concatenate.simps by auto
then have \(6:\)
\((\text{basic } (\text{cup#[]})) \circ (\text{basic } (\text{cup#vert#vert#[]}))
= (\text{basic } (\text{[]}) \circ (\text{cup#[]})) \circ (\text{basic } (\text{vert#vert#[]}))\)
using tensor.simps by auto
then have \(\text{prelim-cup-compress}\) by auto
moreover have \((\text{basic } (\text{[]})) \circ (\text{basic } (\text{cup#[]})))\)
~ \((\text{basic } (\text{[]}) \circ (\text{basic } (\text{cup#[]})))\)
using refl by auto
moreover have \(\text{is-tangle-diagram } ((\text{basic } (\text{cup#[]})) \circ (\text{basic } (\text{vert#vert#[]})))\)
by auto
moreover have \(\text{is-tangle-diagram } ((\text{basic } (\text{[]}))) \circ (\text{basic } (\text{cup#[]})))\)
by auto
ultimately have \(7: ?y \otimes ((\text{basic } (\text{cup#[]})) \circ (\text{basic } (\text{vert#vert#[]}))) ~ ((?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otimes ~ (?y) \otis \text{Example.}
\[ (\text{?y}) \]

\textbf{using} tensor-eq cup-compress Nil-right-tensor is-tangle-diagram.simps(1)

\textbf{refl}

\textbf{by (metis Tangle-Equivalence.tensor-eq)}

\textbf{then have} \((\text{?y}) \otimes (\text{?y}) = (\text{basic ([1]} \otimes ([1])))\)

\实业\((\text{basic (cup#[]} [1]) \otimes (\text{basic (cup#[]} [1])))\)

\textbf{using} tensor.simps(1) \textbf{by (metis compose-Nil)}

\textbf{then have} \((\text{?y}) \otimes ((\text{basic (cup#[])})) \circ ((\text{basic (vert#[]} [1]))))\)

\textbf{using} tensor.simps(1)_concatenate-def \textbf{by auto}

\textbf{then have} \((\text{?y}) \otimes ((\text{basic (cup#[])})) \circ ((\text{basic (vert#[]} [1]))))\)

\实业\((\text{basic ([1]})) \circ ((\text{basic (cup#[]} [1])))\)

\textbf{using} 7 \textbf{by auto}

\textbf{moreover have} \((\text{basic ([1]})) \circ ((\text{basic (cup#[]} [1]))) \circ ((\text{basic (cup#[]} [1])))\)

\textbf{proof—}

\textbf{have} domain-wall \((\text{basic (cup#[]} [1]))) = 0\)

\textbf{by auto}

\textbf{then show} \texttt{thesis using domain-compose sym}

\textbf{by (metis Tangle-Equivalence.domain-compose Tangle-Equivalence.sym is-tangle-diagram.simps(1))}

\textbf{qed}

\textbf{ultimately have} \((\text{?y}) \otimes ((\text{basic (cup#[])})) \circ ((\text{basic (vert#[]} [1]))))\)

\实业\((\text{basic (cup#[]} [1])))\)

\textbf{using trans by (metis (full-types) Example.transitive)}

\textbf{then have} \((\text{basic (cup#[])} \circ ((\text{basic (cup#[]} [1]))) \circ ((\text{basic (cup#[]} [1])))\)

\textbf{by auto}

\textbf{moreover have} \texttt{?temp} \sim \texttt{?temp}

\textbf{using} refl \textbf{by auto}

\textbf{moreover have} is-tangle-diagram \((\text{basic (cup#[])} \circ ((\text{basic (cup#[]} [1]))) \circ ((\text{basic (cup#[]} [1])))\)

\textbf{by auto}

\textbf{moreover have} is-tangle-diagram \((\text{basic (cup#[]} [1])))\)

\textbf{by auto}

\textbf{moreover have} is-tangle-diagram \((\texttt{?temp})\)

\textbf{by auto}

\textbf{moreover have} codomain-wall \((\text{basic (cup#[])} \circ ((\text{basic (cup#[]} [1]))) \circ ((\text{basic (cup#[]} [1])))\)

\textbf{by auto}

\textbf{moreover have} codomain-wall \((\text{basic (cup#[]} [1]))) = \text{domain-wall} \texttt{?temp}

\textbf{by auto}

\textbf{ultimately have} \texttt{8:} \((\text{basic (cup#[])} \circ ((\text{basic (cup#[]} [1]))) \circ ((\text{basic (cup#[]} [1])))\)

\实业\((\text{basic (cup#[]} [1]))) \circ ((\text{?temp})\)

\textbf{using} compose-eq \textbf{by (metis Tangle-Equivalence.compose-eq)}

\textbf{then have} \((\text{basic [cup,cup]} \circ ((\texttt{?temp}))\)

\实业\((\text{basic [cup]} \circ ((\text{left-over } \otimes \text{ straight-line}))\)

\textbf{using} 55 compose-leftassociativity sym wall.simps

\textbf{by (metis Tangle-Equivalence.sym compose-Nil)}

\textbf{moreover have} \((\text{basic [cup]} \circ ((\text{left-over } \otimes \text{ straight-line}))\)

\实业\((\text{basic [cup]} \circ ((\text{straight-line } \otimes \text{ straight-line}))\)

\textbf{using} 4 \textbf{by auto}

\textbf{ultimately have} \((\text{basic [cup,cup]} \circ ((\texttt{?temp}))\)

\textbf{by auto}
\[ \sim (\text{basic } \text{cup}) \circ (\text{straight-line} \otimes \text{straight-line}) \]

**proof**

have \((\text{basic } \text{cup,cup}) \circ (?\text{temp}))\]

\[ \sim (\text{basic } \text{cup}) \circ (\text{left-over} \otimes \text{straight-line}) \]

**using** 8 55 compose-less-associativity sym wall.simps Tangle-Equivalence.sym compose-nil

by (metis)

moreover have \((\text{basic } \text{cup}) \circ (\text{left-over} \otimes \text{straight-line})\]

\[ \sim (\text{basic } \text{cup}) \circ (\text{straight-line} \otimes \text{straight-line}) \]

**using** 4 by auto

moreover have \(((\text{basic } \text{cup,cup}) \circ (?\text{temp}))\]

\[ \sim (\text{basic } \text{cup}) \circ (\text{left-over} \otimes \text{straight-line}) \]

\(\land ((\text{basic } \text{cup}) \circ (\text{left-over} \otimes \text{straight-line})\]

\[ \sim (\text{basic } \text{cup}) \circ (\text{straight-line} \otimes \text{straight-line}) \]

\[ \Rightarrow ?\text{thesis} \]

**using** Example.transitive by auto

ultimately show ?\text{thesis} by auto

qed

then have \((\text{basic } \text{cup,cup}) \circ (?\text{temp}) \sim (\text{basic } \text{cup} \circ \text{cap})\]

**using** trans transp-def 5 by (metis Example.transitive)

moreover have \((\text{basic } \text{cap} \circ \text{cap}) \sim (\text{basic } \text{cap} \circ \text{cap})\]

**using** refl by auto

moreover have is-tangle-diagram \((\text{basic } \text{cup} \circ \text{cup} \circ \text{cap}) \circ (?\text{temp})\]

by auto

moreover have is-tangle-diagram \((\text{basic } \text{cap} \circ \text{cap})\]

by auto

moreover have is-tangle-diagram \((\text{basic } \text{cap} \circ \text{cap})\]

by auto

moreover have codomain-wall \((\text{basic } \text{cap} \circ \text{cup} \circ \text{cap}) \circ (?\text{temp})\]

= domain-wall \((\text{basic } \text{cap} \circ \text{cap})\]

by auto

moreover have codomain-wall \((\text{basic } \text{cap} \circ \text{cap})\) = domain-wall \((\text{basic } \text{cap} \circ \text{cap})\]

by auto

ultimately have 9: \(((\text{basic } \text{cup} \circ \text{cup} \circ \text{cap}) \circ (?\text{temp}) \circ (\text{basic } \text{cap} \circ \text{cap})\]

\[ \sim (\text{basic } \text{cap} \circ \text{cap}) \circ (\text{basic } \text{cap} \circ \text{cap}) \]

**using** Tangle-Equivalence.compose-eq by metis

let \(?z = (\text{basic } \text{cup} \circ \text{cup} \circ \text{cap} \circ (\text{basic } \text{vert} \circ \text{vert} \circ \text{vert}))\]

have 10: \(((\text{basic } \text{cap} \circ \text{cup} \circ \text{cap}) \circ (?\text{temp}) \circ (\text{basic } \text{cap} \circ \text{cap})\]

= \(?z \circ (\text{basic } \text{cap} \circ \text{vert} \circ \text{vert} \circ \text{vert})) \circ (\text{basic } \text{cap} \circ \text{cap})\]

by auto

then have 11: \(((\text{basic } \text{cap} \circ \text{vert} \circ \text{vert} \circ \text{vert}) \circ (\text{basic } \text{cap} \circ \text{cap}) \circ (\text{basic } \text{cap} \circ \text{cap})\]

= \(((\text{basic } \text{cap} \circ \text{cap}) \circ (\text{basic } \text{vert} \circ \text{vert} \circ \text{vert})) \circ (\text{basic } \text{cap} \circ \text{cap})\]

by auto

unfolding concatenate-def by auto

then have 12: \(((\text{basic } \text{cap} \circ \text{vert} \circ \text{vert} \circ \text{vert}) \circ (\text{basic } \text{cap} \circ \text{cap}) \circ (\text{basic } \text{cap} \circ \text{cap})\]

= \(((\text{basic } \text{cap} \circ \text{cap}) \circ (\text{basic } \text{vert} \circ \text{vert} \circ \text{vert})) \circ (\text{basic } \text{cap} \circ \text{cap})\]

by auto

using tensor.simps by auto

let \(?w = (\text{basic } \text{cap} \circ \text{cap}) \circ (\text{basic } \text{cap} \circ \text{cap}))\]

have 13: \(((\text{basic } \text{vert} \circ \text{vert} \circ \text{vert}) \circ (\text{basic } \text{cap} \circ \text{cap})) \sim ?w\]
proof
  have codomain-wall (basic (cap#[])) = 0
    by auto
  then have domain-wall (basic (cap#[])) = 2 by auto
  then have (vert#vert#[])
    = make-vert-block (nat (domain-wall (basic (cap#[]))))
    by (simp add: make-vert-block-def)
  then have compress-top (((basic (vert#vert#[])) o (basic (cap#[]))) ?w
    using compress-top-def by auto
  then have compress (((basic (vert#vert#[])) o (basic (cap#[]))) ?w
    using compress-def by auto
  then have linkrel (((basic (vert#vert#[])) o (basic (cap#[]))) ?w
    using linkrel-def by auto
  then have (((basic (vert#vert#[])) o (basic (cap#[]))) ?w
    using Tangle-Equivalence.equality by auto
  then show thesis by simp
qed
moreover have is-tangle-diagram (((basic (vert#vert#[])) o (basic (cap#[]))))
  by auto
moreover have is-tangle-diagram ?w
  by auto
moreover have ?w ~ ?w
  using refl by auto
ultimately have 14: (?w) ⊗ (((basic (vert#vert#[])) o (basic (cap#[])))) ~ (((?w)⊗ (?w))
  using Tangle-Equivalence.tensor-eq by metis
  then have ((basic(cap#vert#vert#[])) o (basic (cap#[]))) ~ ((?w)⊗ (?w))
    using 13 by auto
moreover have ((?w)⊗ (?w)) = (basic (cap#cap#[])) o (basic ([])
  using tensor.simps by auto
ultimately have (((basic(cap#vert#vert#[])) o (basic (cap#[])))) ~ (basic (cap#cap#[])) o (basic ([]))
  by auto
moreover have ?z ~ ?z
  using refl by auto
moreover have domain-wall (((basic(cap#vert#vert#[])) o (basic ([])))
    = codomain-wall (?z)
  by auto
moreover have domain-wall (((basic(cap#vert#vert#[])) o (basic (cap#[]))))
    = codomain-wall (?z)
  by auto
moreover have is-tangle-diagram (((basic(cap#vert#vert#[])) o (basic (cap#[]))))
  by auto
moreover have is-tangle-diagram (?z)
  by auto
moreover have is-tangle-diagram ((basic(cap#cap#[])) o (basic ([]))
  by auto
ultimately have 14: (?z) o (((basic(cap#vert#vert#[])) o (basic (cap#[]))))
  ~ (?z) o (((basic(cap#cap#[])) o (basic ([]))) (is ?aa ~ ?bb)

42
using \texttt{Tangle-Equivalence.compose-eq} by \texttt{metis}
m{}

moreover have 15: \((?z \circ ((\texttt{basic}(\texttt{cap}\#\texttt{cap}\#[])))) \circ (\texttt{basic}([]))) 
\sim ((?z \circ (\texttt{basic}(\texttt{cap}\#\texttt{cap}\#[])))) (\texttt{is} \ ?bb \sim \ ?cc)

using \texttt{Tangle-Equivalence.codomain-compose} \texttt{Tangle-Equivalence.sym}
\texttt{is-tangle-diagram} (\texttt{basic} [\texttt{cap}, \texttt{cap} \circ \texttt{basic} []]) \texttt{codomain-wall-compose}
\texttt{compose-left-associativity} \texttt{converse-composition-of-tangle-diagrams}
\texttt{domain-block} \texttt{simps(1)} \texttt{domain-wall} \texttt{simps(1)}

by (\texttt{metis} \texttt{(opaque-lifting, mono-tags)} \texttt{Tangle-Equivalence.compose-eq}
\texttt{Tangle-Equivalence.refl}
\texttt{codomain-wall} \texttt{(basic} [\texttt{cup}, \texttt{cup}])
\texttt{domain-wall} \texttt{(basic} [\texttt{vert}, \texttt{over}, \texttt{vert}] \circ \texttt{basic} [\texttt{cap}, \texttt{vert}, \texttt{vert}])

\texttt{domain-wall} (\texttt{basic} [\texttt{cap}, \texttt{cap}] \circ \texttt{basic} [])
\texttt{codomain-wall} (\texttt{basic} [\texttt{cup}, \texttt{cup}] \circ \texttt{basic} [\texttt{vert}, \texttt{over}, \texttt{vert}])
\texttt{comp-of-tangle-dgms} \texttt{domain-wall-compose} \texttt{is-tangle-diagram} \texttt{simps(1)}

ultimately have \((\texttt{?aa} \sim \texttt{?bb}) \land (\texttt{?bb} \sim \texttt{?cc}) \implies \texttt{?aa} \sim \texttt{?cc})

using \texttt{transitive} by \texttt{auto}
then have 16: \(?aa \sim \?cc)

using 14 15 by \texttt{auto}
then have 17: \(((\texttt{basic} (\texttt{cup}\#[])) \circ (\texttt{basic} (\texttt{cap}\#[]))) \sim \texttt{?aa})

using 9 10 \texttt{Tangle-Equivalence.trans} \texttt{Tangle-Equivalence.sym}
by (\texttt{metis} \texttt{(opaque-lifting, no-types)})

have \(((\texttt{basic} (\texttt{cup}\#[])) \circ (\texttt{basic} (\texttt{cap}\#[]))) \sim \texttt{?aa}) \land (\texttt{?aa} \sim \texttt{?cc})
\implies ((\texttt{basic} (\texttt{cup}\#[])) \circ (\texttt{basic} (\texttt{cap}\#[]))) \sim \texttt{?cc}

using \texttt{transitive} by \texttt{auto}
then have \(((\texttt{basic} (\texttt{cap}\#[])) \circ (\texttt{basic} (\texttt{cap}\#[]))) \sim \texttt{?cc})

using 17 16 by \texttt{auto}
then show \texttt{?thesis} using \texttt{Tangle-Equivalence.sym} by \texttt{auto}
qed

end

9 Kauffman Matrix and Kauffman Bracket- Definitions and Properties

theory \texttt{Kauffman-Matrix}

imports \texttt{Matrix-Tensor.Matrix-Tensor}
\texttt{Link-Algebra}
\texttt{HOL-Computational-Algebra.Polynomial}
\texttt{HOL-Computational-Algebra.Fraction-Field}

begin

10 Rational Functions

intpoly is the type of integer polynomials

type-synonym \texttt{intpoly} = \texttt{int poly}
lemma eval-pCons: poly (pCons 0 1) x = x
  using poly-1 poly-pCons by auto

lemma pCons2: (pCons 0 1) ≠ (1::int poly)
  using eval-pCons poly-1 zero-neq-one by metis

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

lemma non-zero: x ≠ 0
  using var-def pCons-eq-0-iff zero-neq-one by (metis)

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 ≠ 0 and d ≠ 0
  shows Fract a b = Fract c d ←→ a * d = c * b
  using eq-fract assms by auto

lemma A-non-zero: A ≠ (0::rat-poly)
  unfolding var-def1
  proof (rule ccontr)
    assume 0: ¬ (Fract x 1 ≠ (0::rat-poly))
    then have Fract x 1 = (0::rat-poly)
      by auto
    moreover have (0::rat-poly) = Fract (0::intpoly) (1::intpoly)
      by (metis Zero-fract-def)
    ultimately have Fract x (1::intpoly) = Fract (0::intpoly) (1::intpoly)
      by auto
    moreover have (1::intpoly) ≠ 0
      by auto
    ultimately have x*(1::intpoly) = (0::intpoly)*(1::intpoly)
      using eq-fract by metis
    then have x = (0::intpoly)
      by auto
    then show False using non-zero by auto
  qed

lemma mult-inv-non-zero:
  assumes (p::rat-poly) ≠ 0
    and p*q = (1::rat-poly)
shows \( q \neq 0 \)
using assms by auto

abbreviation rat-poly-times::rat-poly \Rightarrow rat-poly \Rightarrow rat-poly
where
rat-poly-times \( p \) \( q \) \( \equiv \) \( p \times q \)

abbreviation rat-poly-plus::rat-poly \Rightarrow rat-poly \Rightarrow rat-poly
where
rat-poly-plus \( p \) \( q \) \( \equiv \) \( p + q \)

abbreviation rat-poly-inv::rat-poly \Rightarrow rat-poly
where
rat-poly-inv \( p \) \( \equiv \) \( - p \)

interpretation rat-poly:semiring-0 rat-poly-plus 0 rat-poly-times
by (unfold-locales)

interpretation rat-poly:semiring-1 1 rat-poly-times rat-poly-plus 0
by (unfold-locales)

lemma mat1-equiv:mat1 \( (1::nat) = [[(1::rat-poly)]] \)
by (simp add: mat1I-def vec1I-def)

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
apply(unfold-locales)
apply(auto)
proof –
fix \( p \) \( q \) \( r \)
show rat-poly-times \( p \) (rat-poly-plus \( q \) \( r \))
= rat-poly-plus (rat-poly-times \( p \) \( q \)) (rat-poly-times \( p \) \( r \))
by (simp add: distrib-left)
show rat-poly-times (rat-poly-plus \( p \) \( q \)) \( r \)
= rat-poly-plus (rat-poly-times \( p \) \( r \)) (rat-poly-times \( q \) \( r \))
by (metis comm-semiring-class.distrib)
qed

lemma rat-poly.matrix-mult \( [[A,1],[\theta,A]] \) \( [[A,\theta],[\theta,A]] \) \( = \) \( [[A\times A,A],[\theta,A\times A]] \)
apply(simp add: mat-multI-def)
apply(simp add: matT-vec-multI-def)
apply(auto simp add: replicate-def rat-poly.row-length-def)
apply(auto simp add: scalar-prod)
done

abbreviation
  rat-polymat-tensor::rat-poly mat ⇒ rat-poly mat ⇒ rat-poly mat
  (infixl ⊗ 65)
where
  rat-polymat-tensor p q ≡ rat-poly.Tensor p q

lemma assumes (j::nat) div a = i div a
  and j mod a = i mod a
  shows j = i
proof -
  have a*(j div a) + (j mod a) = j
    using mult-div-mod-eq by simp
  moreover have a*(i div a) + (i mod a) = i
    using mult-div-mod-eq by auto
  ultimately show thesis using assms by metis
qed

lemma [[1]] ⊗ M = M
  by (metis rat-poly.Tensor-left-id)

lemma M ⊗ [[1]] = M
  by (metis rat-poly.Tensor-right-id)

11 Kauffman matrices

We assign every brick to a matrix of rational polynomials

primrec brickmat::brick ⇒ rat-poly mat
where
  brickmat vert = [[1,0],[0,1]]
  | brickmat cap = [[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
  using non-zero One-fract-def monoid-mult-class.mult.right-neutral

