

Knot Theory

T. V. H. Prathamesh

December 14, 2021

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

10 Rational Functions	43
11 Kauffman matrices	46
12 Computations: This section can be skipped	87
13 Tangle moves and Kauffman bracket	98
14 Kauffman_Invariance: Proving the invariance of Kauffman Bracket	107

1 Preliminaries: Definitions of tangles and links

```
theory Preliminaries
imports Main
begin
```

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

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

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

block is obtained by putting bricks next to each other

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

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

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

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

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

```
lemma empty-concatenate: xs ⊗ Nil = xs
by (induction xs) (auto)
```

Associativity properties of Concatenation

lemma *leftright-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* :: *wall* => *wall* => *wall* (**infix** \circ 66) **where**
compose-Nil: $(\text{basic } x) \circ ys = \text{prod } x \text{ } ys$ |
compose-Cons: $((\text{prod } x \text{ } xs) \circ ys) = \text{prod } x \text{ } (xs \circ ys)$

Associativity properties of composition

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

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

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

primrec *block-length*::*block* => *nat*
where
block-length [] = 0 |
block-length (Cons *x y*) = 1 + (*block-length y*)

primrec *domain*::*brick* => *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 $\forall x. (x = \text{vert}) \vee (x = \text{over}) \vee (x = \text{under}) \vee (x = \text{cap}) \vee (x = \text{cup})$
by (*metis brick.exhaust*)
moreover have

$\forall x.(((x = \text{vert}) \vee (x = \text{over}) \vee (x = \text{under}) \vee (x = \text{cap}) \vee (x = \text{cup})) \longrightarrow (\text{domain } x \geq 0))$
using *domain.simps* **by** (*metis order-refl zero-le-numeral zero-le-one*)
ultimately show *?thesis* **by** *auto*
qed

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

where

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

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

where

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

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

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

where

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

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

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

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

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

lemma *domain-wall-compose: domain-wall (xs \circ ys)* = *domain-wall xs*
by (*induction xs*) (*auto*)

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

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

absolute value

definition $\text{abs}::\text{int} \Rightarrow \text{int}$ **where**
 $\text{abs } x \equiv \text{if } (x \geq 0) \text{ then } x \text{ else } (0 - x)$

theorems about abs

lemma *abs-zero*: **assumes** $\text{abs } x = 0$ **shows** $x = 0$
using *abs-def* *assms eq-iff-diff-eq-0*
by *metis*

lemma *abs-zero-equality*: **assumes** $\text{abs } (x - y) = 0$ **shows** $x = y$
using *assms abs-zero eq-iff-diff-eq-0*
by *blast*

lemma *abs-non-negative*: $\text{abs } x \geq 0$
using *abs-def* *diff-0* *le-cases* *neg-0-le-iff-le*
by *auto*

lemma *abs-non-negative-sum*: **assumes** $\text{abs } x + \text{abs } y = 0$
shows $\text{abs } x = 0$ **and** $\text{abs } y = 0$
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*: $(\text{domain } x) \geq 0$
using *domain.simps* *brick.exhaust* *le-cases* *not-numeral-le-zero* *zero-le-one* **by**
(*metis*)

lemma *codomain-nonnegative*: $(\text{codomain } x) \geq 0$
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*: $\text{domain-block } x \geq 0$
by (*induction x*)(*auto simp add: domain-nonnegative*)

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

by (*induction x*)(*auto simp add: codomain-nonnegative*)

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

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

proof –

have $(\text{domain-block } (x\#y)) = (\text{domain } x) + (\text{domain-block } y)$ **by** *auto*

also have $(\text{domain } x) = (\text{domain-block } (x\#Nil))$ **by** *auto*

then have $(\text{domain-block } (x\#Nil) > 0) = (\text{domain } x > 0)$ **by** *auto*

then have $((\text{domain } x > 0) \vee (\text{domain-block } y > 0)) \implies (\text{domain } x + \text{domain-block } y > 0)$

using *domain-nonnegative add-nonneg-pos add-pos-nonneg domain-block-nonnegative*

by *metis*

from *this*

show $((\text{domain-block}(x\#Nil)) > 0) \vee ((\text{domain-block } y) > 0)$
 $\implies (\text{domain-block } (x\#y) > 0)$

by *auto*

qed

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 $\text{domain-block } x = 0$ **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** $\text{domain-block } y > 0$ or $\text{domain-block } y > 0$

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** $\text{codomain-block } y > 0$ or $\text{codomain-block } y > 0$

shows $(\text{codomain-block } (x\otimes y)) > 0$

apply (*simp add: codomain-additive*)