46
lemma inverse2: rat-poly-times B A = 1
  using One-fract-def monoid-mult-class mult-right-neutral mult-fract
  mult-fract-cancel non-zero var-def1 var-def2
  by (metis (opaque-lifting, no-types))

lemma B-non-zero: B ≠ 0
  using A-non-zero mult-inv-non-zero inverse1
  divide-fract div-0 fract-collapse(2)
  monoid-mult-class mult-left-neutral
  mult-fract-cancel non-zero var-def2 zero-neq-one
  by (metis (opaque-lifting, mono-tags))

lemma rat-poly-times p (q + r)
  = (rat-poly-times p q) + (rat-poly-times p r)
  by (metis rat-poly.plus-left-distributivity)

lemma minus-left-distributivity:
  rat-poly-times p (q - r)
  = (rat-poly-times p q) - (rat-poly-times p r)
  using minus-mult-right right-diff-distrib by blast

lemma minus-right-distributivity:
  rat-poly-times (p - q) r = (rat-poly-times p r) - (rat-poly-times q r)
  using minus-left-distributivity rat-poly.comm by metis

lemma equation:
  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
  have rat-poly-times (rat-poly-times A A) A
    = ((A*A)*A)
    by auto
  then have rat-poly-times B (B - rat-poly-times (rat-poly-times A A) A)
    = B*B - B*((A*A)*A)
    using minus-left-distributivity by auto
  moreover have ... = B*B - (B*(A*(A*A)))
    by auto
  moreover have ... = B*B - ((B*A)*(A*A))
    by auto
  moreover have ... = B*B - A*A
    using inverse2 by auto
  ultimately have 1:
    rat-poly-times B (B - rat-poly-times (rat-poly-times A A) A)
    = B*B - A*A
by auto
have \( \text{rat-poly-times} \ (\text{rat-poly-times} \ B \ B) \ B = (B * B) * B \)
by auto
then have
\( (\text{rat-poly-times} \ (A - \text{rat-poly-times} \ (\text{rat-poly-times} \ B \ B) \ B) \ A) \)
\( = (A * A) - ((B * B) * B) * A \)
using \( \text{minus-right-distributivity} \) by auto
moreover have \( ... = (A * A) - ((B * B) * (B * A)) \)
by auto
moreover have \( ... = (A * A) - (B * B) \)
using \( \text{inverse2} \) by auto
ultimately have \( 2: \)
\( (\text{rat-poly-times} \ (A - \text{rat-poly-times} \ (\text{rat-poly-times} \ B \ B) \ B) \ A) \)
\( = (A * A) - (B * B) \)
by auto
have \( B * B - A * A + (A * A) - (B * B) = 0 \)
by auto
with \( 1 \ 2 \) show \( \text{thesis} \) by auto
qed

lemma \( \text{rat-poly-inv} \ A = -A \)
by auto

lemma \( \text{vert-dim:rat-poly.row-length} \ (\text{brickmat vert}) = 2 \) \text{ and } \( \text{length} \ (\text{brickmat vert}) = 2 \)
using \( \text{rat-poly.row-length-def} \) by auto

lemma \( \text{cup-dim:rat-poly.row-length} \ (\text{brickmat cup}) = 1 \) \text{ and } \( \text{length} \ (\text{brickmat cup}) = 4 \)
using \( \text{rat-poly.row-length-def} \) by auto

lemma \( \text{cap-dim:rat-poly.row-length} \ (\text{brickmat cap}) = 4 \) \text{ and } \( \text{length} \ (\text{brickmat cap}) = 1 \)
using \( \text{rat-poly.row-length-def} \) by auto

lemma \( \text{over-dim:rat-poly.row-length} \ (\text{brickmat over}) = 4 \) \text{ and } \( \text{length} \ (\text{brickmat over}) = 4 \)
using \( \text{rat-poly.row-length-def} \) by auto

lemma \( \text{under-dim:rat-poly.row-length} \ (\text{brickmat under}) = 4 \) \text{ and } \( \text{length} \ (\text{brickmat under}) = 4 \)
using \( \text{rat-poly.row-length-def} \) by auto
lemma mat-vert: mat 2 2 (brickmat vert)
unfolding mat-def Ball-def vec-def by auto

lemma mat-cup: mat 1 4 (brickmat cup)
unfolding mat-def Ball-def vec-def by auto

lemma mat-cap: mat 4 1 (brickmat cap)
unfolding mat-def Ball-def vec-def by auto

lemma mat-over: mat 4 4 (brickmat over)
unfolding mat-def Ball-def vec-def by auto

lemma mat-under: mat 4 4 (brickmat under)
unfolding mat-def Ball-def vec-def by auto

primrec rowlength: :: nat ⇒ nat
where
rowlength 0 = 1
|rowlength (Suc k) = 2*(Suc k)

lemma (rat-poly.row-length (brickmat d)) = (2^nat (domain d))
using vert-dim cup-dim cap-dim over-dim under-dim domain.simps
by (cases d) (auto)

lemma rat-poly.row-length (brickmat cup) = 1
unfolding rat-poly.row-length-def by auto

lemma two: (Suc (Suc 0)) = 2
by eval

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

primrec blockmat:: block ⇒ rat-poly mat
where
blockmat [] = [[1]]
|blockmat (l#ls) = (brickmat l) ⊗ (blockmat ls)

lemma blockmat [a] = brickmat a
unfolding blockmat.simps rat-poly.Tensor-right-id by auto

lemma nat-sum:
assumes a ≥ 0 and b ≥ 0
shows nat (a+b) = (nat a) + (nat b)
using assms by auto

lemma rat-poly.row-length (brickmat ls) = (2^nat ((domain-block ls))))
proof(induct ls)
case Nil
show ?case unfolding blockmat.simps(1) rat-poly.row-length-def by auto
next
case (Cons l ls)
show ?case
proof(cases l)
case vert
  have rat-poly.row-length (blockmat ls) = 2 ^ nat (domain-block ls)
    using Cons by auto
then have rat-poly.row-length (blockmat (l#ls))
  = (rat-poly.row-length (brickmat l))
    *(rat-poly.row-length (blockmat ls))
    using blockmat.simps rat-poly.row-length-mat by auto
moreover have ... = 2*(2 ^ nat (domain-block ls))
    using rat-poly.row-length-def Cons vert by auto
moreover have ... = 2*(1 + nat (domain-block ls))
    using domain-block.simps by auto
moreover have ... = 2*(nat (domain l) + nat (domain-block ls))
    using domain.simps vert by auto
moreover have ... = 2*(nat (domain l + domain-block ls))
    using Suc-eq-plus1-left Suc-nat-eq-nat-zadd1
calculation(4) domain.simps(1) domain-block-non-negative
    vert
    by (metis)
major  have ... = 2*(nat (domain-block (l#ls)))
    using domain-block.simps vert by auto
ultimately show ?thesis by metis
next
case over
  have rat-poly.row-length (blockmat ls) = 2 ^ nat (domain-block ls)
    using Cons by auto
then have rat-poly.row-length (blockmat (l#ls))
  = (rat-poly.row-length (brickmat l))
    *(rat-poly.row-length (blockmat ls))
    using blockmat.simps rat-poly.row-length-mat by auto
also have ... = 4*(2 ^ nat (domain-block ls))
    using rat-poly.row-length-def Cons over by auto
also have ... = 2*(2 + nat (domain-block ls))
    using domain-block.simps by auto
also have ... = 2*(nat (domain l) + nat (domain-block ls))
    using domain.simps over by auto
also have ... = 2*(nat (domain l + domain-block ls))
    by (simp add: nat-add-distrib domain-block-nonnegative over)
also have ... = 2*(nat (domain-block (l#ls)))
    by simp
finally show ?thesis .
next
case under
  have rat-poly.row-length (blockmat ls) = 2 ^ nat (domain-block ls)
    using Cons by auto
then have rat-poly.row-length (blockmat (l#ls))
  = (rat-poly.row-length (brickmat l))
    *(rat-poly.row-length (blockmat ls))
    using blockmat.simps rat-poly.row-length-mat by auto
also have ... = 4*(2 ^ nat (domain-block ls))

using \texttt{rat-poly.row-length-def} \texttt{Cons} under \texttt{by auto}
also have \ldots = 2^n(2 + \texttt{nat}(\texttt{domain-block} \texttt{ls}))
using \texttt{domain-block.simps by auto}
also have \ldots = 2^n(\texttt{nat}(\texttt{domain} \texttt{l}) + \texttt{nat}(\texttt{domain-block} \texttt{ls}))
using \texttt{domain.simps under by auto}
also have \ldots = 2^n(\texttt{nat}(\texttt{domain} \texttt{l} + \texttt{domain-block} \texttt{ls}))
by (\texttt{simp add: nat-add-distrib domain-block-nonnegative under})
also have \ldots = 2^n(\texttt{nat}(\texttt{domain-block} \texttt{l}\#\texttt{ls}))
using \texttt{domain-block.simps by auto}
finally show \texttt{?thesis} .
next
case \texttt{cup}
have \texttt{rat-poly.row-length (blockmat} \texttt{ls}) = 2^n \texttt{(domain-block} \texttt{ls})
using \texttt{Cons by auto}
them have \texttt{rat-poly.row-length (blockmat} \texttt{l}\#\texttt{ls})
= (\texttt{rat-poly.row-length (brickmat} \texttt{l})
\times(\texttt{rat-poly.row-length (blockmat} \texttt{ls}))
using \texttt{blockmat.simps rat-poly.row-length-def by auto}
moreover have \ldots = 2^n(\texttt{nat}(\texttt{domain-block} \texttt{ls}))
using \texttt{domain-block.simps by auto}
moreover have \ldots = 2^n(\texttt{nat}(\texttt{domain} \texttt{l} + \texttt{domain-block} \texttt{ls}))
using \texttt{domain.simps cup by auto}
moreover have \ldots = 2^n(\texttt{nat}(\texttt{domain-block} \texttt{l}\#\texttt{ls}))
using \texttt{domain-block.simps\ by auto}
moreover have \ldots = 2^n(\texttt{nat}(\texttt{domain-block} \texttt{l} + \texttt{domain-block} \texttt{ls}))
using \texttt{domain.simps\ cup by auto}
multiplicity have \ldots = 2^n(\texttt{nat}(\texttt{domain-block} \texttt{l}\#\texttt{ls}))
using \texttt{domain-block.simps\ by auto}
ultimately show \texttt{?thesis by metis}
next
case \texttt{cap}
have \texttt{rat-poly.row-length (blockmat} \texttt{ls}) = 2^n \texttt{(domain-block} \texttt{ls})
using \texttt{Cons by auto}
them have \texttt{rat-poly.row-length (blockmat} \texttt{l}\#\texttt{ls})
= (\texttt{rat-poly.row-length (brickmat} \texttt{l})
\times(\texttt{rat-poly.row-length (blockmat} \texttt{ls}))
using \texttt{blockmat.simps rat-poly.row-length-def by auto}
moreover have \ldots = 2^n(\texttt{nat}(\texttt{domain-block} \texttt{ls}))
using \texttt{domain-block.simps cup by auto}
moreover have \ldots = 2^n(\texttt{nat}(\texttt{domain} \texttt{l} + \texttt{domain-block} \texttt{ls}))
using \texttt{domain.simps cup by auto}
moreover have \ldots = 2^n(\texttt{nat}(\texttt{domain-block} \texttt{l}\#\texttt{ls}))
using \texttt{domain-block.simps\ by auto}
multiplicity have \ldots = 2^n(\texttt{nat}(\texttt{domain-block} \texttt{l}\#\texttt{ls}))
using \texttt{domain-block.simps\ by auto}
by (\texttt{simp add: cap domain-block-nonnegative nat-add-distrib})
multiplicity have \ldots = 2^n(\texttt{nat}(\texttt{domain-block} \texttt{l}\#\texttt{ls}))
using \texttt{domain-block.simps\ by auto}

51
ultimately show \( \text{thesis} \) by metis
qed
qed

**lemma** row-length-domain-block:
\[
\text{rat-poly}.\text{row-length} (\text{blockmat} \ \text{ls}) = (2^\text{nat} ((\text{domain-block} \ \text{ls})))
\]

**proof** (induct \text{ls})
case Nil
show \( ?\text{case} \) unfolding blockmat.simps(1) rat-poly.row-length-def by auto
next
case (Cons \text{l} \text{ls})
show \( ?\text{case} \)
proof (cases \text{l})
case vert
have \( \text{rat-poly}.\text{row-length} (\text{blockmat} \ \text{ls}) = 2^\text{nat} (\text{domain-block} \ \text{ls}) \)
  using Cons by auto
then have \( \text{rat-poly}.\text{row-length} (\text{blockmat} (\mathbf{l} \# \text{ls})) \)
  = (\text{rat-poly}.\text{row-length} (\text{brickmat} \ \text{l}))
  * (\text{rat-poly}.\text{row-length} (\text{blockmat} \ \text{ls}))
  using blockmat.simps \ rat-poly.row-length-mat by auto
moreover have \( \ldots \) = \( 2 \times (2^\text{nat} (\text{domain-block} \ \text{ls})) \)
  using rat-poly.row-length-def Cons vert by auto
moreover have \( \ldots \) = \( 2^{(1 + \text{nat} (\text{domain-block} \ \text{ls}))) \)
  using domain-block.simps by auto
moreover have \( \ldots \) = \( 2^{\text{nat} (\text{domain} \ \text{l} + \text{domain-block} \ \text{ls})} \)
  using domain.simps vert by auto
moreover have \( \ldots \) = \( 2^{\text{nat} (\text{domain} \ \text{l} + \text{domain-block} \ \text{ls})} \)
  using Suc-eq-plus1-left Suc-nat-eq-nat-zadd1 calculation(4) domain.simps(1)
  domain-block-non-negative vert
  by metis
moreover have \( \ldots \) = \( 2^{\text{nat} (\text{domain-block} (\mathbf{l} \# \text{ls}))} \)
  using domain-block.simps by auto
ultimately show \( \text{thesis} \) by metis
next
case over
have \( \text{rat-poly}.\text{row-length} (\text{blockmat} \ \text{ls}) = 2^\text{nat} (\text{domain-block} \ \text{ls}) \)
  using Cons by auto
then have \( \text{rat-poly}.\text{row-length} (\text{blockmat} (\mathbf{l} \# \text{ls})) \)
  = (\text{rat-poly}.\text{row-length} (\text{brickmat} \ \text{l}))
  * (\text{rat-poly}.\text{row-length} (\text{blockmat} \ \text{ls}))
  using blockmat.simps \ rat-poly.row-length-mat by auto
moreover have \( \ldots \) = \( 4 \times (2^\text{nat} (\text{domain-block} \ \text{ls})) \)
  using rat-poly.row-length-def Cons over by auto
moreover have \( \ldots \) = \( 2^{(2 + \text{nat} (\text{domain-block} \ \text{ls}))} \)
  using domain-block.simps by auto
moreover have \( \ldots \) = \( 2^{(\text{domain} \ \text{l} + \text{domain-block} \ \text{ls})} \)
  using domain.simps over by auto

52
moreover have \ldots = 2^{\text{nat} (\text{domain \(l + domain-block \(ls)\)) by (simp add: over domain-block-nonnegative nat-add-distrib)}}

moreover have \ldots = 2^{\text{nat} (domain-block (l\#ls)) using domain-block.simps by auto}

ultimately show \(\text{thesis by metis}\)

next

case \text{under}

have \text{rat-poly.row-length (blockmat \(ls) = 2 \sim \text{nat (domain-block \(ls)\)} using \text{Cons by auto}}

then have \text{rat-poly.row-length (blockmat (l\#ls))}
     \quad = (\text{rat-poly.row-length (brickmat \(l))}
     \quad \quad * (\text{rat-poly.row-length (blockmat \(ls)\})
     \quad \quad \quad using blockmat.simps \text{ rat-poly.row-length-mat by auto}

moreover have \ldots = 4*(2 \sim \text{nat (domain-block \(ls)\)} using rat-poly.row-length-def \text{Cons under by auto}}

moreover have \ldots = 2^{2 + \text{nat (domain-block \(ls)\)} using domain-block.simps by auto

moreover have \ldots = 2^{\text{nat (domain \(l) + nat (domain-block \(ls)\)} using domain.simps under by auto

moreover have \ldots = 2^{\text{nat (domain \(l + domain-block \(ls)\)) by (simp add: under domain-block-nonnegative nat-add-distrib)}}

moreover have \ldots = 2^{\text{nat (domain-block (l\#ls))) using domain-block.simps by auto}

ultimately show \(\text{thesis by metis}\)

next

case \text{cup}

have \text{rat-poly.row-length (blockmat \(ls) = 2 \sim \text{nat (domain-block \(ls)\)} using \text{Cons by auto}}

then have \text{rat-poly.row-length (blockmat (l\#ls))}
     \quad = (\text{rat-poly.row-length (brickmat \(l))}
     \quad \quad * (\text{rat-poly.row-length (blockmat \(ls)\})
     \quad \quad \quad using blockmat.simps \text{ rat-poly.row-length-mat by auto}

moreover have \ldots = 1*(2 \sim \text{nat (domain-block \(ls)\)} using rat-poly.row-length-def \text{Cons cup by auto}}

moreover have \ldots = 2^{\text{nat (domain \(l) + nat (domain-block \(ls)\)} using domain-block.simps cup by auto

moreover have \ldots = 2^{\text{nat (domain \(l + domain-block \(ls)\)} using nat-sum cup domain.simps(2)
     \quad nat-0 plus-int-code(2) plus-nat.add-0 \text{ by (metis)}}

moreover have \ldots = 2^{\text{nat (domain-block (l\#ls))) using domain-block.simps by auto}

ultimately show \(\text{thesis by metis}\)

next

case \text{cap}

have \text{rat-poly.row-length (blockmat \(ls) = 2 \sim \text{nat (domain-block \(ls)\)} using \text{Cons by auto}}

then have \text{rat-poly.row-length (blockmat (l\#ls))}
\[
\begin{align*}
&= (\text{rat-poly}\cdot\text{row-length } (\text{brickmat l})) \\
&\quad \times (\text{rat-poly}\cdot\text{row-length } (\text{blockmat ls})) \\
&\quad \text{using blockmat.simps rat-poly.row-length-mat by auto}
\end{align*}
\]
moreover have \( \ldots = 4 \times (2 \sim \text{nat } (\text{domain-block ls})) \)
\begin{align*}
&\quad \text{using rat-poly.row-length-def Cons cap by auto} \\
&\quad \text{moreover have } \ldots = 2^2(2 + \text{nat } (\text{domain-block ls})) \\
&\quad \text{using domain-block.simps by auto} \\
&\quad \text{moreover have } \ldots = 2^2(\text{nat } (\text{domain l}) + \text{nat } (\text{domain-block ls})) \\
&\quad \text{using domain.simps by auto} \\
&\quad \text{moreover have } \ldots = 2^2(\text{nat } (\text{domain-block } (\text{l#ls}))) \\
&\quad \text{using domain-block.simps by auto} \\
&\quad \text{ultimately show } \text{thesis by metis}
\end{align*}
\begin{align*}
&\text{qed} \\
&\text{qed}
\end{align*}