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* **have** *domain-block y = 0*

by (*metis* (*full-types*) *Cons.prem 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.prem 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  $\Rightarrow$  bool
where
is-tangle-diagram (basic x) = True
|is-tangle-diagram (x*xs) = (if is-tangle-diagram xs
  then (codomain-block x = domain-wall xs)
  else False)

```

```

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

```

end

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

```

theory Tangles
imports Preliminaries
begin

```

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

```

typedef Tangle-Diagram = {(x::wall). is-tangle-diagram x}
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*
and *is-tangle-diagram y*
and *(Abs-Tangle-Diagram x) = (Abs-Tangle-Diagram y)*
shows *x = y*
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 ◦ y) ⇒ 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*zs)◦y = (z*(zs◦y))* **by** *auto*
then have *is-tangle-diagram (z*(zs◦y))* **using** *prod* **by** *auto*
moreover then have *1: is-tangle-diagram zs*
using *is-tangle-diagram.simps(2) prod.hyps prod.prem* **by** *metis*
ultimately have *domain-wall (zs◦y) = codomain-block z*
by (*metis is-tangle-diagram.simps(2)*)
moreover have *domain-wall (zs◦y) = domain-wall zs*
using *domain-wall-def domain-wall-compose* **by** *auto*
ultimately have *domain-wall zs = codomain-block z* **by** *auto*
then have *is-tangle-diagram (z*zs)*
by (*metis 1 is-tangle-diagram.simps(2)*)
then show *?case* **by** *auto*
qed

lemma *is-tangle-right-compose:*
is-tangle-diagram (x ◦ y) ⇒ is-tangle-diagram y

```

proof (induct x)
  case (basic z)
    have (basic z)  $\circ$  y = (z*y) using basic by auto
    then have is-tangle-diagram y
      unfolding is-tangle-diagram.simps(2) using basic.prem.s by (metis is-tangle-diagram.simps(2))
    then show ?case using basic.prem.s by auto
  next
  case (prod z zs)
    have ((z*zs)  $\circ$  y) = (z *(zs  $\circ$  y)) by auto
    then have is-tangle-diagram (z*(zs  $\circ$  y)) using prod by auto
    then have is-tangle-diagram (zs  $\circ$  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)
   $\wedge$ (codomain-wall x = domain-wall y))
   $\implies$  is-tangle-diagram (x  $\circ$  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)  $\circ$  y = z*y
      using compose-def by auto
    ultimately have codomain-block z = domain-wall y
      using basic.prem.s 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.prem.s by metis
    then have codomain-block z = domain-wall zs
      using is-tangle-diagram.simps(2) prod.prem.s by metis
    then have codomain-block z = domain-wall (zs  $\circ$  y)
      using domain-wall.simps domain-wall-compose by auto
    moreover have is-tangle-diagram (zs  $\circ$  y)
      using prod.hyps by (metis codomain-wall.simps(2) is-tangle-diagram.simps(2)
prod.prem.s)
    ultimately have is-tangle-diagram (z*(zs  $\circ$  y))
      unfolding is-tangle-diagram-def by auto
    then show ?case by auto

```

qed

lemma *composition-of-tangle-diagrams*:
assumes *is-tangle-diagram* x
and *is-tangle-diagram* y
and $(\text{domain-wall } y = \text{codomain-wall } x)$
shows *is-tangle-diagram* $(x \circ y)$
using *comp-of-tangle-dgms* using *assms* by *auto*

lemma *converse-composition-of-tangle-diagrams*:
 $\text{is-tangle-diagram } (x \circ y) \implies (\text{domain-wall } y) = (\text{codomain-wall } x)$
proof (*induct* x)
case (*basic* z)
have $(\text{basic } z) \circ y = z * y$
using *compose-def basic* by *auto*
then have
 $\text{is-tangle-diagram } ((\text{basic } z) \circ y) \implies$
 $(\text{is-tangle-diagram } y) \wedge (\text{codomain-block } z = \text{domain-wall } y)$
using *is-tangle-diagram.simps(2)* by (*metis*)
then have $(\text{codomain-block } z) = (\text{domain-wall } y)$
using *basic.prem*s by *auto*
moreover have $\text{codomain-wall } (\text{basic } z) = \text{codomain-block } z$
using *domain-wall-compose* by *auto*
ultimately have $(\text{codomain-wall } (\text{basic } z)) = (\text{domain-wall } y)$
by *auto*
then show *?case* by *simp*
next
case (*prod* z zs)
have $\text{codomain-wall } zs = \text{domain-wall } y$
using *prod.hyps prod.prem*s
by (*metis compose-Nil compose-leftassociativity is-tangle-right-compose*)
moreover have $\text{codomain-wall } zs = \text{codomain-wall } (z * zs)$
using *domain-wall-compose* by *auto*
ultimately show *?case* by *metis*