\begin{align*}
\text{lemma } \text{length-codomain-block:length } (\text{blockmat ls}) \\
&= (2^2(\text{nat } ((\text{codomain-block ls})))) \\
\text{proof (induct ls)} \\
\text{case Nil} \\
&\text{show } \text{case unfolding blockmat.simps(1) rat-poly.row-length-def by auto} \\
\text{next} \\
\text{case } (\text{Cons l ls}) \\
&\text{show } \text{case} \\
&\text{proof (cases l)} \\
\text{case vert} \\
&\text{have } \text{length } (\text{blockmat ls}) = 2^2(\text{nat } (\text{codomain-block ls})) \\
&\quad \text{using Cons by auto} \\
&\text{then have } \text{length } (\text{blockmat } (\text{l#ls})) \\
&\quad = (\text{length } (\text{brickmat l})){\times}\text{length } (\text{blockmat ls}) \\
&\quad \text{using blockmat.simps rat-poly.length-Tensor by auto} \\
&\text{moreover have } \ldots = 2^2(2^2(\text{nat } (\text{codomain-block ls}))) \\
&\quad \text{using Cons vert by auto} \\
&\text{moreover have } \ldots = 2^2(1 + \text{nat } (\text{codomain-block ls})) \\
&\quad \text{by auto} \\
&\text{moreover have } \ldots = 2^2(\text{nat } (\text{domain l}) + \text{nat } (\text{domain-block ls})) \\
&\quad \text{using codomain.simps vert by auto} \\
&\text{moreover have } \ldots = 2^2(\text{nat } (\text{domain l + domain-block ls})) \\
&\quad \text{using nat-sum Suc-eg-plus1-left Suc-nat-eg-nat-zadd1 codomain.simps(1) codomain-block-nonnegative nat-numeral numeral-One vert by (metis) } \\
&\text{moreover have } \ldots = 2^2(\text{nat } (\text{domain-block } (\text{l#ls}))) \\
&\quad \text{by auto} \\
&\text{ultimately show } \text{thesis by metis} \\
\text{next} \\
\text{case over} \\
&\text{have } \text{length } (\text{blockmat ls}) = 2^2(\text{nat } (\text{codomain-block ls}))
\end{align*}
using $\text{Cons by auto}$
then have $\text{length (blockmat (l#ls))}$
    $= (\text{length (brickmat l)})*(\text{length (blockmat ls)})$
using blockmat.simps rat-poly.length-Tensor by auto
moreover have $... = 4*(2 ^ \text{nat (codomain-block ls)})$
    using Cons over by auto
moreover have $... = 2^\text{(2 + nat (codomain-block ls))}$
    by auto
moreover have $... = 2^\text{(nat (codomain l) + nat (codomain-block ls))}$
    using codomain.simps over by auto
moreover have $... = 2^\text{(nat (codomain l + codomain-block ls))}$
    using nat-sum over codomain.simps codomain-block-nonnegative
    by auto
moreover have $... = 2^\text{(nat (codomain-block (l#ls)))}$
    by auto
ultimately show $?\text{thesis by metis}$
next
\begin{case}
derived
have $\text{length (blockmat ls)} = 2 ^ \text{nat (codomain-block ls)}$
    using Cons by auto
then have $\text{length (blockmat (l#ls))}$
    $= (\text{length (brickmat l)})*(\text{length (blockmat ls)})$
using blockmat.simps rat-poly.length-Tensor by auto
moreover have $... = 4*(2 ^ \text{nat (codomain-block ls)})$
    using Cons under by auto
moreover have $... = 2^\text{(2 + nat (codomain-block ls))}$
    by auto
moreover have $... = 2^\text{(nat (codomain l) + nat (codomain-block ls))}$
    using codomain.simps under by auto
moreover have $... = 2^\text{(nat (codomain l + codomain-block ls))}$
    using nat-sum under codomain.simps codomain-block-nonnegative
    by auto
moreover have $... = 2^\text{(nat (codomain-block (l#ls)))}$
    by auto
ultimately show $?\text{thesis by metis}$
next
\begin{case}
derived
have $\text{length (blockmat ls)} = 2 ^ \text{nat (codomain-block ls)}$
    using Cons by auto
then have $\text{length (blockmat (l#ls))}$
    $= (\text{length (brickmat l)})*(\text{length (blockmat ls)})$
using blockmat.simps rat-poly.length-Tensor by auto
moreover have $... = 4*(2 ^ \text{nat (codomain-block ls)})$
    using Cons cup by auto
moreover have $... = 2^\text{(2 + nat (codomain-block ls))}$
    by auto
moreover have $... = 2^\text{(nat (codomain l) + nat (codomain-block ls))}$
    using codomain.simps cap by auto
moreover have $... = 2^\text{(nat (codomain l + codomain-block ls))}$
using nat-sum cup codomain.simps
codomain-block-nonnegative
by auto
moreover have ... = 2^(nat (codomain-block (l#ls)))
by auto
ultimately show ?thesis by metis
next
case cap
have length (blockmat ls) = 2 ^ nat (codomain-block ls)
using Cons by auto
then have length (blockmat (l#ls))
  = (length (brickmat l))*(length (blockmat ls))
using blockmat.simps rat-poly.length-Tensor by auto
moreover have ... = 1*(2 ^ nat (codomain-block ls))
using Cons cap by auto
moreover have ... = 2^(0 + nat (codomain-block ls))
by auto
moreover have ... = 2^(nat (codomain l) + nat (codomain-block ls))
using codomain.simps cap by auto
moreover have ... = 2^(nat (codomain l + codomain-block ls))
using nat-sum cap codomain.simps codomain-block-nonnegative
by auto
moreover have ... = 2^(nat (codomain-block (l#ls)))
by auto
ultimately show ?thesis by metis
qed

lemma matrix-blockmat:
mat
(rat-poly.row-length (blockmat ls))
(length (blockmat ls))
(blockmat ls)
proof(induct ls)
case Nil
show ?case
using Nil
unfolding blockmat.simps(1) rat-poly.row-length-def mat-def
vec-def Ball-def by auto
next
case (Cons a ls)
have Cons-1:mat
  (rat-poly.row-length (blockmat ls))
  (length (blockmat ls))
  (blockmat ls)
using Cons by auto
have Cons-2:(blockmat (a#ls)) = (brickmat a)⊗(blockmat ls)
using blockmat.simps by auto

qed
moreover have \( \text{rat-poly.row-length (blockmat (a#ls))} \)
\[ = (\text{rat-poly.row-length (brickmat a)}) \]
\[ \times(\text{rat-poly.row-length (blockmat ls)}) \]
using calculation \( \text{rat-poly.row-length-mat by (metis)} \)
moreover have \( \text{length (blockmat (a#ls))} \)
\[ = (\text{length (brickmat a)}) \]
\[ \times(\text{length (blockmat ls)}) \]
using \( \text{blockmat.simps(2) rat-poly.length-Tensor by (metis)} \)
ultimately have \( \text{Cons-3:mat} \)
\[ (\text{rat-poly.row-length (brickmat a)}) \]
\[ (\text{length (brickmat a)}) \]
\[ (\text{brickmat a}) \]
\[ \Rightarrow \text{thesis using Cons-3 by auto} \]
then show \( \text{thesis using Cons-3 by auto} \)
proof\((\text{cases a})\)
\begin{align*}
\text{case vert} \\
\text{have mat} \\
& (\text{rat-poly.row-length (brickmat a)}) \\
& (\text{length (brickmat a)}) \\
& (\text{brickmat a}) \\
\text{using \text{vert-dim mat-vert rat-poly.matrix-row-length vert by metis}} \\
\text{thus \text{thesis using Cons-3 by auto}} \\
\text{next} \\
\text{case over} \\
\text{have mat} \\
& (\text{rat-poly.row-length (brickmat a)}) \\
& (\text{length (brickmat a)}) \\
& (\text{brickmat a}) \\
\text{using \text{mat-over rat-poly.matrix-row-length over by metis}} \\
\text{thus \text{thesis using Cons-3 by auto}} \\
\text{next} \\
\text{case under} \\
\text{have mat} \\
& (\text{rat-poly.row-length (brickmat a)}) \\
& (\text{length (brickmat a)}) \\
& (\text{brickmat a}) \\
\text{using \text{mat-under rat-poly.matrix-row-length under by metis}} \\
\text{thus \text{thesis using Cons-3 by auto}} \\
\text{next} \\
\text{case cap} \\
\text{have mat} \\
& (\text{rat-poly.row-length (brickmat a)}) \\
& (\text{length (brickmat a)}) \\
& (\text{brickmat a}) \\
\text{using \text{mat-cap rat-poly.matrix-row-length cap by metis}} \\
\text{thus \text{thesis using Cons-3 by auto}}
\end{align*}
next
case cup
have mat
  (rat-poly.row-length (brickmat a))
  (length (brickmat a))
using mat-cup rat-poly.matrix-row-length cup by metis
thus ?thesis using Cons-3 by auto
qed
qed

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 $1 \times 1$ matrix whose entry is the Kauffman bracket.

primrec kauff-mat :: wall ⇒ 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)
  ⇒ (rat-poly.row-length (kauff-mat ws)) = 2^(nat (domain-wall ws))
∧ (length (kauff-mat ws)) = 2^(nat (codomain-wall ws))
∧ (mat
  (rat-poly.row-length (kauff-mat ws))
  (length (kauff-mat ws))
  (kauff-mat ws)))
proof (induct ws)
case (basic w)
  show ?case
    using kauff-mat.simps(1) domain-wall.simps(1)
    row-length-domain-block matrix-blockmat
    length-codomain-block basic by auto
next
case (prod w ws)
  have is-tangle-diagram (w∗ws)
    using prod by auto
  moreover have prod-1:is-tangle-diagram ws
    using is-tangle-diagram.simps prod.prems by metis
  ultimately have prod-2:(codomain-block w) = domain-wall ws
    using is-tangle-diagram.simps by auto
  from prod-1 have prod-3:
    mat
      (rat-poly.row-length (kauff-mat ws))
      (length (kauff-mat ws))
      (kauff-mat ws)
using prod.hyps by auto
moreover have (rat-poly.row-length (kauff-mat ws))
  = \text{2}^\text{n} (\text{nat (domain-wall ws)})
  using prod.hyps prod-1 by auto
moreover have prod-4:length (kauff-mat ws)
  = \text{2}^\text{n} (\text{nat (codomain-wall ws)})
  using prod.hyps prod-1 by auto
moreover have prod-5:
  mat
    (rat-poly.row-length (blockmat w))
    (length (kauff-mat ws))
    (blockmat w)
  using matrix-blockmat by auto
moreover have prod-6:
  rat-poly.row-length (blockmat w)
  = \text{2}^\text{n} (\text{nat (domain-block w)})
  and length (blockmat w) = \text{2}^\text{n} (\text{nat (codomain-block w)})
  using row-length-domain-block length-codomain-block by auto
ultimately have ad1:length (blockmat w)
  = rat-poly.row-length (kauff-mat ws)
  using prod-2 by auto
then have mat
  (rat-poly.row-length (blockmat w))
  (length (kauff-mat ws))
  (rat-poly.mat-empty-column-length (blockmat w) (kauff-mat ws))
  using prod-3 prod-5 mat-mult by auto
then have res1:mat
  (rat-poly.row-length (blockmat w))
  (length (kauff-mat ws))
  (kauff-mat (w*ws))
  using kauff-mat.simps(2) by auto
then have rat-poly.row-length (kauff-mat (w*ws))
  = (rat-poly.row-length (blockmat w))
  using ad1 length-0-conv rat-poly.mat-empty-column-length
  rat-poly.matrix-row-length rat-poly.row-length-def
  rat-poly.unique-row-col(1) by (metis)
moreover have ...
  = \text{2}^\text{n} (\text{nat (domain-wall (w*ws))})
  using prod-6 domain-wall.simps by auto
ultimately have res2:
  rat-poly.row-length (kauff-mat (w*ws))
  = \text{2}^\text{n} (\text{nat (domain-wall (w*ws))})
  by auto
have length (kauff-mat (w*ws)) = length (kauff-mat ws)
  using res1 rat-poly.mat-empty-column-length
  rat-poly.matrix-row-length rat-poly.unique-row-col(2)
  by metis
moreover have ...
  = \text{2}^\text{n} (\text{nat (codomain-wall (w*ws))})
  using prod-4 codomain-wall.simps(2) by auto
ultimately have \[ \text{res3:} \text{length (kauff-mat (w*ws))} = 2^{\text{nat (codomain-wall (w*ws))}} \]

by auto

with \text{res1 res2} show ?case
using
(\text{length (kauff-mat ws)} = 2^{\text{nat (codomain-wall (w * ws))}})
(\text{rat-poly.row-length (blockmat w)} = 2^{\text{nat (domain-wall (w * ws))}})

by (metis)

qed

**Theorem** effective-matrix-kauff-mat:
assumes is-tangle-diagram ws
shows (\text{rat-poly.row-length (kauff-mat ws)}) = 2^{\text{nat (domain-wall ws)}}
and length (\text{kauff-mat ws}) = 2^{\text{nat (codomain-wall ws)}}
and mat (\text{rat-poly.row-length (kauff-mat ws)}) (length (\text{kauff-mat ws})) (\text{kauff-mat ws})

apply (auto simp add: matrix-kauff-mat assms)
using assms matrix-kauff-mat
by (metis)

**Lemma** mat-mult-equiv:
rat-poly.matrix-mult m1 m2 = mat-mult (rat-poly.row-length m1) m1 m2
by auto

**Theorem** associative-rat-poly-mat:
assumes mat (\text{rat-poly.row-length m1}) (\text{rat-poly.row-length m2}) m1
and mat (\text{rat-poly.row-length m2}) (\text{rat-poly.row-length m3}) m2
and mat (\text{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
have (rat-poly.matrix-mult m2 m3) = mat-mult (\text{rat-poly.row-length m2}) m2 m3
using mat-mult-equiv by auto

then have rat-poly.matrix-mult m1 (rat-poly.matrix-mult m2 m3) = mat-mult (\text{rat-poly.row-length m1}) m1 (mat-mult (\text{rat-poly.row-length m2}) m2 m3)
using mat-mult-equiv by auto

moreover have ... = mat-mult (\text{rat-poly.row-length m1}) (mat-mult (\text{rat-poly.row-length m1}) m1 m2) m3
using assms mat-mult-assoc by (metis)

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

proof
have mat
(\text{rat-poly.row-length m1})
(\text{rat-poly.row-length m3}) (rat-poly.matrix-mult m1 m2)
using assms(1) assms(2) mat-mult by (metis)

then have rat-poly.row-length (rat-poly.matrix-mult m1 m2) = (\text{rat-poly.row-length m1})

using assms(1) assms(2) length-0-conv rat-poly.mat-empty-column-length rat-poly.matrix-row-length rat-poly.row-length-Nil rat-poly.unique-row-col(1) rat-poly.unique-row-col(2)
by (metis)

moreover have rat-poly.matrix-mult (rat-poly.matrix-mult m1 m2) m3
  = mat-mult (rat-poly.row-length (rat-poly.matrix-mult m1 m2)) (rat-poly.matrix-mult m1 m2) m3
  using mat-mult-equiv by auto
then show ?thesis using mat-mult-equiv by (metis calculation)
qed
ultimately show ?thesis by auto
qed

It follows from this result that the Kauffman Matrix of a wall representing a link diagram, is a \( 1 \times 1 \) 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)
using assms effective-matrix-kauff-mat unfolding is-link-diagram-def by (metis Preliminaries.abs-zero abs-non-negative-sum(1) comm-monoid-add-class.add-0 nat-0 power-0)

**Theorem tangle-compose-matrix:**

((is-tangle-diagram ws1) \land (is-tangle-diagram ws2) \land (domain-wall ws2 = codomain-wall ws1)) \Rightarrow kauff-mat (ws1 \circ ws2) = rat-poly.matrix-mult (kauff-mat ws1) (kauff-mat ws2)

proof (induct ws1)
case (basic w1)
  have (basic w1) \circ (ws2) = (w1)\ast (ws2)
    using compose.simps by auto
  moreover have kauff-mat ((basic w1) \circ ws2) = rat-poly.matrix-mult (blockmat w1) (kauff-mat ws2)
    using kauff-mat.simps by auto
  then show ?case using kauff-mat.simps(1) by auto
next
case (prod w1 ws1)
  have 1:is-tangle-diagram (w1 \ast ws1)
    using prod.prems by (rule conjE)
  then have 2:(is-tangle-diagram ws1)
    \land (codomain-block w1 = domain-wall ws1)
    using is-tangle-diagram.simps(2) by metis
  then have
    mat (2\textsuperscript{\langle nat (domain-wall ws1)\rangle}) (2\textsuperscript{\langle nat (codomain-wall ws1)\rangle}) (kauff-mat ws1)
    and mat (2\textsuperscript{\langle nat (domain-block w1)\rangle}) (2\textsuperscript{\langle nat (codomain-block w1)\rangle}) (blockmat w1)
    using effective-matrix-kauff-mat matrix-blockmat length-codomain-block
\[ \text{row-length-domain-block} \]

by (auto) (metis)

with 2 have \( \exists \text{mat} \)

\begin{align*}
\text{(rat-poly.row-length (blockmat w1))} \\
\text{(}2^n \text{nat (domain-wall ws1))} \\
\text{(blockmat w1)}
\end{align*}

and \( \text{mat} \)

\begin{align*}
\text{(}2^n \text{nat (domain-wall ws1))} \\
\text{(}2^n \text{nat (domain-wall ws2))} \\
\text{(kauff-mat ws1)}
\end{align*}

and \( (2^n \text{nat (domain-wall ws1))} \)

= \text{(rat-poly.row-length (kauff-mat ws1))} 

using \text{effective-matrix-kauff-mat} \ prod.prems \text{matrix-blockmat}

\text{row-length-domain-block} \text{ by auto}

then have \( \text{mat} \)

\begin{align*}
\text{(rat-poly.row-length (blockmat w1))} \\
\text{(rat-poly.row-length (kauff-mat ws1))} \\
\text{(blockmat w1)}
\end{align*}

and \( \text{mat} \)

\begin{align*}
\text{(rat-poly.row-length (kauff-mat ws1))} \\
\text{(}2^n \text{nat (domain-wall ws2))} \\
\text{(kauff-mat ws1)}
\end{align*}

by (auto)

moreover have \( \text{mat} \)

\begin{align*}
\text{(}2^n \text{nat (domain-wall ws2))} \\
\text{(}2^n \text{nat (codomain-wall ws2))} \\
\text{(kauff-mat ws2)}
\end{align*}

and \( (2^n \text{nat (domain-wall ws2))} \)

= \text{rat-poly.row-length (kauff-mat ws2)} 

using \text{prod.prems} \text{effective-matrix-kauff-mat}

\text{effective-matrix-kauff-mat}

by (auto) (metis \text{prod.prems})

ultimately have \( \text{mat} \)

\begin{align*}
\text{(rat-poly.row-length (blockmat w1))} \\
\text{(rat-poly.row-length (kauff-mat ws1))} \\
\text{(blockmat w1)}
\end{align*}

and \( \text{mat} \)

\begin{align*}
\text{(rat-poly.row-length (kauff-mat ws1))} \\
\text{(rat-poly.row-length (kauff-mat ws2))} \\
\text{(kauff-mat ws1)}
\end{align*}

and \( \text{mat} \)

\begin{align*}
\text{(rat-poly.row-length (kauff-mat ws2))} \\
\text{(}2^n \text{nat (codomain-wall ws2))} \\
\text{(kauff-mat ws2)}
\end{align*}

by (auto)

with 3 have \( \text{rat-poly.matrix-mult} \)

\begin{align*}
\text{(blockmat w1)} \\
\text{(rat-poly.matrix-mult (kauff-mat ws1))} \\
\text{(kauff-mat ws2)}
\end{align*}
\[ \text{left-mat-compose:} \]
\[ \text{assumes is-tangle-diagram ws} \]  
\[ \text{and codomain-wall ws} = 0 \]  
\[ \text{shows kauff-mat ws} = (\text{kauff-mat (ws o (basic []))}) \]  
\[ \text{proof} - \]
\[ \text{have mat (rat-poly.row-length (kauff-mat ws)) 1 (kauff-mat ws)} \]  
\[ \text{using effective-matrix-kauff-mat assms nat-0 power-0 by metis} \]  
\[ \text{moreover have (kauff-mat (basic [])) = mat1 1} \]  
\[ \text{using kauff-mat.simps(1) blockmat.simps(1) mat1-equiv by auto} \]  
\[ \text{moreover then have 1:(kauff-mat (ws o (basic [])))} \]  
\[ = \text{rat-poly.matrix-mult} \]  
\[ (\text{kauff-mat ws}) \]  
\[ (\text{kauff-mat (basic [])}) \]  
\[ \text{using tangle-compose-matrix assms is-tangle-diagram.simps by auto} \]  
\[ \text{ultimately have rat-poly.matrix-mult (kauff-mat ws) (kauff-mat (basic []))} \]  
\[ = (\text{kauff-mat ws}) \]  
\[ \text{using mat-mult-equiv mat1-mult-right by auto} \]  
\[ \text{then show ?thesis using 1 by auto} \]  
\[ \text{qed} \]
rat-poly.unique-row-col(1)
by metis
then show thesis using 1 by auto
qed

lemma left-id-blockmat:blockmat [] ⊗ blockmat b = blockmat b
unfolding blockmat.simps(1) rat-poly.Tensor-left-id by auto

lemma tens-assoc:
∀ a xs ys.(brickmat a ⊗ (blockmat xs ⊗ blockmat ys) 
= (brickmat a ⊗ blockmat xs) ⊗ blockmat ys)
proof-
have ∀ a.(mat
(rat-poly.row-length (brickmat a))
(length (brickmat a))
(brickmat a))
using brickmat.simps
unfolding mat-def rat-poly.row-length-def Ball-def vec-def
apply(auto)
by (case-tac a) (auto)
moreover have ∀ xs. (mat
(rat-poly.row-length (blockmat xs))
(length (blockmat xs))
(blockmat xs))
using matrix-blockmat by auto
moreover have ∀ ys. mat
(rat-poly.row-length (blockmat ys))
(length (blockmat ys))
(blockmat ys)
using matrix-blockmat by auto
ultimately show thesis using rat-poly.associativity by auto
qed

lemma kauff-mat-tensor-distrib:
∀ xs∀ ys.(kauff-mat (basic xs ⊗ basic ys)
= kauff-mat (basic xs) ⊗ kauff-mat (basic ys))
apply(rule allI)
apply (rule allI)
apply (induct-tac xs)
apply(auto)
apply (metis rat-poly.vec-mat-Tensor-vector-id)
apply (simp add:tens-assoc)
done

lemma blockmat-tensor-distrib:
(blockmat (a ⊗ b)) = (blockmat a) ⊗ (blockmat b)
proof-
have blockmat (a ⊗ b) = kauff-mat (basic (a ⊗ b))
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 kauff-mat ws ≠ [] 
proof−
  have (length (kauff-mat ws) = 2^(nat (codomain-wall ws))) 
    using effective-matrix-kauff-mat assms by auto 
  then have (length (kauff-mat ws)) ≥ 1 
    by auto 
  then show ?thesis by auto 
qed 

lemma is-tangle-diagram-length-rowlength: 
  assumes is-tangle-diagram (w*w) 
  shows length (blockmat w) = rat-poly.row-length (kauff-mat w) 
proof−
  have (codomain-block w = domain-wall ws) 
    using assms is-tangle-diagram.simps by metis 
  moreover have rat-poly.row-length (kauff-mat ws) = 2^(nat (domain-wall ws)) 
    using effective-matrix-kauff-mat by (metis assms is-tangle-diagram.simps(2)) 
  moreover have length (blockmat w) = 2^(nat (codomain-block w)) 
    using matrix-blockmat length-codomain-block by auto 
  ultimately show ?thesis by auto 
qed
lemma is-tangle-diagram-matrix-match:
  assumes is-tangle-diagram (w1 * ws1)
  and is-tangle-diagram (w2 * ws2)
  shows rat-poly.matrix-match (blockmat w1)
  (kauff-mat ws1) (blockmat w2) (kauff-mat ws2)
  unfolding rat-poly.matrix-match-def
  apply(auto)
  proof -
    show mat (rat-poly.row-length (blockmat w1)) (length (blockmat w1)) (blockmat w1)
      using matrix-blockmat by auto
    next
    have is-tangle-diagram ws1
      using assms(1) is-tangle-diagram.simps(2) by metis
    then show mat (rat-poly.row-length (kauff-mat ws1)) (length (kauff-mat ws1))
      (kauff-mat ws1)
      using matrix-kauff-mat by metis
    next
    show mat (rat-poly.row-length (blockmat w2)) (length (blockmat w2)) (blockmat w2)
      using matrix-blockmat by auto
    next
    have is-tangle-diagram ws2
      using assms(2) is-tangle-diagram.simps(2) by metis
    then show mat (rat-poly.row-length (kauff-mat ws2)) (length (kauff-mat ws2))
      (kauff-mat ws2)
      using matrix-kauff-mat by metis
    next
    show length (blockmat w1) = rat-poly.row-length (kauff-mat ws1)
      using is-tangle-diagram-length-rowlength assms(1) by auto
    next
    show length (blockmat w2) = rat-poly.row-length (kauff-mat ws2)
      using is-tangle-diagram-length-rowlength assms(2) by auto
    next
    assume 0: blockmat w1 = []
    show False using 0
      by (metis blockmat-non-empty)
    next
    assume 1: kauff-mat ws1 = []
    have is-tangle-diagram ws1
      using assms(1) is-tangle-diagram.simps(2) by metis
    then show False using 1 kauff-mat-non-empty by auto
    next
    assume 0: blockmat w2 = []
    show False using 0
      by (metis blockmat-non-empty)
    next
    assume 1: kauff-mat ws2 = []
have is-tangle-diagram ws2
  using assms(2) is-tangle-diagram.simps(2) by metis
then show False using 1 kauff-mat-non-empty by auto
qed