qed

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

(**infixl** \circ 65)

where
compose-Tangle x $y = \text{Abs-Tangle-Diagram}$
 $((\text{Rep-Tangle-Diagram } x) \circ (\text{Rep-Tangle-Diagram } y))$

theorem *well-defined-compose*:
assumes *is-tangle-diagram* x
and *is-tangle-diagram* y

and $\text{domain-wall } x = \text{codomain-wall } y$
shows $(\text{Abs-Tangle-Diagram } x) \circ (\text{Abs-Tangle-Diagram } y)$
 $\quad = (\text{Abs-Tangle-Diagram } (x \circ y))$
using $\text{Abs-Tangle-Diagram-inverse } \text{assms}(1) \text{ assms}(2) \text{ compose-Tangle-def}$
 $\quad \text{mem-Collect-eq}$
by *auto*

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

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

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 $\text{make-vert-block}::\text{nat} \Rightarrow \text{block}$
where
 $\text{make-vert-block } 0 = []$
 $|\text{make-vert-block } (\text{Suc } n) = \text{vert}\#(\text{make-vert-block } n)$

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

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

fun $\text{tensor}::\text{wall} \Rightarrow \text{wall} \Rightarrow \text{wall}$ (**infixr** \otimes 65)
where
 $1:\text{tensor } (\text{basic } x) (\text{basic } y) = (\text{basic } (x \otimes y))$
 $|\text{2:tensor } (x*cs) (\text{basic } y) = ($

```

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

```

5 Properties of tensor product of tangles

lemma *Nil-left-tensor*: $xs \otimes (\text{basic } []) = xs$
by (*cases xs*) (*auto simp add:empty-concatenate*)

lemma *Nil-right-tensor*: $(\text{basic } []) \otimes xs = xs$
by (*cases xs*) (*auto*)

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

definition *tensor-Tangle* :: *Tangle-Diagram* \Rightarrow *Tangle-Diagram* \Rightarrow *Tangle-Diagram*
(**infixl** \otimes 65)

where

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

lemma *tensor (basic [vert]) (basic ([vert])) = (basic (([vert]) ⊗ ([vert])))*
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 $(xs \otimes ys) = \text{domain-wall } xs + \text{domain-wall } ys$

proof(*cases xs*)

fix x

assume $A:xs = \text{basic } x$

then have *domain-wall* $(xs \otimes ys) = \text{domain-wall } xs + \text{domain-wall } ys$

proof(*cases ys*)

fix y

assume $B:ys = \text{basic } y$

have *domain-block* $(x \otimes y) = \text{domain-block } x + \text{domain-block } y$

using *domain-additive* **by** *auto*

then have *domain-wall* $(xs \otimes ys) = \text{domain-wall } xs + \text{domain-wall } ys$

```

    using tensor.simps(1) A B by auto
  thus ?thesis by auto
next
fix z zs
assume C:ys = (z*zs)
have domain-wall (xs ⊗ ys) = domain-wall xs + domain-wall ys
proof(cases (codomain-block x) = 0)
  assume codomain-block x = 0
  then have (xs ⊗ ys) = (x ⊗ z)*zs
    using A C tensor.simps(4) by auto
  then have domain-wall (xs ⊗ ys) = domain-block (x ⊗ z)
    by auto
  moreover have domain-wall ys = domain-block z
    unfolding domain-wall-def C by auto
  moreover have domain-wall xs = domain-block x
    unfolding domain-wall-def A by auto
  moreover have domain-block (x ⊗ z) = domain-block x + domain-block z
    using domain-additive by auto
  ultimately show ?thesis by auto
next
assume codomain-block x ≠ 0
have (xs ⊗ ys)
  = (x ⊗ z)
    *((basic (make-vert-block (nat (codomain-block x)))) ⊗ zs)
  using tensor.simps(3) A C ⟨codomain-block x ≠ 0⟩ by auto
then have domain-wall (xs ⊗ ys) = domain-block (x ⊗ z)
  by auto
  moreover have domain-wall ys = domain-block z
    unfolding domain-wall-def C by auto
  moreover have domain-wall xs = domain-block x
    unfolding domain-wall-def A by auto
  moreover have domain-block (x ⊗ z) = domain-block x + domain-block z
    using domain-additive by auto
  ultimately show ?thesis by auto
qed
then show ?thesis by auto
qed
then show ?thesis by auto
next
fix z zs
assume D:xs = z * zs
then have domain-wall (xs ⊗ ys) = domain-wall xs + domain-wall ys
proof(cases ys)
  fix y
  assume E:ys = basic y
  then have domain-wall (xs ⊗ ys) = domain-wall xs + domain-wall ys
  proof(cases codomain-block y = 0)
    assume codomain-block y = 0
    have (xs ⊗ ys) = (z ⊗ y)*zs