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

primrec make-vert-equiv :: nat ⇒ rat-poly mat
where
|make-vert-equiv 0 = [[1]]
|make-vert-equiv (Suc k) = ((mat1 2)⊗(make-vert-equiv k))

lemma mve1:make-vert-equiv 1 = (mat1 2)
using make-vert-equiv.simps brickmat.simps(1)
One-nat-def rat-poly.Tensor-right-id
by (metis)

lemma assumes i<2 and j<2
shows (make-vert-equiv 1)!i!j = (if i = j then 1 else 0)
apply(simp add:mve1)
apply(simp add:rat-poly.Tensor-right-id)
using make-vert-equiv.simps mat1-index assms by (metis)

lemma mat1-vert-equiv:(mat1 2) = (brickmat vert) (is ?l = ?r)
proof–
  have ?r = [[1,0],[0,1]]
    using brickmat.simps by auto
  then have rat-poly.row-length ?r = 2 and length ?r = 2
    using rat-poly.row-length-def by auto
  moreover then have 1:mat 2 2 ?r
    using mat-vert by metis
  ultimately have 2:(∀ i < 2. ∀ j < 2.
    ((?r) ! i ! j = (if i = j then 1 else 0)))
    by (auto dest!: less-2-cases)
  qed

have 3:mat 2 2 (mat1 2)
  by (metis mat1)

have 4:(∀ i < 2. ∀ j < 2.
    ((?l) ! i ! j = (if i = j then 1 else 0)))
  by (metis mat1-index)
then have \(\forall i < 2. \forall j < 2. ((\text{?l}) ! i ! j = (?r ! i ! j))\)
   using 2 by auto
with 1 3 have \(\text{?l} = ?r\)
   by (metis mat-eqI)
then show \(\text{thesis}\) by auto
qed

lemma blockmat-make-vert:
blockmat (make-vert-block n) = (make-vert-equiv n)
apply (induction n)
apply (simp)
unfolding make-vert-block.simps blockmat.simps make-vert-equiv.simps
using mat1-vert-equiv by auto

lemma prop-make-vert-equiv:
shows rat-poly.row-length (make-vert-equiv n) = \(2^n\)
  and length (make-vert-equiv n) = \(2^n\)
  and mat
    (rat-poly.row-length (make-vert-equiv n))
    (length (make-vert-equiv n))
    (make-vert-equiv n)
proof –
  have 1:make-vert-equiv n = (blockmat (make-vert-block n))
    using blockmat-make-vert by auto
moreover have 2:domain-block (make-vert-block n) = int n
    using domain-make-vert by auto
moreover have 3:codomain-block (make-vert-block n) = int n
    using codomain-make-vert by auto
ultimately show rat-poly.row-length (make-vert-equiv n) = \(2^n\)
    and length (make-vert-equiv n) = \(2^n\)
    and mat
      (rat-poly.row-length (make-vert-equiv n))
      (length (make-vert-equiv n))
      (make-vert-equiv n)
    apply (metis nat-int row-length-domain-block)
  using 1 2 3 apply (metis length-codomain-block nat-int)
  using 1 2 3 by (metis matrix-blockmat)
qed

abbreviation nat-mult::nat ⇒ nat ⇒ nat \((\text{infixl } \ast n 65)\)
where
nat-mult a b ≡ ((a::nat)*b)

lemma equal-div-mod: assumes \((j::nat) \div a = (i \div a)\)
    and \((j \mod a) = (i \mod a)\)
shows \(j = i\)
proof –
  have \(j = a * (j \div a) + (j \mod a)\)
    by auto
then have $j = a \ast (i \div a) + (i \mod a)$
using assms by auto
then show ?thesis by auto
qed

lemma equal-div-mod2:($(j::nat) \div a) = (i \div a)$
\wedge ((j \mod a) = (i \mod a)) = (j = i)$
using equal-div-mod by metis

lemma impl-rule:
assumes $(\forall i < m. \forall j < n. (P i) \land (Q j))$
and $(\forall i \ j. (P i) \land (Q j) \rightarrow R i j)$
shows $(\forall i < m. \forall j < n. R i j)$
using assms by metis

lemma implic:
assumes $(\forall i \ j. ((P i \ j) \rightarrow (Q i \ j)))$
and $(\forall i \ j. ((Q i \ j) \rightarrow (R i \ j)))$
shows $(\forall i \ j. ((P i \ j) \rightarrow (R i \ j)))$
using assms by auto

lemma assumes $a < (b \ast c)$
shows $(a::nat) \div b < c$
using assms by (metis rat-poly.div-right-ineq)

lemma mult-if-then:($(v = (if P then 1 else 0))$
\wedge $(w = (if Q then 1 else 0)))
\implies (rat-poly-times v w = (if (P \land Q) then 1 else 0))$
by auto

lemma rat-poly-unity:rat-poly-times 1 1 = 1
by auto

lemma $(P \land Q) \rightarrow R \implies (P \rightarrow Q \rightarrow R)$
by auto
lemma length $(mat1 2) = 2$
apply(simp add:mat1I-def)
done

theorem make-vert-equiv-mat:
make-vert-equiv n = $(mat1 (2^n))$
proof(induction n)
case 0
show ?case using 0 mat1-equiv by auto
next
case $(Suc k)$
have 1:make-vert-equiv k = $(mat1 (2^k))$
using Suc by auto
moreover then have make-vert-equiv $(k+1) = (mat1 2) \otimes (mat1 (2^k))$
using `make-vert-equiv.simps(2)` by auto
then have `(mat1 2) ⊗ (mat1 (2^k)) = mat1 (2^(k+1))`

proof

have `1:mat (2^(k+1)) (2^(k+1)) (mat1 (2^(k+1)))`
using `mat1` by auto
have `2:(∀ i < 2^(k+1). ∀ j < 2^(k+1).
(mat1 (2^(k+1)) ![i ![j = (if i = j then 1 else 0)))]
by (metis `mat1-index`
have `3:rat-poly.row-length (mat1 2) = 2`
by (metis `mat1-vert-equiv vert-dim`
have `4:length (mat1 2) = 2`
by (simp add: `mat1I-def`
then have `5:mat
(rat-poly.row-length (mat1 2))
(length (mat1 2))
(mat1 2)`
by (metis `4 mat1 mat1-vert-equiv vert-dim`
moreover have `6:rat-poly.row-length (mat1 (2^k)) = 2^k`
and `7:length ((mat1 (2^k))) = 2^k`
using `Suc`
by (metis prop-make-vert-equiv (1)) (simp add: `mat1I-def`
then have `8:mat
(rat-poly.row-length (mat1 (2^k)))
(length (mat1 (2^k)))
(mat1 (2^k))`
using `Suc mat1` by (metis)
then have `9:
(∀ i < (2^(k+1)). ∀ j < (2^(k+1)).
((rat-poly.Tensor (mat1 2) (mat1 (2^k))![i ![j = rat-poly-times
(((mat1 2) ![j div (length (mat1 (2^k))))
![i div (rat-poly.row-length (mat1 (2^k)))])))
(((mat1 (2^k)) ![j mod length (mat1 (2^k)))]
![i mod (rat-poly.row-length (mat1 (2^k)))]))))`
proof

have `(∀ i < ((rat-poly.row-length (mat1 2))
* n (rat-poly.row-length (mat1 (2^k))))).
∀ j < ((length (mat1 2))
* n (length (mat1 (2^k))))).
(rat-poly-times
(((mat1 2) ![j div (length (mat1 (2^k)))]
![i div (rat-poly.row-length (mat1 (2^k)))])))
(((mat1 (2^k)) ![j mod length (mat1 (2^k)))]
![i mod (rat-poly.row-length (mat1 (2^k)))]))))`
using `5 8 rat-poly.effective-matrix-Tensor-elements2`
by (metis `3 4 6 7 rat-poly.comm`
mOREOVER have (rat-poly.row-length (mat1 2))

70
\[ n(\text{rat-poly}.\text{row-length}(\text{mat1}(2^k))) \]
\[ = 2^{\lceil k+1 \rceil} \]

**using** 3 6 by auto

moreover have \( (\text{length}(\text{mat1}(2))) \)
\[ n(\text{length}(\text{mat1}(2^k))) \]
\[ = 2^{\lceil k+1 \rceil} \]

**using** 4 7 by (metis 3 6 calculation(2))

ultimately show \( ?\text{thesis by metis} \)

**qed**

have 10: \( \forall i \ j. ((i \ \text{div} (\text{rat-poly}.\text{row-length}(\text{mat1}(2^k)))) < 2) \)
\[ \land (j \ \text{div} \ \text{length}(\text{mat1}(2^k))) < 2 \]
\[ \longrightarrow ((\text{mat1}(2))!((j \ \text{div} \ \text{length}(\text{mat1}(2^k)))) \)
\[ = (i \ \text{div} (\text{rat-poly}.\text{row-length}(\text{mat1}(2^k)))) \]
\[ = (\text{if} \]
\[ ((j \ \text{div} \ \text{length}(\text{mat1}(2^k)))) \]
\[ = (i \ \text{div} (\text{rat-poly}.\text{row-length}(\text{mat1}(2^k)))) \]
\[ \text{then} \ 1 \]
\[ \text{else} \ 0)) \]

**using** \( \text{mat1-index by metis 6 7} \)

have 11: \( \forall j. (j < (2^{\lceil k+1 \rceil})) \longrightarrow j \ \text{div} \ \text{length}(\text{mat1}(2^k))) < 2 \)

**proof** –

have \( 2^{\lceil k+1 \rceil} = (2 \ast n (2^k)) \)

**by auto**

then show \( ?\text{thesis} \)

**using** 7 allI Suc.IH prop-make-vert-equiv(1)

rat-poly.div-left-ineq by (metis)

**qed**

moreover have 12:
\[ \forall i. (i < (2^{\lceil k+1 \rceil})) \]
\[ \longrightarrow (i \ \text{div} (\text{rat-poly}.\text{row-length}(\text{mat1}(2^k)))) < 2 \]

**proof** –

have \( 2^{\lceil k+1 \rceil} = (2 \ast n (2^k)) \)

**by auto**

then show \( ?\text{thesis} \)

**using** 7 allI by (metis Suc.IH prop-make-vert-equiv(1))

rat-poly.div-left-ineq by (metis)

**qed**

ultimately have 13:
\[ \forall i \ j. ((i < (2^{\lceil k+1 \rceil})) \land j < (2^{\lceil k+1 \rceil})) \longrightarrow \]
\[ ((i \ \text{div} (\text{rat-poly}.\text{row-length}(\text{mat1}(2^k)))) < 2) \]
\[ \land (j \ \text{div} \ \text{length}(\text{mat1}(2^k))) < 2 \]

**by auto**

have 14: \( \forall i \ j. (i < (2^{\lceil k+1 \rceil})) \land (j < (2^{\lceil k+1 \rceil})) \longrightarrow \)
\[ (((\text{mat1}(2)) \)
\[ !((j \ \text{div} \ \text{length}(\text{mat1}(2^k)))) \]
\[ = (i \ \text{div} (\text{rat-poly}.\text{row-length}(\text{mat1}(2^k)))) \]
\[ = (\text{if} \]
\[ ((j \ \text{div} \ \text{length}(\text{mat1}(2^k)))) \]
\[ = (i \ \text{div} (\text{rat-poly}.\text{row-length}(\text{mat1}(2^k)))) \]
\[ \text{then} \ 1 \]

71
\begin{verbatim}
apply(rule allI)
apply(rule allI)
proof
fix i j
assume 0:(i::nat) < 2 ^ (k + 1) \land (j::nat) < 2 ^ (k + 1)
have ((i div (rat-poly.row-length (mat1 (2 \^ k)))) < 2)
\land ((j div (length (mat1 (2 \^ k)))) < 2)
  using 0 13 by auto
then show \(((mat1 2) (! (j div (length (mat1 (2 \^ k)))) )
\land (! (i div (rat-poly.row-length (mat1 (2 \^ k)))) )
= (if ((j div (length (mat1 (2 \^ k)))) )
  = (i div (rat-poly.row-length (mat1 (2 \^ k)))) )
  then 1
  else 0))
  using 10 by (metis 6)
qed
have 15:\forall i j.(((i mod (rat-poly.row-length (mat1 (2 \^ k)))) < 2 \^ k) 
\land (j mod length (mat1 (2 \^ k))) < 2 \^ k)
\longrightarrow (((mat1 2) (i mod (rat-poly.row-length (mat1 (2 \^ k)))) )
\land (j mod (length (mat1 (2 \^ k)))) )
= (if 
  ((j mod (length (mat1 (2 \^ k)))) )
  = (i mod (rat-poly.row-length (mat1 (2 \^ k)))) )
  then 1
  else 0))
  using mat1-index by (metis 6 7)
have 16:\forall j.(j < (2 \^ (k+1)) \longrightarrow j mod (length (mat1 (2 \^ k))) < 2 \^ k)
proof-
have 2 \^ (k+1) = (2 \ast n (2 \^ k))
  by auto
then show \?thesis
  using 7 allI mod-less-divisor
  nat-zero-less-power-iff zero-less-numeral by (metis)
qed
moreover have 17:\forall i.(i < (2 ^ (k+1)))
\longrightarrow (i mod (rat-poly.row-length (mat1 (2 \^ k)))) < 2 \^ k)
proof-
have 2 \^ (k+1) = (2 \ast n (2 \^ k))
  by auto
then show \?thesis using 7 allI by (metis 6 calculation)
qed
ultimately have 18:
\forall i j.((i < (2 ^ (k+1))) \land j < (2 ^ (k+1)) \longrightarrow ((i mod (rat-poly.row-length (mat1 (2 \^ k)))) )
\land (j mod (length (mat1 (2 \^ k)))) < 2 \^ k)
\end{verbatim}
by (metis 7)
have 19:\forall i,j.(i < (2^{(k+1)}))\land (j < (2^{(k+1)})) \rightarrow
((mat1 (2^k))
  ![j mod (length (mat1 (2^k)))])
![i mod (rat-poly.row-length (mat1 (2^k)))])
= (if
  ((j mod (length (mat1 (2^k)))))
  = (i mod (rat-poly.row-length (mat1 (2^k)))))
then 1
else 0)
apply(rule allI)
apply(rule allI)
proof
fix i j
assume 0:(i::nat) < 2 ^ (k + 1) \land (j::nat) < 2 ^ (k + 1)
have ((i mod (rat-poly.row-length (mat1 (2^k))))) < 2^k
  \land((j mod (length (mat1 (2^k))))) < 2^k
using 0 18 by auto
then show (((mat1 (2^k))
  ![j mod (length (mat1 (2^k)))])
  !(i mod(rat-poly.row-length (mat1 (2^k)))))
= (if
  ((j mod (length (mat1 (2^k)))))
  then 1
  else 0))
  using 15 by (metis 6)
qed
have (\forall i, \forall j.
  (i <(2^{(k+1)})) \land (j < (2^{(k+1)}))
  \rightarrow rat-poly-times
  ((mat1 2)
    ![j div (length (mat1 (2^k)))])
    ![i div (rat-poly.row-length (mat1 (2^k)))])
  ((mat1 (2^k))
    ![j mod length (mat1 (2^k))])
    !(i mod (rat-poly.row-length (mat1 (2^k)))))
  = (if
    ((j div (length (mat1 (2^k)))))
    = (i div (rat-poly.row-length (mat1 (2^k)))))
    \land((j mod (length (mat1 (2^k)))))
    = (i mod (rat-poly.row-length (mat1 (2^k)))))
then 1
else 0))
apply(rule allI)
apply(rule allI)
proof
fix i j
assume 0: \((i::\text{nat}) < 2^{(k+1)}) \land (j::\text{nat}) < (2^{(k+1)})\)

have s1: \((\text{mat1} \ 2)
\begin{align*}
(i \text{ div} \ (\text{length} \ (\text{mat1} \ (2^k)))) \\
(i \text{ div} \ (\text{rat-poly.row-length} \ (\text{mat1} \ (2^k))))
\end{align*}
= (\begin{align*}
((i \text{ div} \ (\text{length} \ (\text{mat1} \ (2^k))))
= (i \text{ div} \ (\text{rat-poly.row-length} \ (\text{mat1} \ (2^k))))
\end{align*}
\begin{align*}
\text{then} \ 1 \\
\text{else} \ 0
\end{align*})

using 0 14 by metis

moreover have s2: \((\text{mat1} \ (2^k))
\begin{align*}
(j \text{ mod} \ (\text{length} \ (\text{mat1} \ (2^k)))) \\
(i \text{ mod} \ (\text{rat-poly.row-length} \ (\text{mat1} \ (2^k))))
\end{align*}
= (\begin{align*}
((j \text{ mod} \ (\text{length} \ (\text{mat1} \ (2^k))))
= (i \text{ mod} \ (\text{rat-poly.row-length} \ (\text{mat1} \ (2^k))))
\end{align*}
\begin{align*}
\text{then} \ 1 \\
\text{else} \ 0
\end{align*})

using 0 19 by metis

show \text{rat-poly-times}
\begin{align*}
((\text{mat1} \ 2)
\begin{align*}
(j \text{ div} \ (\text{length} \ (\text{mat1} \ (2^k)))) \\
(i \text{ div} \ (\text{rat-poly.row-length} \ (\text{mat1} \ (2^k))))
\end{align*}
((\text{mat1} \ (2^k))
\begin{align*}
(j \text{ mod} \ (\text{length} \ (\text{mat1} \ (2^k)))) \\
(i \text{ mod} \ (\text{rat-poly.row-length} \ (\text{mat1} \ (2^k))))
\end{align*}
= (\begin{align*}
((j \text{ div} \ (\text{length} \ (\text{mat1} \ (2^k))))
= (i \text{ div} \ (\text{rat-poly.row-length} \ (\text{mat1} \ (2^k))))
\end{align*}
\begin{align*}
\text{then} \ 1 \\
\text{else} \ 0
\end{align*})
\begin{align*}
\text{apply}(\text{simp}) \\
\text{apply}(\text{rule conjI})
\end{align*}

proof–

show \(j \text{ div} \ (\text{length} \ (\text{mat1} \ (2^k))) = i \text{ div} \ (\text{rat-poly.row-length} \ (\text{mat1} \ (2^k))) \land (j \text{ mod} \ (\text{length} \ (\text{mat1} \ (2^k))) = i \text{ mod} \ (\text{rat-poly.row-length} \ (\text{mat1} \ (2^k)))\))