```

```

    using tensor.simps(2) D E ‹codomain-block y = 0› by auto
  then have domain-wall (xs ⊗ ys) = domain-block (z ⊗ y)
    by auto
  moreover have domain-wall xs = domain-block z
    unfolding domain-wall-def D by auto
  moreover have domain-wall ys = domain-block y
    unfolding domain-wall-def E by auto
  moreover have domain-block (z ⊗ y) = domain-block z + domain-block y
    using domain-additive by auto
  ultimately show ?thesis by auto
next
assume codomain-block y ≠ 0
have (xs ⊗ ys)
  =
  (z ⊗ y)
  *(zs⊗(basic (make-vert-block (nat (codomain-block y))))))
  using tensor.simps(3) D E ‹codomain-block y ≠ 0› by auto
then have domain-wall (xs ⊗ ys) = domain-block (z ⊗ y)
  by auto
moreover have domain-wall ys = domain-block y
  unfolding domain-wall-def E by auto
moreover have domain-wall xs = domain-block z
  unfolding domain-wall-def D by auto
moreover have domain-block (z ⊗ y) = domain-block z + domain-block y
  using domain-additive by auto
ultimately show ?thesis by auto
qed
then show ?thesis by auto
next
fix w ws
assume F:ys = w*ws
have (xs ⊗ ys) = (z ⊗ w) * (zs ⊗ ws)
  using D F by auto
then have domain-wall (xs ⊗ ys) = domain-block (z ⊗ w)
  by auto
moreover have domain-wall ys = domain-block w
  unfolding domain-wall-def F by auto
moreover have domain-wall xs = domain-block z
  unfolding domain-wall-def D by auto
moreover have domain-block (z ⊗ w) = domain-block z + domain-block w
  using domain-additive by auto
ultimately show ?thesis by auto
qed
then show ?thesis by auto
qed

```

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

lemma *tensor-codomain-wall-additivity*:

```

codomain-wall (xs ⊗ ys) = codomain-wall xs + codomain-wall ys
proof(induction xs ys rule:tensor.induct)
fix xs ys
let ?case = (codomain-wall ((basic xs) ⊗ (basic ys))
              = (codomain-wall (basic (xs)))
                + (codomain-wall (basic (ys))))
show ?case using codomain-wall.simps codomain-block.simps tensor.simps
by (metis codomain-additive)

next
fix x xs y
assume case-2:
  codomain-block y ≠ 0
  ⇒ codomain-wall
    (xs ⊗ basic (make-vert-block (nat (codomain-block y))))
    = codomain-wall xs
      + codomain-wall
        (basic (make-vert-block (nat (codomain-block y))))

let ?case = codomain-wall ((x*xs) ⊗ (basic y))
          = (codomain-wall (x*xs)) + (codomain-wall (basic y))
show ?case
proof(cases (codomain-block y = 0))
case True
  have ((x*xs) ⊗ (basic y)) = (x ⊗ y)*xs
    using Tangle-Algebra.2 True by auto
  then have codomain-wall ((x*xs) ⊗ (basic y))
    = codomain-wall ((x ⊗ y)*xs)
    by auto
  then have ... = codomain-wall xs
    using codomain-wall.simps by auto
  then have ... = codomain-wall xs + codomain-wall (basic y)
    using True codomain-wall.simps(1) by auto
  then show ?thesis by auto
next
case False
  have (x*xs) ⊗ (basic y)
    = (x ⊗ y)
      *(xs ⊗ (basic (make-vert-block (nat (codomain-block y)))))
    using False by (metis Tangle-Algebra.2)
  moreover then have codomain-wall ((x*xs) ⊗ (basic y))
    = codomain-wall(...)
    by auto
  moreover have ...
    = codomain-wall
      (xs ⊗ (basic (make-vert-block (nat (codomain-block y)))))
    using domain-wall.simps by auto
  moreover have ...
    = codomain-wall xs
      + codomain-wall

```



```

      (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 ‹basic  $x \otimes y * ys = (x \otimes y) * (basic$ 
(make-vert-block (nat (codomain-block  $x$ )))  $\otimes ys$ › ‹codomain-wall  $((x \otimes y) * (basic$ 
(make-vert-block (nat (codomain-block  $x$ )))  $\otimes ys)$  = codomain-wall (basic (make-vert-block
(nat (codomain-block  $x$ )))  $\otimes ys$ › ‹codomain-wall (basic (make-vert-block (nat (codomain-block
 $x$ )))  $\otimes ys$ ) = codomain-block  $x + codomain-wall$   $ys$ ›)
qed
next
fix  $x$   $xs$   $y$   $ys$ 
assume case-4:codomain-wall  $(xs \otimes ys) = codomain-wall$   $xs + codomain-wall$ 
 $ys$ 
let ?case = codomain-wall  $((x*xs) \otimes (y*ys))$ 
= codomain-wall  $(x*xs) + codomain-wall$   $(y*ys)$ 
have  $((x*xs) \otimes (y*ys)) = (x \otimes y)*(xs \otimes ys)$ 
using 4 by auto
then have codomain-wall  $(...) = codomain-wall$   $(...)$ 
by auto
then have  $... = codomain-wall$   $(xs \otimes ys)$ 
using codomain-wall.simps(2) by auto
then have  $... = codomain-wall$   $xs + codomain-wall$   $ys$ 
using case-4 by auto
then have  $... = codomain-wall$   $(x*xs) + (codomain-wall$   $(y*ys))$ 
using codomain-wall.simps(2) by auto
then show ?case by (metis ‹codomain-wall  $((x \otimes y) * (xs \otimes ys)) = codomain-wall$ 
 $(xs \otimes ys)$ › ‹ $x * xs \otimes y * ys = (x \otimes y) * (xs \otimes ys)$ › case-4)
qed

```

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

```

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

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))
  = (codomain-block  $x$ ) + codomain-block(make-vert-block 0)
```

using *codomain-additive* **by** *auto*

moreover have *codomain-block* (*make-vert-block* 0) = 0

by *auto*

ultimately have *codomain-block* $(x \otimes (make-vert-block$ 0)) = *codomain-block* (x)

by *auto*

```

moreover have is-tangle-diagram  $xs$ 
  using prod.prems by (metis is-tangle-diagram.simps(2))
then have is-tangle-diagram  $((x \otimes (\text{make-vert-block } 0)) * xs)$ 
  using is-tangle-diagram.simps(2) by (metis calculation prod.prems)
then have is-tangle-diagram  $((x * xs) \otimes (\text{basic } (\text{make-vert-block } 0)))$ 
  by auto
then show ?thesis using  $0$  by (metis)
next
case (Suc k)
have codomain-block  $(\text{make-vert-block } (k+1)) = \text{int } (k+1)$ 
  using codomain-make-vert by auto
then have  $(\text{nat } (\text{codomain-block } (\text{make-vert-block } (k+1)))) = k+1$ 
  by auto
then have  $\text{make-vert-block } (\text{nat } (\text{codomain-block } (\text{make-vert-block } (k+1))))$ 
   $= \text{make-vert-block } (k+1)$ 
  by auto
moreover have codomain-wall  $(\text{basic } (\text{make-vert-block } (k+1))) > 0$ 
  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  $1: (x * xs) \otimes (\text{basic } (\text{make-vert-block } (k+1)))$ 
   $= (x \otimes (\text{make-vert-block } (k+1))) * (xs \otimes (\text{basic } (\text{make-vert-block } (k+1))))$ 
  using tensor.simps(2) by simp
have domain-wall  $(xs \otimes (\text{basic } (\text{make-vert-block } (k+1))))$ 
   $= \text{domain-wall } xs + \text{domain-wall } (\text{basic } (\text{make-vert-block } (k+1)))$ 
  using tensor-domain-wall-additivity by auto
then have  $2:$ 
   $\text{domain-wall } (xs \otimes (\text{basic } (\text{make-vert-block } (k+1))))$ 
   $= (\text{domain-wall } xs) + \text{int } (k+1)$ 
  using domain-make-vert domain-wall.simps(1) by auto
moreover have  $3:$  codomain-block  $(x \otimes (\text{make-vert-block } (k+1)))$ 
   $= \text{codomain-block } x + \text{int } (k+1)$ 
  using codomain-additive codomain-make-vert by (metis)
have is-tangle-diagram  $(x * xs)$ 
  using prod.prems by auto
then have  $4:$  codomain-block  $x = \text{domain-wall } xs$ 
  using is-tangle-diagram.simps(2) by metis
from  $2\ 3\ 4$  have
   $\text{domain-wall } (xs \otimes (\text{basic } (\text{make-vert-block } (k+1))))$ 
   $= \text{codomain-block } (x \otimes (\text{make-vert-block } (k+1)))$ 
  by auto
moreover have is-tangle-diagram  $(xs \otimes (\text{basic } (\text{make-vert-block } (k+1))))$ 
  using prod.hyps prod.prems by (metis Suc Suc-eq-plus1 is-tangle-diagram.simps(2))
ultimately have is-tangle-diagram  $((x * xs) \otimes (\text{basic } (\text{make-vert-block } (k+1))))$ 
  using  $1$  by auto
then show ?thesis using Suc Suc-eq-plus1 by metis
qed
then show ?case by auto
qed