\rightarrow \text{rat-poly-times}
\begin{align*}
((\text{mat1} \ 2)
\begin{align*}
(j \text{ div} \ (\text{length} \ (\text{mat1} \ (2^k)))) \\
(i \text{ div} \ (\text{rat-poly.row-length} \ (\text{mat1} \ (2^k))))
\end{align*}
((\text{mat1} \ (2^k))
\begin{align*}
(j \text{ mod} \ (\text{length} \ (\text{mat1} \ (2^k)))) \\
(i \text{ mod} \ (\text{rat-poly.row-length} \ (\text{mat1} \ (2^k))))
\end{align*}
= 1

proof–
have
\[ j \div \text{length} (\text{mat1} (2^k)) = i \div \text{rat-poly.row-length} (\text{mat1} (2^k)) \]
\[ \land j \mod \text{length} (\text{mat1} (2^k)) = i \mod \text{rat-poly.row-length} (\text{mat1} (2^k)) \]
\[ \implies \text{rat-poly-times} (\text{mat1} 2 ! (j \div \text{length} (\text{mat1} (2^k))) ! (i \div \text{rat-poly.row-length} (\text{mat1} (2^k)))) (\text{mat1} (2^k) ! (j \mod \text{length} (\text{mat1} (2^k))) ! (i \mod \text{rat-poly.row-length} (\text{mat1} (2^k)))) = 1 \]

proof−
assume local-assms:
\[ j \div \text{length} (\text{mat1} (2^k)) = i \div \text{rat-poly.row-length} (\text{mat1} (2^k)) \]
\[ \land j \mod \text{length} (\text{mat1} (2^k)) = i \mod \text{rat-poly.row-length} (\text{mat1} (2^k)) \]

have (\text{mat1} 2 ! (j \div \text{length} (\text{mat1} (2^k))) ! (i \div \text{rat-poly.row-length} (\text{mat1} (2^k)))) = 1

using s1 local-assms by metis
moreover have (\text{mat1} (2^k) ! (j \mod \text{length} (\text{mat1} (2^k))) ! (i \mod \text{rat-poly.row-length} (\text{mat1} (2^k)))) = 1

using s2 local-assms by metis
ultimately show \(?thesis
by (metis ?thesis prop-make-vert-equiv(1) prop-make-vert-equiv(2) rat-poly.right-id)

qed

then show \(?thesis by auto

qed

show
\[ (j \div \text{length} (\text{mat1} (2^k)) = i \div \text{rat-poly.row-length} (\text{mat1} (2^k)) \rightarrow j \mod \text{length} (\text{mat1} (2^k)) \neq i \mod \text{rat-poly.row-length} (\text{mat1} (2^k))) \rightarrow (j \div \text{length} (\text{mat1} (2^k))) ! (j \mod \text{length} (\text{mat1} (2^k))) = 0 \lor (\text{mat1} (2^k) ! (j \mod \text{length} (\text{mat1} (2^k))) ! (i \mod \text{rat-poly.row-length} (\text{mat1} (2^k)))) = 0 \]

proof−

have (j \div \text{length} (\text{mat1} (2^k)) = i \div \text{rat-poly.row-length} (\text{mat1} (2^k)) \land j \mod \text{length} (\text{mat1} (2^k)) \neq i \mod \text{rat-poly.row-length} (\text{mat1} (2^k))) \rightarrow (j \div \text{length} (\text{mat1} (2^k))) ! (j \mod \text{length} (\text{mat1} (2^k))) = 0 \lor (\text{mat1} (2^k) ! (j \mod \text{length} (\text{mat1} (2^k))) ! (i \mod \text{rat-poly.row-length} (\text{mat1} (2^k)))) = 0

proof−

assume local-assms:
\[ (j \div \text{length} (\text{mat1} (2^k)) = i \div \text{rat-poly.row-length} (\text{mat1} (2^k)) \land j \mod \text{length} (\text{mat1} (2^k)) \neq i \mod \text{rat-poly.row-length} (\text{mat1} (2^k))) \rightarrow (j \div \text{length} (\text{mat1} (2^k))) ! (j \mod \text{length} (\text{mat1} (2^k))) = 0 \lor (\text{mat1} (2^k) ! (j \mod \text{length} (\text{mat1} (2^k))) ! (i \mod \text{rat-poly.row-length} (\text{mat1} (2^k)))) = 0

proof−

assume local-assms:
have mat1 \( (2 \cdot 2^k) \) ! \((j \mod \text{length}(\text{mat1} \ (2 \cdot 2^k)))! (i \mod \text{rat-poly.row-length} \ (\text{mat1} \ (2 \cdot 2^k))) = 0 \)

using s2 local-assms by metis
then show ?thesis by auto
qed
then have l:
\((j \div \text{length}(\text{mat1} \ (2 \cdot 2^k))) = i \div \text{rat-poly.row-length} \ (\text{mat1} \ (2 \cdot 2^k)) \)
\(\wedge\) \((j \mod \text{length}(\text{mat1} \ (2 \cdot 2^k))) \neq i \mod \text{rat-poly.row-length} \ (\text{mat1} \ (2 \cdot 2^k))) \)
\(\longrightarrow\)
mat1 \(2 \cdot 2^k \) ! \((j \div \text{length}(\text{mat1} \ (2 \cdot 2^k)))! (i \div \text{rat-poly.row-length} \ (\text{mat1} \ (2 \cdot 2^k))) \)
\(\forall\mat1 \ (2 \cdot 2^k) \) ! \((j \mod \text{length}(\text{mat1} \ (2 \cdot 2^k)))! (i \mod \text{rat-poly.row-length} \ (\text{mat1} \ (2 \cdot 2^k))) = 0 \)
by auto
show \((j \div \text{length}(\text{mat1} \ (2 \cdot 2^k))) = i \div \text{rat-poly.row-length} \ (\text{mat1} \ (2 \cdot 2^k)) \)
\(\longrightarrow\)
\((j \mod \text{length}(\text{mat1} \ (2 \cdot 2^k))) \neq i \mod \text{rat-poly.row-length} \ (\text{mat1} \ (2 \cdot 2^k))) \)
\(\Longrightarrow\)
mat1 \(2 \cdot 2^k \) ! \((j \div \text{length}(\text{mat1} \ (2 \cdot 2^k)))! (i \div \text{rat-poly.row-length} \ (\text{mat1} \ (2 \cdot 2^k))) \)
\(\forall\mat1 \ (2 \cdot 2^k) \) ! \((j \mod \text{length}(\text{mat1} \ (2 \cdot 2^k)))! (i \mod \text{rat-poly.row-length} \ (\text{mat1} \ (2 \cdot 2^k))) = 0 \)
proof-
have \((j \div \text{length}(\text{mat1} \ (2 \cdot 2^k))) = i \div \text{rat-poly.row-length} \ (\text{mat1} \ (2 \cdot 2^k)) \)
\(\longrightarrow\)
\((j \mod \text{length}(\text{mat1} \ (2 \cdot 2^k))) \neq i \mod \text{rat-poly.row-length} \ (\text{mat1} \ (2 \cdot 2^k))) \)
\(\Longrightarrow\)
mat1 \(2 \cdot 2^k \) ! \((j \div \text{length}(\text{mat1} \ (2 \cdot 2^k)))! (i \div \text{rat-poly.row-length} \ (\text{mat1} \ (2 \cdot 2^k))) \)
\(\forall\mat1 \ (2 \cdot 2^k) \) ! \((j \mod \text{length}(\text{mat1} \ (2 \cdot 2^k)))! (i \mod \text{rat-poly.row-length} \ (\text{mat1} \ (2 \cdot 2^k))) = 0 \)
assume local-assm1:
\((j \div \text{length}(\text{mat1} \ (2 \cdot 2^k))) = i \div \text{rat-poly.row-length} \ (\text{mat1} \ (2 \cdot 2^k)) \)
\(\longrightarrow\)
\((j \mod \text{length}(\text{mat1} \ (2 \cdot 2^k))) \neq i \mod \text{rat-poly.row-length} \ (\text{mat1} \ (2 \cdot 2^k))) \)
have \((j \div \text{length}(\text{mat1} \ (2 \cdot 2^k))) = i \div \text{rat-poly.row-length} \ (\text{mat1} \ (2 \cdot 2^k)) \)
\(\Longrightarrow\)
mat1 \((2 \cdot 2^k) \) ! \((j \mod \text{length}(\text{mat1} \ (2 \cdot 2^k)))! (i \mod \text{rat-poly.row-length} \ (\text{mat1} \ (2 \cdot 2^k))) = 0 \)
using s2 local-assm1 by (metis 7)
then have l1: \((j \div \text{length}(\text{mat1} \ (2 \cdot 2^k))) = i \div \text{rat-poly.row-length} \ (\text{mat1} \ (2 \cdot 2^k)))
\(\Longrightarrow\) ?thesis
by auto
moreover have \((j \div \text{length}(\text{mat1} \ (2 \cdot 2^k))) = i \div \text{rat-poly.row-length} \ (\text{mat1} \ (2 \cdot 2^k))) \)
ultimately have

\[(2 \land k) \rightarrow (\mat 2 \land 1 (j \div \text{length} (\mat 1 (2 \land k))) \land (i \div \text{rat-poly.row-length} (\mat 1 (2 \land k)))) = 0\]

using \textit{s1 bymetis}
then have 
\[\neg(j \div \text{length} (\mat 1 (2 \land k)) = i \div \text{rat-poly.row-length} (\mat 1 (2 \land k))) \rightarrow \textit{thesis}\]
            by auto
then show \textit{thesis using 11 by auto}
qed
then show \textit{thesis by auto}
qed
qed
qed
qed
qed

then have
\[(\forall i. \forall j. (i < (2 \land (k+1))) \land (j < (2 \land (k+1)))) \rightarrow
((\text{rat-poly.Tensor} (\mat 1 2) (\mat 1 (2 \land k))) \land i) = (\text{if}
((j \div (2 \land k)) = (i \div (2 \land k))
\land (j \text{mod} (2 \land k)) = (i \text{mod} (2 \land k)))
then 1
else 0))\]
            by \textit{(metis (opaque-lifting, no-types) 6 7)}
then have \[20:((\forall i. \forall j. (i < (2 \land (k+1))) \land (j < (2 \land (k+1)))) \rightarrow
((\text{rat-poly.Tensor} (\mat 1 2) (\mat 1 (2 \land k))) \land i) = (\text{if} (j = i)
then 1
else 0))\]
            using \textit{equal-div-mod2 by auto}
with \[2\] have
\[(\forall i. \forall j. (i < (2 \land (k+1))) \land (j < (2 \land (k+1)))) \rightarrow
((\text{rat-poly.Tensor} (\mat 1 2) (\mat 1 (2 \land k))) \land i) = (\mat 1 (2 \land (k+1)))\]
            by \textit{metis}
then have \[(\forall i < (2 \land (k+1)). \forall j < (2 \land (k+1)),
((\text{rat-poly.Tensor} (\mat 1 2) (\mat 1 (2 \land k))) \land i) = (\mat 1 (2 \land (k+1)))\]
            by \textit{auto}
moreover have \[\mat (2 \land (k+1)) (2 \land (k+1)) (\text{rat-poly.Tensor} (\mat 1 2) (\mat 1 (2 \land k)))\]
            using \textit{make-vert-equiv \((k + 1) = \mat 1 2 \otimes \mat 1 (2 \land k))\}
            by \textit{(metis prop-make-vert-equiv\((1)\) prop-make-vert-equiv\((2)\) prop-make-vert-equiv\((3)\))}\nultimately have
\[(\text{rat-poly.Tensor} (\mat 1 2) (\mat 1 (2 \land k))) = (\mat 1 (2 \land (k + 1)))\]
using 1 mat-eqI by metis
then show ?thesis by auto
qed
then show ?case using make-vert-equiv.simps
using (make-vert-equiv (k + 1) = mat1 2 ⊗ mat1 (2 ^ k),
by (metis Suc-eq-plus1)
qed

theorem make-vert-block-map-blockmat:
  blockmat (make-vert-block n) = (mat1 (2 ^ n))
  by (metis blockmat-make-vert make-vert-equiv-mat)

lemma mat1-rt-mult: assumes mat nr nc m1
  shows rat-poly.matrix-mult m1 (mat1 (nc)) = m1
  using assms mat1-mult-right rat-poly.mat-empty-row-length
  rat-poly.matrix-row-length
  rat-poly.row-length-def rat-poly.unique-row-col(1) by (metis)

lemma mat1-vert-block:
  rat-poly.matrix-mult
    (blockmat b)
    (blockmat (make-vert-block (nat (codomain-block b))))
    = (blockmat b)
  proof –
  have mat
    (rat-poly.row-length (blockmat b))
    (2 ^ (nat (codomain-block b)))
    (blockmat b)
    using length-codomain-block matrix-blockmat
    by auto
  moreover have (blockmat (make-vert-block (nat (codomain-block b))))
    = mat1 (2 ^ (nat (codomain-block b)))
    using make-vert-block-map-blockmat by auto
  ultimately show ?thesis using mat1-rt-mult by auto
  qed

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::
  rat-poly mat ⇒ rat-poly mat ⇒ rat-poly mat ⇒ bool
where
  ∧ (mat (rat-poly.row-length A2) (length A2) A2)
  ∧ (mat (rat-poly.row-length B1) 1 B1)
  ∧ (A1 ≠ []) ∧ (A2 ≠ []) ∧ (B1 ≠ [])
  ∧ (length A1 = rat-poly.row-length A2)
**Theorem** weak-distributivity1:
weak-matrix-match $A_1 A_2 B_1$

\[ \Rightarrow \left( (\text{rat-poly matrix-mult } A_1 A_2) \otimes B_1 \right) = (\text{rat-poly matrix-mult } (A_1 \otimes B_1) (A_2)) \]

**Proof** –
assume `assms: weak-matrix-match A1 A2 B1`
have 1: `length B1 = 1`
   using `assms weak-matrix-match-def`
   by (metis `rat-poly matrix-row-length` `rat-poly unique-row-col(2)`)
have `[1] \neq []`
   by auto
moreover have `mat 1 1 [1]`
   unfolding `mat-def` `Ball-def` `vec-def` by auto
moreover have `rat-poly row-length [1] = length B1`
   unfolding `rat-poly row-length-def` 1 by auto
ultimately have `rat-poly matrix-match A1 A2 B1 [1]`
   unfolding `rat-poly matrix-match-def`
   using `assms weak-matrix-match-def` 1 `blockmat.simps(1)`
   `matrix-blockmat` by (metis (opaque-lifting, no-types))
then have \(( (\text{rat-poly matrix-mult } A_1 A_2) \otimes (\text{rat-poly matrix-mult } B_1 [1]) )) = (\text{rat-poly matrix-mult } (A_1 \otimes B_1) (A_2 \otimes [1]))\)
   using `rat-poly distributivity` by auto
moreover have `(\text{rat-poly matrix-mult } B_1 [1]) = B_1`
   using `weak-matrix-match-def` `assms mat1-equiv mat1-mult-right`
   by (metis)
moreover have `(A_2 \otimes [1]) = A_2`
   using `rat-poly Tensor-right-id` by (metis)
ultimately show `?thesis` by auto
qed

**Definition** weak-matrix-match2::
\[ \text{rat-poly mat } \Rightarrow \text{rat-poly mat } \Rightarrow \text{rat-poly mat } \Rightarrow \text{bool} \]

where
\[ \text{weak-matrix-match2 } A_1 B_1 B_2 \equiv (\text{mat } (\text{rat-poly row-length } A_1) 1 A_1)
\wedge (\text{mat } (\text{rat-poly row-length } B_1) (\text{length } B_1) B_1)
\wedge (\text{mat } (\text{rat-poly row-length } B_2) (\text{length } B_2) B_2)
\wedge (\text{length } B_1 = \text{rat-poly row-length } B_2) \]

**Theorem** weak-distributivity2:
weak-matrix-match2 $A_1 B_1 B_2$

\[ \Rightarrow (A_1 \otimes (\text{rat-poly matrix-mult } B_1 B_2)) = (\text{rat-poly matrix-mult } (A_1 \otimes B_1) (B_2)) \]

**Proof** –
assume `assms: weak-matrix-match2 A1 B1 B2`
have 1: `length A1 = 1`
   using `assms weak-matrix-match2-def`
   by (metis `rat-poly matrix-row-length` `rat-poly unique-row-col(2)`)
have `[1] \neq []`
by auto
moreover have mat 1 1
  unfolding mat-def Ball-def vec-def by auto
moreover have rat-poly.row-length [[1]] = length A1
  unfolding rat-poly.row-length-def 1 by auto
ultimately have rat-poly.matrix-match A1 [[1]] B1 B2
  unfolding rat-poly.matrix-match-def
  using assms weak-matrix-match2-def
  I blockmat.simps(1) matrix-blockmat
  by (metis (opaque-lifting, no-types))
then have ((rat-poly.matrix-mult A1 [[1]])⊗(rat-poly.matrix-mult B1 B2))
  = (rat-poly.matrix-mult (A1 ⊗ B1) ([[1]] ⊗ B2))
  using rat-poly.distributivity by auto
moreover have (rat-poly.matrix-mult A1 [[1]]) = A1
  using weak-matrix-match2-def
  assms mat1-equiv mat1-mult-right
  by (metis)
moreover have ([[1]] ⊗ B2) = B2
  by (metis rat-poly.Tensor-left-id)
ultimately show ?thesis by auto
qed

lemma is-tangle-diagram-weak-matrix-match:
assumes is-tangle-diagram (w1∗ws1)
  and codomain-block w2 = 0
shows weak-matrix-match (blockmat w1) (kauff-mat ws1) (blockmat w2)
unfolding weak-matrix-match-def
apply(auto)
proof–
  show mat
    (rat-poly.row-length (blockmat w1))
    (length (blockmat w1))
    (blockmat w1)
    using matrix-blockmat by auto
next
have is-tangle-diagram ws1
  using assms(1) is-tangle-diagram.simps(2) by metis
then show mat
  (rat-poly.row-length (kauff-mat ws1))
  (length (kauff-mat ws1))
  (kauff-mat ws1)
  using matrix-kauff-mat by metis
next
have mat
  (rat-poly.row-length (blockmat w2))
  (length (blockmat w2))
  (blockmat w2)
  using matrix-blockmat by auto
then have \( \text{mat} \left( \text{rat-poly}, \text{row-length} \left( \text{blockmat } w_2 \right) \right) 1 \left( \text{blockmat } w_2 \right) \)
using \( \text{assms}(2) \) length-codomain-block by auto
then show \( \text{mat} \left( \text{rat-poly}, \text{row-length} \left( \text{blockmat } w_2 \right) \right) \left( \text{Suc } 0 \right) \left( \text{blockmat } w_2 \right) \)
by auto
next
show \( \text{length} \left( \text{blockmat } w_1 \right) = \text{rat-poly}, \text{row-length} \left( \text{kauff-mat } w_2 \right) \)
using \( \text{is-tangle-diagram-length-rowlength} \) \( \text{assms}(1) \) by auto
next
assume 0: blockmat \( w_1 = [] \)
show False using 0
by (metis blockmat-non-empty)
next
assume 1: kauff-mat \( w_1 = [] \)
have \( \text{is-tangle-diagram } w_1 \)
using \( \text{assms}(1) \) \( \text{is-tangle-diagram.simps}(2) \) by metis
then show False using 1 kauff-mat-non-empty by auto
next
assume 0: blockmat \( w_2 = [] \)
show False using 0
by (metis blockmat-non-empty)
qed

lemma \( \text{is-tangle-diagram-weak-matrix-match2} \):
assumes \( \text{is-tangle-diagram} \left( w_2 \ast w_3 \right) \)
and \( \text{codomain-block } w_1 = 0 \)
shows \( \text{weak-matrix-match2} \left( \text{blockmat } w_1 \right) \left( \text{blockmat } w_2 \right) \left( \text{kauff-mat } w_3 \right) \)
unfolding \( \text{weak-matrix-match2-def} \)
apply (auto)
proof (~)
  have \( \text{mat} \left( \text{rat-poly}, \text{row-length} \left( \text{blockmat } w_1 \right) \right) \)
  (\( \text{length} \left( \text{blockmat } w_1 \right) \))
  (\( \text{blockmat } w_1 \))
  using matrix-blockmat by auto
then have \( \text{mat} \left( \text{rat-poly}, \text{row-length} \left( \text{blockmat } w_1 \right) \right) 1 \left( \text{blockmat } w_1 \right) \)
using \( \text{assms}(2) \) length-codomain-block by auto
then show \( \text{mat} \left( \text{rat-poly}, \text{row-length} \left( \text{blockmat } w_1 \right) \right) \left( \text{Suc } 0 \right) \left( \text{blockmat } w_1 \right) \)
by auto
next
have \( \text{is-tangle-diagram } w_2 \)
using \( \text{assms}(1) \) \( \text{is-tangle-diagram.simps}(2) \) by metis
then show \( \text{mat} \left( \text{rat-poly}, \text{row-length} \left( \text{kauff-mat } w_2 \right) \right) \)
(\( \text{length} \left( \text{kauff-mat } w_2 \right) \))
(\( \text{kauff-mat } w_2 \))
using matrix-kauff-mat by metis

81
next
show mat
  (rat-poly.row-length (blockmat w2))
  (length (blockmat w2))
  (blockmat w2)
by (metis matrix-blockmat)
next
show length (blockmat w2) = rat-poly.row-length (kauff-mat ws2)
  using is-tangle-diagram-length-rowlength assms(1) by auto
next
assume 0:blockmat w1 = []
show False using 0
  by (metis blockmat-non-empty)
next
assume 1:kauff-mat ws2 = []
have is-tangle-diagram ws2
  using assms(1) is-tangle-diagram.simps(2) by metis
then show False using 1 kauff-mat-non-empty by auto
next
assume 0:blockmat w2 = []
show False using 0
  by (metis blockmat-non-empty)
qed

lemma is-tangle-diagram-vert-block:
  is-tangle-diagram (b ∗ (basic (make-vert-block (nat (codomain-block b)))))
proof
  have domain-wall (basic (make-vert-block (nat (codomain-block b))))
    = (codomain-block b)
    using domain-wall.simps make-vert-block.simps
    by (metis codomain-block-nonnegative domain-make-vert int-nat-eq)
  then show ?thesis using is-tangle-diagram.simps by auto
qed

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

theorem Tensor-Invariance:
(is-tangle-diagram ws1) ∧ (is-tangle-diagram ws2)
⇒ (kauff-mat (ws1 ⊗ ws2) = (kauff-mat ws1) ⊗ (kauff-mat ws2))
proof(induction rule:tensor.induct)
case 1
  show ?case using kauff-mat-tensor-distrib by auto
next
fix a b as bs
assume hyps: is-tangle-diagram as ∧ is-tangle-diagram bs
  ⇒ (kauff-mat (as ⊗ bs) = kauff-mat as ⊗ kauff-mat bs)
assume prems: is-tangle-diagram (a*as) ∧ is-tangle-diagram (b*bs)
let ?case = kauff-mat (a ∗ as) ⊗ kauff-mat (b ∗ bs)
\( = \text{kauff-mat} (a \ast as) \otimes \text{kauff-mat} (b \ast bs) \)

have 0:rat-poly.matrix-match
\[
\begin{align*}
\text{(blockmat } a) \\
\text{(kauff-mat } as) \\
\text{(blockmat } b) \\
\text{(kauff-mat } bs)
\end{align*}
\]

using prems is-tangle-diagram-matrix-match by auto

have 1:is-tangle-diagram as \& is-tangle-diagram bs

using prems is-tangle-diagram.simps by metis

have kauff-mat \(((a \ast as) \otimes (b \ast bs))\)
\[
= \text{kauff-mat} (((a \otimes b) \ast (as \otimes bs)))
\]

using tensor.simps by auto

moreover have ... = rat-poly.matrix-mult
\[
\begin{align*}
\text{(blockmat } (a \otimes b)) \\
\text{(kauff-mat } (as \otimes bs))
\end{align*}
\]

using kauff-mat.simps(2) by auto

moreover have ... = rat-poly.matrix-mult
\[
\begin{align*}
(((\text{blockmat } a) \otimes (\text{blockmat } b)) \\
((\text{kauff-mat } as) \otimes (\text{kauff-mat } bs))
\end{align*}
\]

using hyps 1 kauff-mat-tensor-distrib by auto

moreover have ... = (rat-poly.matrix-mult (\text{blockmat } a) (\text{kauff-mat } as))
\[
\otimes (\text{rat-poly.matrix-mult } (\text{blockmat } b) (\text{kauff-mat } bs))
\]

using 0 rat-poly.distributivity by auto

moreover have ... = kauff-mat (a*as) \otimes kauff-mat (b*bs)

by auto

ultimately show ?case by metis

next

fix a b as bs

assume hyps:codomain-block b \(\neq 0\)

\[
\begin{align*}
\Rightarrow \text{is-tangle-diagram as} \\
& \text{\& is-tangle-diagram} \\
& (\text{(basic } (\text{make-vert-block } (\text{nat } (\text{codomain-block } b)))))
\Rightarrow \text{kauff-mat}
& (\text{(as } \otimes \text{ basic } (\text{make-vert-block } (\text{nat } (\text{codomain-block } b))))
= \text{kauff-mat as} \\
& \otimes \text{kauff-mat}
& (\text{(basic } (\text{make-vert-block } (\text{nat } (\text{codomain-block } b)))))
\end{align*}
\]

assume prems:is-tangle-diagram (a * as) \& is-tangle-diagram (basic b)

let ?case = \text{kauff-mat} (a \ast as \otimes \text{basic } b)
\[
= \text{kauff-mat} (a \ast as) \otimes \text{kauff-mat} (\text{basic } b)
\]

show ?case

proof(cases codomain-block b = 0)

case True

have \(((a \ast as) \otimes (\text{basic } b)) = ((a \otimes b) \ast as)\)

using tensor.simps True by auto

then have kauff-mat \(((a \ast as) \otimes (\text{basic } b))\)
\[
= \text{kauff-mat} (((a \otimes b) \ast as))
\]
by auto
moreover have ... =
  rat-poly.matrix-mult
  (blockmat (a ⊗ b))
  (kauff-mat as)
by auto
moreover have ... =
  rat-poly.matrix-mult
  ((blockmat a) ⊗ (blockmat b))
  (kauff-mat as)
using blockmat-tensor-distrib by (metis)
ultimately have $T_1$:
  kauff-mat ((a * as) ⊗ (basic b))
= rat-poly.matrix-mult
  ((blockmat a) ⊗ (blockmat b))
  (kauff-mat as)
by auto
then have weak-matrix-match
  (blockmat a)
  (kauff-mat as)
  (blockmat b)
using is-tangle-diagram-weak-matrix-match True prems by auto
then have rat-poly.matrix-mult
  ((blockmat a) ⊗ (blockmat b))
  (kauff-mat as)
  = ((rat-poly.matrix-mult
       (blockmat a)
       (kauff-mat as))
       ⊗ (blockmat b))
using weak-distributivity1 by auto
moreover have ... = (kauff-mat (a*as)) ⊗ (kauff-mat (basic b))
by auto
ultimately show thesis using $T_1$ by metis
next
case False
let $?bs = (basic (make-vert-block (nat (codomain-block b))))$
have $F_0$: rat-poly.matrix-match
  (blockmat a)
  (kauff-mat as)
  (blockmat b)
  (kauff-mat ?bs)
using prems is-tangle-diagram-vert-block
is-tangle-diagram-matrix-match by metis
have $F_1$: codomain-block b ≠ 0
using False by auto
have $F_2$: is-tangle-diagram as
  ∧ is-tangle-diagram $?bs$
using is-tangle-diagram.simps prems by metis
then have $F_3$: kauff-mat
\[(\text{as} \otimes \text{basic})(\text{make-vert-block}(\text{nat}(\text{codomain-block} b)))\] = 
\[\text{kauff-mat as} \otimes \text{kauff-mat ?bs}\]

using \(F1\) \text{hyps} by \text{auto}

moreover have \(((\text{a*as}) \otimes (\text{basic} b)) = (\text{a } \otimes \text{b}) \ast (\text{as} \otimes \text{?bs})\)

using \(\text{False tensor.simps}\) by \text{auto}

moreover then have \(
\text{kauff-mat}((\text{a*as}) \otimes (\text{basic} b)) = \text{kauff-mat}((\text{a } \otimes \text{b}) \ast (\text{as} \otimes \text{?bs}))\)

by \text{auto}

moreover then have \(\ldots = \text{rat-poly.matrix-mult}\)
\[(\text{blockmat}(\text{a} \otimes \text{b}))\]
\[(\text{kauff-mat}(\text{as} \otimes \text{?bs}))\]

using \(\text{kauff-mat.simps}\) by \text{auto}

moreover then have \(\ldots = \text{rat-poly.matrix-mult}\)
\[((\text{blockmat a}) \otimes (\text{blockmat b}))\]
\[((\text{kauff-mat as}) \otimes (\text{kauff-mat ?bs}))\]

using \(F3\) \text{blockmat-tensor-distrib} by \text{(metis)}

moreover then have \(\ldots = (\text{rat-poly.matrix-mult}(\text{blockmat a}) \otimes (\text{kauff-mat as}))\)

using \(\text{rat-poly.distributivity} F0\) by \text{auto}

moreover then have \(\ldots = (\text{rat-poly.matrix-mult}(\text{blockmat a}) \otimes (\text{kauff-mat as}))\)
\[(\text{blockmat b})\]

using \(\text{mat1-vert-block}\) by \text{auto}

moreover then have \(\ldots = (\text{kauff-mat}(\text{a*as})) \otimes (\text{kauff-mat}(\text{basic} b))\)

using \(\text{kauff-mat.simps}\) by \text{auto}

ultimately show \(?\text{thesis}\) by \text{metis}

\text{qed}

next

fix \(a \ b as bs\)

assume \text{hyps}:
\[\text{codomain-block} b \neq 0\]

\[\implies \text{is-tangle-diagram}\]
\[\text{basic}(\text{make-vert-block}(\text{nat}(\text{codomain-block} b)))\]
\[\land (\text{is-tangle-diagram} \text{as})\]

\[\implies \text{kauff-mat}(\text{basic}(\text{make-vert-block}(\text{nat}(\text{codomain-block} b))) \otimes \text{as})\]
\[\otimes \text{kauff-mat} \text{as}\]

assume \text{prems}: \text{is-tangle-diagram} (\text{basic} b) \land \text{is-tangle-diagram} (a * \text{as})

let \(?\text{case} = \text{kauff-mat}(\text{ (basic} b) \otimes (\text{a * as}))\)
\[\implies \text{kauff-mat}(\text{ (basic} b) \otimes \text{kauff-mat} (a * \text{as}))\]

show \(?\text{case}\)

proof\((\text{cases codomain-block} b = 0)\)

case \text{True}

have \(((\text{basic} b) \otimes (\text{a * as})) = ((b \otimes a) * \text{as})\)
using tensor.simps True by auto
then have kauff-mat ((basic b) ⊗ (a * as)) = kauff-mat ((b ⊗ a) * as) by auto
moreover have ... = rat-poly.matrix-mult
  (blockmat (b ⊗ a)) (kauff-mat as) by auto
moreover have ... = rat-poly.matrix-mult
  ((blockmat b) ⊗ (blockmat a)) (kauff-mat as) using blockmat-tensor-distrib by (metis)
ultimately have T1:kauff-mat ((basic b) ⊗ (a * as)) = rat-poly.matrix-mult
  ((blockmat b) ⊗ (blockmat a)) (kauff-mat as) by auto
then have weak-matrix-match2
  (blockmat b) (blockmat a) (kauff-mat as) using is-tangle-diagram-weak-matrix-match2 True prems by auto
have F0:rat-poly.matrix-match
  ((blockmat b) ⊗ (blockmat a)) (kauff-mat as) = (blockmat b)
  ⊗ (rat-poly.matrix-mult (blockmat a)(kauff-mat as)) using weak-distributivity2 by auto
moreover have ... = (kauff-mat (basic b)) ⊗ (kauff-mat (a * as)) by auto
ultimately show ?thesis using T1 by metis
next case False
let ?bs = (basic (make-vert-block (nat (codomain-block b))))
have F0:rat-poly.matrix-match
  (blockmat b) (kauff-mat ?bs) (blockmat a) (kauff-mat as) using prems is-tangle-diagram-weak-matrix-match
  is-tangle-diagram-matrix-match by metis
have F1:codomain-block b ≠ 0 using False by auto
have F2: is-tangle-diagram as ∧ is-tangle-diagram ?bs using is-tangle-diagram.simps prems by metis
then have F3:kauff-mat (?bs ⊗ as) = kauff-mat ?bs ⊗ kauff-mat as using F1 hyps by auto
moreover have \((\text{basic}\ b \otimes (a\ast a)) = (b \otimes a) \ast (\mathcal{P} \otimes \mathcal{S})\)
using False tensor.simps by auto
moreover then have
\[\text{kauff-mat}\ ((\text{basic}\ b \otimes (a\ast a)) = \text{kauff-mat}((b \otimes a) \ast (\mathcal{P} \otimes \mathcal{S}))\]
by auto
moreover then have ...
\[= \text{rat-poly.matrix-mult}\]
\[(\text{blockmat}\ (b \otimes a))\]
\[(\text{kauff-mat}\ (\mathcal{P} \otimes \mathcal{S}))\]
using kauff-mat.simps by auto
moreover then have ...
\[= \text{rat-poly.matrix-mult}\]
\[((\text{blockmat}\ b) \otimes (\text{blockmat}\ a))\]
\[((\text{kauff-mat}\ \mathcal{P}) \otimes (\text{kauff-mat}\ a))\]
using F3 by (metis blockmat-tensor-distrib)
moreover then have ...
\[= (\text{rat-poly.matrix-mult}\]
\[(\text{blockmat}\ b)\]
\[(\text{kauff-mat}\ \mathcal{P})\]
\[\otimes (\text{rat-poly.matrix-mult}\]
\[(\text{blockmat}\ a)\]
\[(\text{kauff-mat}\ a)\]
using rat-poly.distributivity F0 by auto
moreover then have ...
\[= (\text{blockmat}\ b)\]
\[\otimes (\text{rat-poly.matrix-mult}\]
\[(\text{blockmat}\ a)\]
\[(\text{kauff-mat}\ a)\]
using mat1-vert-block by auto
moreover then have ...
\[= (\text{kauff-mat}\ \text{basic}\ b)\]
\[\otimes (\text{kauff-mat}\ (a\ast a))\]
using kauff-mat.simps by auto
ultimately show \(\text{thesis}\) by metis

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))

end
(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)

proof –
have (rat-poly-times (rat-poly-times A A) (rat-poly-times A A)) = A^4
  by (simp add: numeral-Bit0)
moreover have (rat-poly-times (rat-poly-times B B) (rat-poly-times B B))
  = B^4
  by (simp add: numeral-Bit0)
moreover have (rat-poly-times 2 (rat-poly-times A (rat-poly-times A (rat-poly-times
B B))))
  = 2
  using inverse1 by (metis mult-2-right one-add-one rat-poly.assoc rat-poly.comm)
ultimately show ?thesis by auto
qed

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)
proof –
  have 1:(A − rat-poly-times (rat-poly-times B B) B)
    = A − (B^3)
    by (metis power3-eq-cube)
  then have 2:(rat-poly-times (A − rat-poly-times (rat-poly-times B B) B) B)
    = A*B − (B^3)*B
    by (metis minus-right-distributivity)
  then have ... = 1 − (B^4)
    by (simp add: inverse1 numeral-Bit0 power3-eq-cube)
  then have (rat-poly-times B (rat-poly-times (A − rat-poly-times (rat-poly-times
B B) B) B))
    = B − (B^4)*B
    using 2
    by (metis minus-right-distributivity mult.commute mult.right-neutral)
  then have 3:(rat-poly-times B (rat-poly-times (A − rat-poly-times (rat-poly-times
B B) B) B))
    = B − (B^5)
    by (metis (no-types, lifting) inverse1 minus-right-distributivity
mult.left-commute mult.right-neutral power2-eq-square power-numeral-odd)
  have (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)))
    = (A − (B^3))*(A*( A − (B^3)))
    using 1 by auto
moreover then have ...
    = (A − (B^3))*(A*A − (A*(B^3)))
    by (metis minus-left-distributivity)
moreover then have \( \ldots = (A - (B^3)) \ast (A \ast A - (B^2)) \)

using \textit{inverse1}

by (simp add: power2-eq-square power3-eq-cube)

moreover then have \( \ldots = A \ast (A \ast A - (B^2)) - (B^3) \ast (A \ast A - (B^2)) \)

by (metis minus-right-distributivity)

moreover then have \( \ldots = ((A^3) - B) - B + (B^5) \)

proof


case have: \( A \ast (A \ast A - (B^2)) = (A \ast A \ast A - A \ast (B^2)) \)

by (simp add: right-diff-distrib)

moreover have \( \ldots = (A \ast A \ast A - A \ast (B^2)) \)

by (metis power2-eq-square)

moreover have \( \ldots = ((A^3) - ((A::rat-poly) \ast B) \ast B) \)

by (simp add: power3-eq-cube)

moreover have \( \ldots = ((A^3) - ((1::rat-poly) \ast B)) \)

by (metis inverse1)

moreover have \( \ldots = (A^3) - B \)

by auto

ultimately have \( s1::(A::rat-poly) \ast (A \ast A - (B^2)) = (A^3) - (B::rat-poly) \)

by metis

have \( s2::((B::rat-poly) \ast (A^3) \ast (A \ast A - (B^2)) = (B^3) \ast (A \ast A) - (B^3) \ast (nat)) \ast (B^2) \)

by (metis minus-left-distributivity power3-eq-cube)

moreover then have \( \ldots = (((B::rat-poly) \ast (A \ast A)) - (B^5)) \)

using power-add

proof

have \( (B^3) \ast (B^2) = (B^5) \)

by (metis One-nat-def Suc-1 numeral-3-eq-3 power-Suc power-numeral-odd)

then show \( \text{thesis} \) using \( s2 \) by auto

qed

moreover then have \( \ldots = (((B::rat-poly) \ast B \ast B) \ast (A \ast A)) - (B^5) \)

by (metis power3-eq-cube)

moreover then have \( \ldots = (((B::rat-poly) \ast (B \ast (B \ast A) \ast A))))) - (B^5) \)

by auto

moreover then have \( \ldots = (((B::rat-poly) \ast (B \ast (1 \ast A))) - (B^5)) \)

using \textit{inverse2} by auto

moreover then have \( \ldots = (((B::rat-poly) \ast (B \ast A))) - (B^5) \)

by auto

moreover then have \( \ldots = (((B::rat-poly))) - (B^5) \)

using \textit{inverse2}

by simp

ultimately have \( ((B::rat-poly) \ast (A \ast A - (B^2)) = (B::rat-poly) - (B^5)) \)

by auto

then have \( A \ast (A \ast A - (B^2)) - (B^3) \ast (A \ast A - (B^2)) \)

\[= (A^3) - (B::rat-poly) - ((B::rat-poly) - (B^5))\]

using \( s1 \) by auto

then show \( \text{thesis} \) by auto

qed

ultimately have \( (rat-poly-times (A - rat-poly-times (rat-poly-times B B)) B) \)
\[(\text{rat-poly-times } A (A - \text{rat-poly-times} (\text{rat-poly-times} B B) B)))\]
\[= ((A^3) - B) - B + (B^5)\]
by auto
then have \( \exists l = B - (B^5) + ((A^3) - B) - B + (B^5)\)
using \(3\) by auto
then have \(4\): \(\exists l = (A^3) - B\)
by auto
have \(\exists r = A*((A - \text{rat-poly-times} (\text{rat-poly-times} B B) B)*A)\)
by auto
moreover then have \(\ldots = A*(A - (B^3)) \ast A\)
using \(I\) by auto
moreover have \(\ldots = A*(A \ast A - (B^3)*A)\)
by (simp add: minus-left-distributivity mult.commute)
moreover have \(\ldots = A*(A \ast A - (B*B*B)*A)\)
by (metis power3-eq-cube)
moreover have \(\ldots = A*(A \ast A - (B*B*(B*A)))\)
by auto
moreover have \(\ldots = A*(A \ast A - B*B)\)
using inverse2 minus-left-distributivity by auto
moreover have \(\ldots = A* A \ast A - A*(B*B)\)
by (metis minus-left-distributivity rat-poly.comm)
moreover have \(\ldots = A^3 - (A*B)*B\)
by (metis ab-semigroup-mult-class.mult-ac(1) power3-eq-cube)
moreover have \(\ldots = A^3 - B\)
using inverse1 by (metis monoid-mult-class.mult.left-neutral)
ultimately have \(\exists r = A^3 - B\)
by auto
then show \(\exists \text{thesis using } 4\) by auto
qed

lemma computation2:
\(\text{rat-poly-plus} (\text{rat-poly-times } A (\text{rat-poly-times} (B - \text{rat-poly-times} (\text{rat-poly-times} A A) A) A) A)\)
\[= (\text{rat-poly-times} (B - \text{rat-poly-times} (\text{rat-poly-times} A A) A) A)\]
\[= (\text{rat-poly-times} B (B - \text{rat-poly-times} (\text{rat-poly-times} A A) A))\]
\[= \text{rat-poly-times} B (B - \text{rat-poly-times} (\text{rat-poly-times} A A) A)\]
(is \(\exists l = ?r\))

proof
have \(1\): \((B - \text{rat-poly-times} (\text{rat-poly-times} A A) A)\)
\[= B - (A^3)\]
by (metis power3-eq-cube)
then have \(2\): \((\text{rat-poly-times} (B - \text{rat-poly-times} (\text{rat-poly-times} A A) A) A)\)
\[= B \ast A - (A^3)*A\]
by (metis minus-right-distributivity)
then have \(\ldots = 1 - (A^4)\)
using inverse2
by (metis mult.commute one-plus-numeral power-add power-one-right
semiring-norm(2)
semiring-norm(4))
then have \((\text{rat-poly-times } A \text{ (rat-poly-times } (B - \text{rat-poly-times } (\text{rat-poly-times } A \cdot A) \cdot A))\)
\[= A - (A^4)\cdot A\]
using 2
by (simp add: minus-left-distributivity)
then have \(3:\text{(rat-poly-times } A \text{ (rat-poly-times } (B - \text{rat-poly-times } (\text{rat-poly-times } A \cdot A) \cdot A))\)
\[= A - (A^5)\]
by (simp add: numeral-\text{Bit0 numeral-\text{Bit1})
have \((\text{rat-poly-times } (B - \text{rat-poly-times } (\text{rat-poly-times } A \cdot A)) \cdot A)\)
\[= (B - (A^3))\cdot (B\cdot (B - (A^3)))\]
using 1 by auto
moreover then have \(\ldots = (B - (A^3))\cdot (B\cdot (B - (A^2)))\)
by (metis minus-left-distributivity)
moreover then have \(\ldots = (B - (A^3))\cdot (B\cdot (B - (A^2)))\)
using inverse2
by (simp add: power2-\text{eq-square power3-\text{eq-cube})
moreover then have \(\ldots = (B\cdot (B\cdot (B - (A^2))) - (A^3))\cdot (B\cdot (B - (A^2)))\)
by (metis minus-right-distributivity)
moreover then have \(\ldots = (((B^3) - A) - A + (A^5)\)
proof-
have \(B\cdot (B\cdot (B - (A^2))) = (B\cdot B\cdot B - B\cdot (A^2))\)
by (simp add: right-diff-distrib)
moreover have \(\ldots = (B\cdot B\cdot B - B\cdot (A\cdot A))\)
by (metis power2-\text{eq-square)
moreover have \(\ldots = ((B^3) - ((B::\text{rat-poly})\cdot A)\cdot A)\)
by (simp add: power3-\text{eq-cube)
moreover have \(\ldots = ((B^3) - ((1::\text{rat-poly})\cdot A))\)
by (metis inverse2)
moreover have \(\ldots = (B^3) - A\)
by auto
ultimately have \(s1::(B::\text{rat-poly})\cdot (B\cdot B - (A^2)) = (B^3) - (A::\text{rat-poly})\)
by metis
have \(s2::((A::\text{rat-poly})\cdot A\cdot A\cdot B - (A^2)) = (A^3)\cdot (B\cdot B) - (A^\text{3::nat})\cdot A\cdot (A^2)\)
by (metis minus-left-distributivity power3-\text{eq-cube)
moreover then have \(\ldots = (((A::\text{rat-poly})\cdot A\cdot A\cdot B - (A^2)) - (A^5)\)
using power-add
proof-
have \((A^\text{3::nat})\cdot (A^2) = A^5\)
by (metis One-nat-def Suc-1 numeral-3-\text{eq-3 power-Suc power-numeral-odd)
then show \(?thesis\) using \(s2\) by auto
qed
moreover then have \(\ldots = (((((A::\text{rat-poly})\cdot (A\cdot B))\cdot B)) - (A^5)\)
by auto
moreover then have \(\ldots = (((((A::\text{rat-poly})\cdot (A\cdot (1\cdot B))\cdot B)) - (A^5)\)
moreover then have \(\ldots = (((((A::\text{rat-poly})\cdot (A\cdot (1\cdot B))\cdot B)) - (A^5)\)