```

```

theorem is-tangle-make-vert-left:
  (is-tangle-diagram xs)  $\implies$  is-tangle-diagram ((basic (make-vert-block n))  $\otimes$  xs)
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 ( (make-vert-block 0)  $\otimes$  x)
          = (codomain-block x) + codomain-block(make-vert-block 0)
          using codomain-additive by auto
      moreover have codomain-block (make-vert-block 0) = 0
        by auto
      ultimately have codomain-block ( (make-vert-block 0)  $\otimes$  x) = codomain-block
(x)
        by auto
      moreover have is-tangle-diagram xs
        using prod.prems by (metis is-tangle-diagram.simps(2))
      then have is-tangle-diagram (( (make-vert-block 0)  $\otimes$  x)*xs)
        using is-tangle-diagram.simps(2) by (metis calculation prod.prems)
      then have is-tangle-diagram ((basic (make-vert-block 0))  $\otimes$  (x*xs))
        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: (basic (make-vert-block (k+1)))  $\otimes$  (x*xs)
        = ((make-vert-block (k+1))  $\otimes$  x)*((basic (make-vert-block (k+1)))  $\otimes$ 
xs)
        using tensor.simps(3) by simp
      have domain-wall ((basic (make-vert-block (k+1)))  $\otimes$  xs)
        = domain-wall xs + domain-wall (basic (make-vert-block (k+1)))
        using tensor-domain-wall-additivity by auto

```

then have 2:
 $domain-wall ((basic (make-vert-block (k+1))) \otimes xs)$
 $= (domain-wall xs) + int (k+1)$
using *domain-make-vert domain-wall.simps(1)* **by auto**
moreover have 3: $codomain-block ((make-vert-block (k+1)) \otimes x)$
 $= codomain-block x + int (k+1)$
using *codomain-additive codomain-make-vert*
by (*simp add: codomain-additive*)
have *is-tangle-diagram (x*xs)*
using *prod.premis* **by auto**
then have 4: $codomain-block x = domain-wall xs$
using *is-tangle-diagram.simps(2)* **by metis**
from 2 3 4 have
 $domain-wall ((basic (make-vert-block (k+1))) \otimes xs)$
 $= codomain-block ((make-vert-block (k+1)) \otimes x)$
by auto
moreover have *is-tangle-diagram ((basic (make-vert-block (k+1))) \otimes xs)*
using *prod.hyps prod.premis* **by** (*metis Suc Suc-eq-plus1 is-tangle-diagram.simps(2)*)
ultimately have *is-tangle-diagram ((basic (make-vert-block (k+1))) \otimes (x*xs))*
using 1 **by auto**
then show ?thesis using *Suc Suc-eq-plus1* **by metis**
qed
then show ?case by auto
qed

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

proof –
assume *A: (codomain-block y) ≠ 0*
assume *B:*
 $is-tangle-diagram (xs)$
 $\wedge is-tangle-diagram ((basic (make-vert-block (nat (codomain-block y))))))$
 \implies
 $is-tangle-diagram (xs \otimes ((basic (make-vert-block (nat (codomain-block y))))))$
have $is-tangle-diagram (x * xs) \wedge is-tangle-diagram (basic y) \implies is-tangle-diagram (x * xs \otimes basic y)$
 xs
by (*metis is-tangle-diagram.simps(2)*)
moreover have $is-tangle-diagram (basic (make-vert-block (nat (codomain-block y))))$
using *is-tangle-diagram.simps(1)* **by auto**
ultimately have