91
using inverse1 by auto
moreover then have ... = (((((A::rat-poly)*B)) - (A^5))
by auto
moreover then have ... = (((((A::rat-poly))) - (A^5))
using inverse1 by auto
ultimately have (A::rat-poly)^3*(B*B - (A^2)) = ((A::rat-poly) - (A^5))
by auto
then have B*(B*B - (A^2)) - (A^3)*(B*B - (A^2))
= (B^3) - (A::rat-poly) - ((A::rat-poly) - (A^5))
using s1 by auto
then show thesis by auto
qed
ultimately have (rat-poly-times (B - rat-poly-times (rat-poly-times A A) A) A)
(rat-poly-times B (B - rat-poly-times (rat-poly-times A A) A))
= ((B^3) - A) - A + (A^5)
by auto
then have l = A - (A^5) + ((B^3) - A) - A + (A^5)
using 3 by auto
then have 4: l = (B^3) - A
by auto
have r = B*((B - rat-poly-times (rat-poly-times A A) A)*B)
by auto
moreover then have ... = B*(B - (A^3))*B
using 1 by auto
moreover have ... = B*(B*B - (A^3)*B)
using minus-left-distributivity by (simp add: minus-left-distributivity
mult.commute)
moreover have ... = B*(B*B - (A*A*A)*B)
by (metis power3-eq-cube)
moreover have ... = B*(B*B - (A*A*(A*B)))
by auto
moreover have ... = B*(B*B - A*A)
using inverse1 by auto
moreover have ... = B*B*B - B*(A*A)
by (metis minus-left-distributivity rat-poly.comm)
moreover have ... = B^3 - (B*A)*A
by (metis ab-semigroup-mult-class.mult-ac(1) power3-eq-cube)
moreover have ... = B^3 - A
using inverse2 by (metis monoid-mult-class.mult.left-neutral)
ultimately have r = B^3 - A
by auto
then show thesis using 4 by auto
qed

lemma computation-swingneg: rat-poly-times B (rat-poly-times (B - rat-poly-times
(rat-poly-times A A) A) B) =
rat-poly-plus
(rat-poly-times (B - rat-poly-times (rat-poly-times A A) A)
(rat-poly-times B (B - rat-poly-times (rat-poly-times A A) A)))
\[(\text{rat-poly-times} \ A \ \text{rat-poly-times} \ (B - \text{rat-poly-times} \ \text{rat-poly-times} \ A \ A) \ A)\]

using computation2 by auto

lemma computation-tappos: \text{rat-poly-inv} \ (\text{rat-poly-times} \ (A - \text{rat-poly-times} \ \text{rat-poly-times} \ B \ B) \ A) = \\
\text{rat-poly-times} \ (B - \text{rat-poly-times} \ \text{rat-poly-times} \ A \ A) \ B\text{is } ?l = ?r

proof -

have \text{rat-poly-times} \ (A - \text{rat-poly-times} \ \text{rat-poly-times} \ B \ B) \ A) = \\
\text{rat-poly-times} \ ((A - ((B+B)*B))*A)
by auto

moreover then have ... = \text{rat-poly-inv} \ ((A*A) - (B*B))
by (metis minus-left-distributivity rat-poly.comm)

ultimately have \text{rat-poly-inv} \ ((A*A) - (B*B))
by auto

then have \text{thesis using } 1 by auto

qed

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) \ A\text{is } ?l = ?r

proof -

have \text{rat-poly-times} \ (B - \text{rat-poly-times} \ \text{rat-poly-times} \ A \ A) \ A) \ B) = \\
((B - ((A*A)*A))*B)
by auto

moreover then have ... = \text{rat-poly-times} \ (B*B) - (A*A)*A B
by (metis minus-left-distributivity rat-poly.comm)

ultimately have \text{thesis using } 1 by auto

qed
ultimately have \( \hat{?l} = \text{rat-poly-inv} ((B\ast B) - (A\ast A)) \)
   by auto
then have \( 1 : \hat{?l} = (A\ast A) - (B\ast B) \)
   by auto
have \( \hat{?r} = (A - ((B\ast B) \ast B)) \ast A \)
   by auto
moreover have ... = \( A\ast A - ((B\ast B) \ast B) \ast A \)
   by (metis minus-left-distributivity rat-poly.comm)
moreover have ... = \( (A\ast A) - ((B\ast B) \ast (B\ast A)) \)
   by auto
moreover have ... = \( ((A :: \text{rat-poly}) \ast A) - (B\ast B) \)
   using inverse2 by auto
ultimately have \( \hat{?r} = (A\ast A) - (B\ast B) \)
   by auto
then show \( \text{thesis} \) using 1 by auto
qed

lemma computation-downpos: rat-poly-times A (A - rat-poly-times (rat-poly-times B B) B) =
   rat-poly-inv (rat-poly-times B (B - rat-poly-times (rat-poly-times A A) A))
   using computation-downpos-prelim by (metis rat-poly.comm)

lemma computation-positive-flip: rat-poly-plus
   (rat-poly-inv (rat-poly-times A (rat-poly-times (A - rat-poly-times (rat-poly-times B B) B) A)))
   (rat-poly-inv (rat-poly-times B (rat-poly-times A B))) =
   rat-poly-inv (rat-poly-times A (rat-poly-times A A)) (is \( ?l = ?r \))
proof
  have \( \text{rat-poly-inv} (\text{rat-poly-times} B (\text{rat-poly-times} A B))\)
   = \( \text{rat-poly-inv} (\text{rat-poly-times} B 1) \)
   using inverse1 by auto
moreover have ... = - B
   by auto
ultimately have \( 1 : (\text{rat-poly-inv} (\text{rat-poly-times} B (\text{rat-poly-times} A B))) = - B \)
   by auto
have \( \text{rat-poly-times} A (\text{rat-poly-times} (A - \text{rat-poly-times} (\text{rat-poly-times} B B) B) A))\)
   = \( A \ast ((A - ((B\ast B) \ast B)) \ast A) \)
   by auto
moreover then have ... = \( A \ast ((A\ast A) - ((B\ast B) \ast B) \ast A) \)
   by (metis minus-left-distributivity rat-poly.comm)
moreover then have ... = \( A \ast ((A\ast A) - ((B\ast B) \ast 1)) \)
   using inverse2 by auto
moreover then have ... = \( A \ast ((A\ast A) - (B\ast B)) \)
   by auto
moreover then have ... = \( A \ast ((A\ast A) - (A \ast (B\ast B))) \)
   by (metis minus-left-distributivity)
moreover then have ... = \( A \ast (A\ast A) - (1 \ast B) \)
using inverse1 by auto
moreover then have ... = (A*(A*A)) - B 
by auto
ultimately have (rat-poly-times A (rat-poly-times (A - rat-poly-times (rat-poly-times B B) B) A))

= (A*(A*A)) - B 
by auto
then have rat-poly-inv (rat-poly-times A (rat-poly-times (A - rat-poly-times (rat-poly-times B B) B) A))

= B - (A*A*A) 
by auto
then have 3: ?l = - (A*A*A) 
using I by auto
moreover have ?r = - (A*A*A) 
by auto
ultimately show ?thesis by auto
qed

lemma computation-negative-flip-rat-poly-plus

(rat-poly-times B (rat-poly-times B (rat-poly-times (B - rat-poly-times (rat-poly-times
A A) A) B))))

= (rat-poly-inv (rat-poly-times A (rat-poly-times B A)))

(rat-poly-inv (rat-poly-times A (rat-poly-times B A))) =

(rat-poly-inv (rat-poly-times B (rat-poly-times B B)) (is ?l = ?r)

proof--

have (rat-poly-inv (rat-poly-times A (rat-poly-times B A)))

= (rat-poly-inv (rat-poly-times A I))

using inverse2 by auto
moreover have ... = - A 
by auto
ultimately have 1:(rat-poly-inv (rat-poly-times A (rat-poly-times B A))) = - A 
by auto
have (rat-poly-times B (rat-poly-times (B - rat-poly-times (rat-poly-times A A) A) B)))

= B*(((B - ((A*A)*A))*B)

by auto
moreover then have ... = B*((B*B) - ((A*A)*A*B))

by (metis minus-left-distributivity rat-poly.comm)
moreover then have ... = B*((B*B) - ((A*A)*I))

using inverse1 by auto
moreover then have ... = B*((B*B) - (A*A))

by auto
moreover then have ... = B*(B*B) - (B*(A*A))

by (metis minus-left-distributivity)
moreover then have ... = (B*(B*B)) - (1*A)

using inverse2 by auto
moreover then have ... = (B*(B*B)) - A 
by auto
ultimately have (rat-poly-times B (rat-poly-times (B - rat-poly-times (rat-poly-times
A A) A) B)) = (B*(B*B)) − A

by auto

then have rat-poly-inv (rat-poly-times B (rat-poly-times (B − rat-poly-times (rat-poly-times A A) A) B))

= A − (B*B*B)

by auto

then have 3:?l = − (B*B*B)

using 1 by auto

moreover have ?r = − (B*B*B)

by auto

ultimately show ?thesis by auto

qed

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 –

have rat-poly-times (rat-poly-times A A) A

= ((A*A)*A)

by auto

then have rat-poly-times B (B − rat-poly-times (rat-poly-times A A) A)

= B*B − B*(A*(A*A))

using minus-left-distributivity by auto

moreover have ... = B*B − (B*(A*(A*A)))

by auto

moreover have ... = B*B − ((B*A)*(A*A))

by auto

moreover have ... = B*B − A*A

using inverse2 by auto

ultimately have 1:rat-poly-times B (B − rat-poly-times (rat-poly-times A A) A)

= B*B − A*A

by auto

have rat-poly-times (rat-poly-times B B) B = (B*B)*B

by auto

then have (rat-poly-times (A − rat-poly-times (rat-poly-times B B) B) A)

= (A*A) − ((B*B)*B)*A

using minus-right-distributivity by auto

moreover have ... = (A*A) − ((B*B)*(B*A))

by auto

moreover have ... = (A*A) − (B*B)

using inverse2 by auto

ultimately have 2:(rat-poly-times (A − rat-poly-times (rat-poly-times B B) B) A)

= (A*A) − (B*B)

by auto

have B*B − A*A + (A*A) − (B*B) = 0
by auto
with 1 2 show ?thesis by auto
qed

lemma aux1: \((A - \text{rat-poly-times} (\text{rat-poly-times} B B)) B\)
\(= A - (B^3)\)
using power3-eq-cube by (metis)

lemma square-subtract::\(((p::\text{rat-poly}) - (q::\text{rat-poly}))^2)\)
\(= (p^2) - (2*p*q) + (q^2)\)

proof -
  have 1: \(((p::\text{rat-poly}) - (q::\text{rat-poly}))^2)\)
  \(= (p - q) * (p - q)\)
  by (metis power2-eq-square)
  then have \((p - q) * (p - q)\) \(= (p - q) * p - (p - q) * q\)
  by (metis minus-right-distributivity rat-poly.comm)
  moreover have \((p - q) * p\) \(= p * p - q * p\)
  by (metis minus-left-distributivity rat-poly.comm)
  moreover have \((p - q) * q\) \(= p * q - q * q\)
  by (metis minus-left-distributivity rat-poly.comm)
  ultimately have \((p - q) * (p - q)\) \(= p * p - q * p - (p * q - q * q)\)
  by auto
  moreover have \(\ldots \) \(= (p * p) - q * p - p * q + q * q\)
  by auto
  moreover have \(\ldots \) \(= (p^2) - p * q - p * q + (q^2)\)
  using power2-eq-square by (simp add: power2-eq-square)

ultimately show ?thesis using 1 by auto
qed

lemma cube-minus::\? p q (((p::\text{rat-poly}) - (q::\text{rat-poly}))^3)\)
\(= (p^3) - 3*(p^2)*q + 3*(p)*(q^2) - (q^3)\)
apply(rule allI)
apply(rule allI)
proof -
  fix p q
  have 1: \(((p::\text{rat-poly}) - (q::\text{rat-poly}))^3\)
  \(= (p - q) * (p - q)^2\)
  by (metis One-nat-def Suc-1 numeral-3-eq-3 power-Suc)
  then have \((p - q)^2 = (p^2) - (2 * p * q) + (q^2)\)
  using square-subtract by auto
  then have 2: \((p - q) * (p - q)^2\)
  \(= (p - q) * ((p^2) - (2 * p * q) + (q^2))\)
  by auto
  moreover have 3: \((p - q) * (p - q)^2\)
  \(= p * ((p^2) - (2 * p * q) + (q^2))\)
  \(- (q * ((p^2) - (2 * p * q) + (q^2)))\)
  by (metis minus-right-distributivity)
  moreover have \(p * ((p^2) - (2 * p * q) + (q^2))\)
  \(= p * (p^2) - p * (2 * p * q) + (p * (q^2))\)
  using minus-left-distributivity by (simp add: distrib-left)

97
moreover have $p \cdot (p^2) = p^3$
by (metis One-nat-def Suc-1 numeral-3-eq-3 power-Suc)

moreover have $p \cdot (2 \cdot p \cdot q) = 2 \cdot (p^2) \cdot q$
by (metis (no-types, lifting) distrib-left mult-2 power2-eq-square semigroup-mult-class.mult_assoc)

ultimately have $4 : p \cdot ((p^2) - (2 \cdot p \cdot q) + (q^2))$
= $(p^3) - (2 \cdot (p^2) \cdot q) + (p \cdot (q^2))$
by auto

have $q \cdot ((p^2) - (2 \cdot p \cdot q) + (q^2))$
= $q \cdot (p^2) - q \cdot (2 \cdot p \cdot q) + (q \cdot (q^2))$
by (simp add: distrib-left minus-left-distributivity)

moreover have $q \cdot (p^2) = (p^2) \cdot q$
by simp

moreover have $q \cdot (2 \cdot p \cdot q) = 2 \cdot p \cdot (q^2)$
by (simp add: power2-eq-square)

ultimately have $5 : q \cdot ((p^2) - (2 \cdot p \cdot q) + (q^2))$
= $(p^2) \cdot q - 2 \cdot p \cdot (q^2) + (q^3)$
by (metis One-nat-def Suc-1 numeral-3-eq-3 power-Suc)

with 1 2 3 4 have $(p - q)^3$
= $(p^3) - (2 \cdot (p^2) \cdot q) + (p \cdot (q^2))$
- $(p \cdot (p^2) - 2 \cdot p \cdot (q^2) + (q^3))$
by auto

moreover have ... = $(p^3) - 3 \cdot (p^2) \cdot (q) + 3 \cdot (p) \cdot (q^2) - (q^3)$
by auto

ultimately show $(p - q)^3$
= rat-poly-plus $(p^3)$
rat-poly-times
(rat-poly-times 3 $(p^2)$) $q$
(rat-poly-times (rat-poly-times 3 $p$) $(q^2)$)
- $(p \cdot (p^2) - 2 \cdot p \cdot (q^2) + (q^3))$
by auto

qed

lemma power-mult: $(p :: \text{rat-poly}) \cdot m ^ n = (p ^ m) \cdot (n :: \text{nat})$
by (metis power-mult)

lemma cube-minus2:
fixes p q
shows $((p :: \text{rat-poly}) ^ 3) - (q :: \text{rat-poly}) ^ 3$
= $(p^3) - 3 \cdot (p^2) \cdot (q) + 3 \cdot (p) \cdot (q^2) - (q^3)$
using cube-minus by auto

lemma subst-poly: assumes $a = b$ shows $p \cdot \text{rat-poly} \cdot a = p \cdot b$
using assms by auto
lemma sub1:
  assumes p*q = 1
  shows r*(p*q) = r*1
  using assms by metis

lemma n-distrib: \( (A^\cdot n::nat)\cdot(B^\cdot n) = (A\cdot B)^\cdot n \)
  by (induct n)(auto)

lemma rat-poly-id-pow: \( (1::rat-poly)^\cdot n = 1 \)
  by (induct n)(auto)

lemma power-prod: \( (A^\cdot n::nat)\cdot(B^n) = (1::rat-poly)^\cdot n \)
  apply (simp add: n-distrib)
  apply (simp add: inverse1)
  done

lemma (pCons 0 1) ≠ 0
  by (metis non-zero var-def)

end

13 Tangle moves and Kauffman bracket

theory Linkrel-Kauffman
imports Computations
begin

lemma mat1-vert-wall-left:
  assumes is-tangle-diagram b
  shows rat-poly.m_matrix_mult (kauff-mat b)
  proof
    have mat \( 2^n (nat (domain-wall b))) \( (kauff-mat b)
      by (metis assms matrix-kauff-mat)
    moreover have \( blockmat (make-vert-block (nat (domain-wall b)))
      = mat1 (2^n (nat (domain-wall b)))
      using make-vert-block-map-blockmat by auto
    ultimately show \( thesis \) by (metis blockmat-make-vert mat1-mult-left prop-make-vert-equiv(1))
    qed

lemma mat1-vert-wall-right:
  assumes is-tangle-diagram b
  shows rat-poly.m_matrix_mult (kauff-mat b) 
  (blockmat (make-vert-block (nat (codomain-wall b))))
proof -

have \( \text{mat} \left( \text{rat-polynomial} \cdot \text{row-length} \left( \text{kauff-matrix} \ b \right) \right) (2^n (\text{codomain-wall} \ b)) \) (\( \text{kauff-matrix} \ b \))
  by (metis assms matrix-kauff-mat)

moreover have \( \text{block-matrix} \left( \text{make-vert-block} \ (\text{nat} \ (\text{codomain-wall} \ b)) \right) \).

ultimately show \( \text{thesis} \) by (metis mat1-rt-mult)
qed

lemma compress-top-inv: \((\text{compress-top} \ w1 \ w2) \implies \text{kauff-matrix} \ w1 = \text{kauff-matrix} \ w2\)
proof -

assume \( \text{assm}: \text{compress-top} \ w1 \ w2 \)

have \( \exists \ B, ((w1 = (\text{basic} (\text{make-vert-block} \ (\text{nat} \ (\text{domain-wall} \ B)))) \circ B) \land (w2 = (B \circ (\text{basic} ([])))) \land (\text{codomain-wall} \ B = 0) \land (is-tangle-diagram \ B)) \)
  using compress-top-def \( \text{assm} \) by auto

then obtain \( B \) where \( (w1 = (\text{basic} (\text{make-vert-block} \ (\text{nat} \ (\text{domain-wall} \ B)))) \circ B) \land (w2 = (B \circ (\text{basic} ([])))) \land (\text{codomain-wall} \ B = 0) \land (is-tangle-diagram \ B) \)
  by auto

then have \( 1: (w1 = (\text{basic} (\text{make-vert-block} \ (\text{nat} \ (\text{domain-wall} \ B)))) \circ B) \land (w2 = (B \circ (\text{basic} ([])))) \land (\text{codomain-wall} \ B = 0) \land (is-tangle-diagram \ B) \)
  by auto

then have \( \text{kauff-matrix}(w1) = (\text{kauff-matrix} ((\text{make-vert-block} \ (\text{nat} \ (\text{domain-wall} \ B)))) \circ B)) \)
  by auto

moreover then have \( ... = \text{kauff-matrix} ((\text{make-vert-block} \ (\text{nat} \ (\text{domain-wall} \ B))) \ast B) \)
  by auto

moreover then have \( ... = \text{rat-polynomial} \cdot \text{matrix-mult} \ (\text{block-matrix} \ (\text{make-vert-block} \ (\text{nat} \ (\text{domain-wall} \ B)))) \)
  \( (\text{kauff-matrix} \ B) \)
  by auto

moreover then have \( ... = (\text{kauff-matrix} \ B) \)
  using \( 1 \) \( \text{mat1-vert-wall-left} \) by (metis)

ultimately have \( \text{kauff-matrix}(w1) = \text{kauff-matrix} \ B \)
  by auto

moreover have \( \text{kauff-matrix} \ w2 = \text{kauff-matrix} \ B \)
  using \( 1 \) by (metis left-mat-compose)

ultimately show \( \text{thesis} \) by auto
qed

lemma domain-make-vert-int: \((n \geq 0) \implies (\text{domain-block} \ (\text{make-vert-block} \ (\text{nat} \ n))) = n \)
  using domain-make-vert by auto
lemma compress-bottom-inv: (compress-bottom w1 w2) ⟷ kauff-mat w1 = kauff-mat w2

proof

assume assm: compress-bottom w1 w2

have ∃ B. (\langle w1 = B \circ (basic (make-vert-block (nat (codomain-wall B)))) \rangle
and (w2 = (basic ([[]] \circ B))) and (domain-wall B = 0)
and (is-tangle-diagram B))

using compress-bottom-def assm by auto

then obtain B where \langle w1 = B \circ (basic (make-vert-block (nat (codomain-wall B)))) \rangle
and (w2 = (basic ([[]] \circ B))) and (domain-wall B = 0)
and (is-tangle-diagram B))

by auto

then have kauff-mat(w1) = (kauff-mat (B \circ (basic (make-vert-block (nat (codomain-wall B))))))

by auto

moreover then have ...
= rat-poly.matrix-mult (kauff-mat B)
= (kauff-mat (basic (make-vert-block (nat (codomain-wall B)))))

proof

have is-tangle-diagram B

using \  by auto

moreover have is-tangle-diagram (basic (make-vert-block (nat (codomain-wall B))))

using is-tangle-diagram.simps by auto

moreover have codomain-wall B = domain-wall (basic (make-vert-block (nat (codomain-wall B))))

proof

have codomain-wall B ≥ 0

apply (induct B)

by (auto) (metis codomain-block-nonnegative)

then have domain-block (make-vert-block (nat (codomain-wall B)))

= codomain-wall B

using domain-make-vert-int by auto

then show thesis unfolding domain-wall.simps(1) by auto

qed

ultimately show thesis using tangle-compose-matrix by auto

qed

moreover then have ...
= rat-poly.matrix-mult (kauff-mat B)
= (blockmat (make-vert-block (nat (codomain-wall B))))

using kauff-mat.simps(1) tangle-compose-matrix by auto

moreover then have ...
= (kauff-mat B)

using 1 mat1-vert-wall-right by auto

ultimately have kauff-mat(w1) = kauff-mat B
by auto
moreover have kauff-mat \( w_2 = \text{kauff-mat} \ B \)
using 1 by (metis right-mat-compose)
ultimately show \( \text{thesis} \) by auto
qed

**Theorem** compress-inv: \( \text{compress} \ w_1 \ w_2 \Rightarrow (\text{kauff-mat} \ w_1 = \text{kauff-mat} \ w_2) \)
**Unfolding** compress-def using compress-bottom-inv compress-top-inv by auto

**Lemma** straighten-topdown-computation: \( \text{kauff-mat} (\text{basic} ([\text{vert}, \text{cup}])) \circ (\text{basic} ([\text{cap}, \text{vert}]))) = \text{kauff-mat} (\text{basic} ([\text{vert}]) \circ (\text{basic} ([\text{vert}]))) \)
apply(simp add:kauff-mat-def)
apply(simp add:mat-multiI-def)
apply(simp add:matT-vec-multiI-def)
apply(auto simp add:replicate-def rat-poly.row-length-def)
apply(auto simp add:scalar-prod)
apply (auto simp add:inverse1 inverse2)
done

**Theorem** straighten-topdown-inv: \( \text{straighten-topdown} \ w_1 \ w_2 \Rightarrow (\text{kauff-mat} \ w_1) = (\text{kauff-mat} \ w_2) \)
**Unfolding** straighten-topdown-def using straighten-topdown-computation by auto

**Lemma** straighten-downtop-computation: \( \text{kauff-mat} (\text{basic} ([\text{cup}, \text{vert}])) \circ (\text{basic} ([\text{vert}, \text{cap}])) = \text{kauff-mat} (\text{basic} ([\text{vert}]) \circ (\text{basic} ([\text{vert}]))) \)
apply(simp add:kauff-mat-def)
apply(simp add:mat-multiI-def)
apply(simp add:matT-vec-multiI-def)
apply(auto simp add:replicate-def rat-poly.row-length-def)
apply(auto simp add:scalar-prod)
apply (auto simp add:inverse1 inverse2)
done

**Theorem** straighten-downtop-inv: \( \text{straighten-downtop} \ w_1 \ w_2 \Rightarrow (\text{kauff-mat} \ w_1) = (\text{kauff-mat} \ w_2) \)
**Unfolding** straighten-downtop-def using straighten-downtop-computation by auto

**Theorem** straighten-inv: \( \text{straighten} \ w_1 \ w_2 \Rightarrow (\text{kauff-mat} \ w_1) = (\text{kauff-mat} \ w_2) \)
**Unfolding** straighten-def using straighten-topdown-inv straighten-downtop-inv by auto

**Lemma** kauff-mat-swingpos:
\( \text{kauff-mat} (r-over\text{-braid}) = \text{kauff-mat} (l-over\text{-braid}) \)
apply (simp)
apply(simp add:mat-multI-def)
apply(simp add:matT-vec-multI-def)
apply(auto simp add:replicate-def rat-poly.row-length-def)
apply(auto simp add:scalar-prod)
apply(auto simp add:computation-swingpos)
done

lemma swing-pos-inv: swing-pos w1 w2 \implies kauff-mat w1 = kauff-mat w2
unfolding swing-pos-def using kauff-mat-swingpos by auto

lemma kauff-mat-swingneg:
kauff-mat (r-under-braid) = kauff-mat (l-under-braid)
apply(simp)
apply(simp add:mat-multI-def)
apply(simp add:matT-vec-multI-def)
apply(auto simp add:replicate-def rat-poly.row-length-def)
apply(auto simp add:scalar-prod)
apply(auto simp add:computation-swingneg)
done

lemma swing-neg-inv: swing-neg w1 w2 \implies kauff-mat w1 = kauff-mat w2
unfolding swing-neg-def using kauff-mat-swingneg by auto

theorem swing-inv:
swing w1 w2 \implies kauff-mat w1 = kauff-mat w2
unfolding swing-def using swing-pos-inv swing-neg-inv by auto

lemma rotate-topp-kauff-mat: kauff-mat ((basic [vert,over])\circ(basic [cap, vert]))
= kauff-mat ((basic [under,vert])\circ(basic [vert,cap]))
apply(simp)
apply(simp add:mat-multI-def)
apply(simp add:matT-vec-multI-def)
apply(auto simp add:replicate-def rat-poly.row-length-def)
apply(auto simp add:scalar-prod)
apply(simp add:computation-toppos)
done

lemma rotate-topp-inv: rotate-topp w1 w2 \implies kauff-mat w1 = kauff-mat w2
unfolding rotate-topp-def using rotate-topp-kauff-mat by auto

lemma rotate-topneg-kauff-mat: kauff-mat ((basic [vert,under])\circ(basic [cap, vert]))
= kauff-mat ((basic [over,vert])\circ(basic [vert,cap]))
apply(simp add:mat-multI-def)
apply(simp add:matT-vec-multI-def)
apply(auto simp add:replicate-def rat-poly.row-length-def)
apply(auto simp add:scalar-prod)
apply(simp add: computation-toppos)
done

lemma rotate-topneg-inv: rotate-topneg w1 w2 \implies (kauff-mat w1) = (kauff-mat w2)
unfolding rotate-topneg-def using rotate-topneg-kauff-mat by auto

lemma rotate-downpos-kauff-mat:
kauff-mat ((basic [cup,vert]) ◦ (basic [vert,over])) = kauff-mat ((basic [vert,cup]) ◦ (basic [under,vert]))
apply(simp add: mat-multI-def)
apply(simp add: matT-vec-multI-def)
apply(auto simp add: replicate-def rat-poly.row-length-def)
apply(auto simp add: scalar-prod)
apply(simp add: computation-downpos)
done

lemma rotate-downpos-inv: rotate-downpos w1 w2 \implies (kauff-mat w1) = (kauff-mat w2)
unfolding rotate-downpos-def using rotate-downpos-kauff-mat by auto

lemma rotate-downneg-kauff-mat:
kauff-mat ((basic [cup,vert]) ◦ (basic [vert,under])) = kauff-mat ((basic [vert,cup]) ◦ (basic [over,vert]))
apply(simp add: mat-multI-def)
apply(simp add: matT-vec-multI-def)
apply(auto simp add: replicate-def rat-poly.row-length-def)
apply(auto simp add: scalar-prod)
apply(simp add: computation-downpos)
done

lemma rotate-downneg-inv: rotate-downneg w1 w2 \implies (kauff-mat w1) = (kauff-mat w2)
unfolding rotate-downneg-def using rotate-downneg-kauff-mat by auto

theorem rotate-inv: rotate w1 w2 \implies (kauff-mat w1) = (kauff-mat w2)
unfolding rotate-def using rotate-downneg-inv rotate-downpos-inv rotate-topneg-inv
rotate-toppos-inv by auto

lemma positive-flip-kauff-mat:
kauff-mat (left-over) = kauff-mat (right-over)
apply(simp add:mat-multI-def)
apply(simp add:matT-vec-multI-def)
apply(auto simp add:replicate-def rat-poly.row-length-def)
apply(auto simp add:scalar-prod)
using computation-positive-flip apply auto[1]
using computation-positive-flip by auto

lemma uncross-positive-flip-inv: uncross-positive-flip w1 w2 \implies (kauff-mat w1)
= (kauff-mat w2)
unfolding uncross-positive-flip-def using positive-flip-kauff-mat by auto

lemma negative-flip-kauff-mat: kauff-mat (left-under) = kauff-mat (right-under)
apply(simp add:mat-multI-def)
apply(simp add:matT-vec-multI-def)
apply(auto simp add:replicate-def rat-poly.row-length-def)
apply(auto simp add:scalar-prod)
using computation-negative-flip apply auto
done

lemma uncross-negative-flip-inv: uncross-negative-flip w1 w2 \implies (kauff-mat w1)
= (kauff-mat w2)
unfolding uncross-negative-flip-def using negative-flip-kauff-mat by auto

theorem framed-uncross-inv:(framed-uncross w1 w2) \implies (kauff-mat w1) = (kauff-mat w2)
unfolding framed-uncross-def using uncross-negative-flip-inv uncross-positive-flip-inv
by auto

lemma pos-neg-kauff-mat:
kauff-mat ((basic [over]) o (basic [under]))
= kauff-mat ((basic [vert,vert]) o (basic [vert,vert]))
apply(simp add:mat-multI-def)
apply(simp add:matT-vec-multI-def)
apply(auto simp add:replicate-def rat-poly.row-length-def)
apply(auto simp add:scalar-prod)
apply(auto simp add:inverse1 inverse2)
apply(auto simp add:computation-pull-pos-neg)
done

lemma pull-posneg-inv: pull-posneg w1 w2 \implies (kauff-mat w1) = (kauff-mat w2)
unfolding pull-posneg-def using pos-neg-kauff-mat by auto

lemma neg-pos-kauff-mat:kauff-mat ((basic [under]) o (basic [over]))
= kauff-mat ((basic [vert,vert]) o (basic [vert,vert]))
apply(simp add:mat-multI-def)
apply\((\text{simp add: matT-vec-mulI-def})\)
apply\((\text{auto simp add: replicate-def rat-poly.row-length-def})\)
apply\((\text{auto simp add: scalar-prod})\)
apply\((\text{auto simp add: inverse1 inverse2})\)
using computation-pull-pos-neg by \((\text{simp add: computation-downpos})\)

lemma pull-negpos-inv\: pull-negpos \(w1\) \(w2\) \(\Rightarrow\) \((\text{kauff-mat } w1) = (\text{kauff-mat } w2)\)
unfolding pull-negpos-def using neg-pos-kauff-mat by auto

theorem pull-inv\: pull \(w1\) \(w2\) \(\Rightarrow\) \((\text{kauff-mat } w1) = (\text{kauff-mat } w2)\)
unfolding pull-def using pull-posneg-inv pull-negpos-inv by auto

theorem slide-inv\: slide \(w1\) \(w2\) \(\Rightarrow\) \((\text{kauff-mat } w1) = (\text{kauff-mat } w2)\)
proof
  assume assm\: slide \(w1\) \(w2\)
  have \(\exists\ B.\ ((w1 = ((\text{basic (make-vert-block (nat (domain-block B)))})\circ(\text{basic } B)))\)
    \& \((w2 = ((\text{basic } B)\circ(\text{basic (make-vert-block (nat (codomain-block B)))})))))\)
    \& ((domain-block B) \(\neq\) 0)) using slide-def assm by auto
  then obtain B where \(((w1 = ((\text{basic (make-vert-block (nat (domain-block B)))})\circ(\text{basic } B))))\)
    \& ((domain-block B) \(\neq\) 0)) by auto
  then have kauff-mat \(w1\) = kauff-mat (basic B)
    proof
      have s1\:\text{mat} \((2^\text{nat (domain-block B))})\) (length (blockmat B)) (blockmat B)
        by (metis matrix-blockmat row-length-domain-block)
      have \(w1 = ((\text{basic (make-vert-block (nat (domain-wall (basic B)))})\circ(\text{basic } B)))\)
        using 1 domain-wall.simps by auto
      then have kauff-mat \(w1\) = rat-poly.matrix-mult
        (kauff-mat (basic (make-vert-block (nat (domain-wall (basic B)))))
          (kauff-mat (basic B)))
        using tangle-compose-matrix is-tangle-diagram.simps
        by (metis compose-Nil kauff-mat.simps(1) kauff-mat.simps(2))
      moreover then have \(\ldots = \text{rat-poly.matrix-mult (mat1 (2^\text{nat (domain-block B))}) (blockmat B)}\)
        using kauff-mat.simps(1) domain-wall.simps(1) by (metis make-vert-block-map-blockmat)
      moreover have \(\ldots = (\text{blockmat B})\)
        using s1 mat1-mult-left by (metis make-vert-equiv-mat prop-make-vert-equiv(1))
      ultimately show \(?thesis by auto
      qed
    moreover have kauff-mat \(w2\) = kauff-mat (basic B)
    proof
      

106
have \( s1 : \text{mat} \left( 2^n \text{ (domain-block B)} \right) \left( 2^n \text{ (codomain-block B)} \right) \) (blockmat B)

by \( \text{metis length-codomain-block matrix-blockmat row-length-domain-block} \)

have \( w2 = \left( (\text{basic B}) \circ (\text{make-vert-block (nat (codomain-wall (basic B)))}) \right) \)

using 1 domain-wall.simps by auto

then have kauff-mat \( w2 = \)

rat-poly.matrix-mult

(kauff-mat (basic B))

(kauff-mat (basic (make-vert-block (nat (codomain-wall (basic B))))))

using kauff-mat.simps(1) domain-wall.simps(1)

moreover then have \( \ldots = \) rat-poly.matrix-mult (blockmat B) (mat1 \( 2^n \text{ (nat (codomain-block B))} \))

using kauff-mat.simps(1) domain-wall.simps(1)

moreover have \( \ldots = \) (blockmat B)

using \( s1 \) by (metis blockmat-make-vert-codomain-wall.simps(1) make-vert-equiv-mat)

ultimately show \( ?\text{thesis by auto} \)

qed

ultimately show \( ?\text{thesis by auto} \)

qed

theorem framed-linkrel-inv:framed-linkrel \( w1 w2 \Rightarrow (\text{kauff-mat } w1) = (\text{kauff-mat } w2) \)

unfolding framed-linkrel-def

apply(auto)

using framed-uncross-inv pull-inv straighten-inv swing-inv rotate-inv compress-inv slide-inv

by auto

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:(\( w1::\text{wall} \)) \( \sim f w2 \Rightarrow \text{kauff-mat } w1 = \text{kauff-mat } w2 \)

proof(induction rule:Framed-Tangle-Equivalence.induct)

case refl

show ?case using refl by auto
next
case sym
  show ?case using sym by auto
next
case trans
  show ?case using trans by auto
next
case compose-eq
  show ?case using compose-eq tangle-compose-matrix by auto
next
case codomain-compose
  show ?case using codomain-compose left-mat-compose by auto
next
case domain-compose
  show ?case using domain-compose right-mat-compose by auto
next
case tensor-eq
  show ?case using tensor-eq.IH Tensor-Invariance by (metis)
next
case equality
  show ?case using framed-linkrel-inv equality by auto
qed

lemma rat-poly-times A B = 1
  using inverse1 by (metis)

we calculate kauffman bracket of a few links

kauffman bracket of an unknot with zero crossings

lemma kauff-mat ([cup]*)*(basic [cap]) = \([-\langle A^2\rangle - \langle B^2\rangle\])
apply(simp add: mat-multI-def)
apply(simp add: matT-vec-multI-def)
apply(auto simp add:replicate-def rat-poly,row-length-def)
apply(auto simp add:scalar-prod)
by (simp add: power2-eq-square)

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

lemma kauff-mat ([cup,cup]*)*(basic [cap, cap]) = \([\langle (A^4) + (B^4) \rangle + 2]\)
apply(simp add: mat-multI-def)
apply(simp add:matT-vec-multI-def)
apply(auto simp add:replicate-def rat-poly,row-length-def)
apply(auto simp add:scalar-prod)
apply(auto simp add:unlink-computation)
done

definition trefoil-polynomial::rat-poly
where
\[
\text{trefoil-polynomial} \equiv \text{rat-poly-plus}
\]
\[
\begin{align*}
& \quad \text{(rat-poly-times \ (rat-poly-times \ A \ A)} \\
& \quad \text{(rat-poly-plus)} \\
& \quad \text{(rat-poly-times \ B)} \\
& \quad \text{(rat-poly-times \ B)} \\
& \quad \text{(rat-poly-times \ (A - \ rat-poly-times \ (rat-poly-times \ B \ B) \ B)} \\
& \quad \text{(rat-poly-times \ A \ A)))} \\
& \quad \text{(rat-poly-times \ (A - \ rat-poly-times \ (rat-poly-times \ B \ B) \ B)} \\
& \quad \text{(rat-poly-plus \ (rat-poly-times \ B \ (rat-poly-times \ B \ (rat-poly-times \ (A - \ rat-poly-times \ (rat-poly-times \ B \ B) \ B) \ (rat-poly-times \ A \ A))))} \\
& \quad \text{(rat-poly-times \ (A - \ rat-poly-times \ (rat-poly-times \ B \ B) \ B)} \\
& \quad \text{(rat-poly-times \ (rat-poly-times \ B \ B) \ (rat-poly-times \ A \ A))}) \\
& \quad \text{(rat-poly-plus \ (rat-poly-times \ 2)} \\
& \quad \text{(rat-poly-times \ 2)} \\
& \quad \text{(rat-poly-times \ A)} \\
& \quad \text{(rat-poly-times \ A)} \\
& \quad \text{(rat-poly-times \ (rat-poly-times \ A \ (rat-poly-times \ A \ (rat-poly-times \ B \ B)))}) \\
& \quad \text{(rat-poly-times \ (rat-poly-times \ B \ B) \ (rat-poly-times \ B))} \\
& \quad \text{(rat-poly-times \ (A - \ rat-poly-times \ (rat-poly-times \ B \ B) \ B)} \\
& \quad \text{(rat-poly-times \ B \ (rat-poly-times \ B \ (rat-poly-times \ B \ B)))})
\end{align*}
\]

\text{kauffman bracket of trefoil}

\textbf{lemma trefoil:}
\[
\text{kauff-mat} \ (\text{[cap,cap]*[vert,over,vert]*[vert,over,vert]*[vert,over,vert] *(basic [cap,cap])}) = [\text{[trefoil-polynomial]}]
\]
\textbf{by} (simp add: mat-multI-def matT-vec-multI-def rat-poly.row-length-def scalar-prod trefoil-polynomial-def)

\textbf{end}

\textbf{theory Knot-Theory}

\textbf{imports Kauffman-Invariance Example}

\textbf{begin}

\end