$((is-tangle-diagram\ xs)$
 $\wedge(is-tangle-diagram\ (basic\ (make-vert-block\ (nat\ (codomain-block\ y))))))$
 $\longrightarrow is-tangle-diagram\ (xs\ \otimes\ basic\ (make-vert-block\ (nat\ (codomain-block\ y))))$
 \implies
 $is-tangle-diagram\ (x * xs) \wedge is-tangle-diagram\ (basic\ y) \longrightarrow$
 $is-tangle-diagram\ (xs\ \otimes\ basic\ (make-vert-block\ (nat\ (codomain-block\ y))))$
by metis
moreover have $1: codomain-block\ y = domain-wall\ (basic\ (make-vert-block\ (nat\ (codomain-block\ y))))$
using $codomain-block-nonnegative\ domain-make-vert\ domain-wall.simps(1)$
int-nat-eq **by auto**
moreover have $2: is-tangle-diagram\ (x * xs) \wedge is-tangle-diagram\ (basic\ y) \longrightarrow$
 $codomain-block\ x = domain-wall\ xs$
using $is-tangle-diagram.simps(2)$ **by metis**
moreover have $codomain-block\ (x\ \otimes\ y) = codomain-block\ x + codomain-block\ y$
using $codomain-additive$ **by auto**
moreover have $domain-wall\ (xs\ \otimes\ (basic\ (make-vert-block\ (nat\ (codomain-block\ y))))))$
 $= domain-wall\ xs + domain-wall\ (basic\ (make-vert-block\ (nat\ (codomain-block\ y))))$
using $tensor-domain-wall-additivity$ **by auto**
moreover then have $is-tangle-diagram\ (x * xs) \wedge is-tangle-diagram\ (basic\ y)$
 \longrightarrow
 $domain-wall\ (xs\ \otimes\ (basic\ (make-vert-block\ (nat\ (codomain-block\ y))))))$
 $= codomain-block\ (x\ \otimes\ y)$
by $(metis\ 1\ 2\ calculation(4))$
ultimately have
 $(is-tangle-diagram\ xs)$
 $\wedge(is-tangle-diagram\ (basic\ (make-vert-block\ (nat\ (codomain-block\ y))))))$
 $\longrightarrow is-tangle-diagram\ (xs\ \otimes\ basic\ (make-vert-block\ (nat\ (codomain-block\ y))))$
 \implies
 $is-tangle-diagram\ (x * xs) \wedge is-tangle-diagram\ (basic\ y) \longrightarrow$
 $is-tangle-diagram\ ((x\ \otimes\ y) * (xs\ \otimes\ (basic\ (make-vert-block\ (nat\ (codomain-block\ y))))))$
using $is-tangle-diagram.simps(2)$ **by auto**
then have
 $is-tangle-diagram\ (x * xs) \wedge is-tangle-diagram\ (basic\ y) \longrightarrow$
 $is-tangle-diagram\ ((x * xs) \otimes (basic\ y))$
by $(metis\ Tangle-Algebra.2\ \langle$
 $codomain-block\ y \neq 0 \rangle is-tangle-make-vert-right)$
then show $?thesis$ **by auto**
qed

lemma *simp2*:

$(codomain-block\ x) \neq 0$
 \implies
 $is-tangle-diagram\ (basic\ (make-vert-block\ (nat\ (codomain-block\ x))))$
 $\wedge is-tangle-diagram\ (ys)$

\longrightarrow
 $is-tangle-diagram ((basic (make-vert-block (nat (codomain-block x)))) \otimes ys)$
 \implies
 $(is-tangle-diagram (basic x)$
 $\wedge is-tangle-diagram (y*ys)$
 $\longrightarrow is-tangle-diagram ((basic x) \otimes (y*ys)))$

proof –

assume $A: (codomain-block x) \neq 0$

assume $B: is-tangle-diagram (basic (make-vert-block (nat (codomain-block x))))$
 $\wedge is-tangle-diagram (ys) \longrightarrow$
 $is-tangle-diagram$
 $((basic (make-vert-block (nat (codomain-block x)))) \otimes ys)$

have $is-tangle-diagram (basic x) \wedge is-tangle-diagram (y*ys)$
 $\longrightarrow is-tangle-diagram ys$

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

moreover have $(is-tangle-diagram$
 $(basic (make-vert-block (nat (codomain-block x))))))$

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

ultimately have
 $((is-tangle-diagram ys)$
 $\wedge (is-tangle-diagram (basic (make-vert-block (nat (codomain-block x))))))$
 $\longrightarrow is-tangle-diagram ((basic (make-vert-block (nat (codomain-block x)))) \otimes$
 $ys))$
 \implies
 $is-tangle-diagram (basic x) \wedge is-tangle-diagram (y*ys) \longrightarrow$
 $is-tangle-diagram$
 $((basic (make-vert-block (nat (codomain-block x)))) \otimes ys)$

by $metis$

moreover have $1: codomain-block x$
 $= domain-wall (basic (make-vert-block (nat (codomain-block$
 $x))))$

using $codomain-block-nonnegative domain-make-vert domain-wall.simps(1)$
 $int-nat-eq$ **by** $auto$

moreover have $2: is-tangle-diagram (basic x) \wedge is-tangle-diagram (y*ys) \longrightarrow$
 $codomain-block y = domain-wall ys$

using $is-tangle-diagram.simps(2)$ **by** $metis$

moreover have $codomain-block (x \otimes y) = codomain-block x + codomain-block y$

using $codomain-additive$ **by** $auto$

moreover have $domain-wall ((basic (make-vert-block (nat (codomain-block x))))$
 $\otimes ys)$
 $= domain-wall (basic (make-vert-block (nat (codomain-block x))))$
 $+ domain-wall ys$

using $tensor-domain-wall-additivity$ **by** $auto$

moreover then have $is-tangle-diagram (basic x) \wedge is-tangle-diagram (y*ys) \longrightarrow$
 $domain-wall ((basic (make-vert-block (nat (codomain-block x)))) \otimes ys)$
 $= codomain-block (x \otimes y)$

by $(metis 1 2 calculation(4))$

ultimately have

$(is-tangle-diagram\ ys)$
 $\wedge (is-tangle-diagram\ (basic\ (make-vert-block\ (nat\ (codomain-block\ x))))))$
 $\longrightarrow is-tangle-diagram\ ((basic\ (make-vert-block\ (nat\ (codomain-block\ x)))) \otimes$
 $ys)$
 \implies
 $is-tangle-diagram\ (basic\ x) \wedge is-tangle-diagram\ (y*ys)$
 \longrightarrow
 $is-tangle-diagram\ ((x \otimes y)*((basic\ (make-vert-block\ (nat\ (codomain-block$
 $x)))) \otimes ys))$
using *is-tangle-diagram.simps(2)* **by auto**
then have
 $is-tangle-diagram\ (basic\ x) \wedge is-tangle-diagram\ (y*ys) \longrightarrow$
 $is-tangle-diagram\ ((basic\ x) \otimes (y*ys))$
by (*metis Tangle-Algebra.3 A B*)
then show *?thesis* **by auto**
qed

lemma *simp3*:

$is-tangle-diagram\ xs \wedge is-tangle-diagram\ ys \longrightarrow is-tangle-diagram\ (xs \otimes ys)$
 \implies
 $is-tangle-diagram\ (x * xs) \wedge is-tangle-diagram\ (y * ys)$
 $\longrightarrow is-tangle-diagram\ (x * xs \otimes y * ys)$

proof –

assume *A*: $is-tangle-diagram\ xs \wedge is-tangle-diagram\ ys \longrightarrow is-tangle-diagram\ (xs$
 $\otimes ys)$
have $is-tangle-diagram\ (x*xs) \longrightarrow (codomain-block\ x = domain-wall\ xs)$
using *is-tangle-diagram.simps(2)* **by auto**
moreover have $is-tangle-diagram\ (y*ys) \longrightarrow (codomain-block\ y = domain-wall$
 $ys)$
using *is-tangle-diagram.simps(2)* **by auto**
ultimately have $is-tangle-diagram\ (x*xs) \wedge is-tangle-diagram\ (y*ys)$
 $\longrightarrow codomain-block\ (x \otimes y) = domain-wall\ (xs \otimes ys)$
using *codomain-additive tensor-domain-wall-additivity* **by auto**
moreover have $is-tangle-diagram\ (x*xs) \wedge is-tangle-diagram\ (y*ys)$
 $\longrightarrow is-tangle-diagram\ (xs \otimes ys)$
using *A* *is-tangle-diagram.simps(2)* **by auto**
moreover have $(x*xs) \otimes (y*ys) = (x \otimes y)*(xs \otimes ys)$
using *tensor.simps(4)* **by auto**
ultimately have $is-tangle-diagram\ (x*xs) \wedge is-tangle-diagram\ (y*ys)$
 $\longrightarrow is-tangle-diagram\ ((x*xs) \otimes (y*ys))$
by auto
then show *?thesis* **by simp**
qed

theorem *is-tangle-diagramness*:

shows $(is-tangle-diagram\ x) \wedge (is-tangle-diagram\ y) \longrightarrow is-tangle-diagram\ (tensor\ x$
 $y)$


```

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 ys
let ?case = (is-tangle-diagram (basic x))^(is-tangle-diagram (y*ys))
           →is-tangle-diagram ((basic x) ⊗ (y*ys)))
from simp2 show ?case
by (metis Tangle-Algebra.3 codomain-additive comm-monoid-add-class.add-0

      is-tangle-diagram.simps(2) is-tangle-make-vert-left)
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))
  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 rtran-
clpD symmetry3 symp-def tranclp-into-rtranclp)

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

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

Link uncross

abbreviation $right-over::wall$

where

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

abbreviation $left-over::wall$

where

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

abbreviation $right-under::wall$

where

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

abbreviation $left-under::wall$

where

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

abbreviation $straight-line::wall$

where

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

definition $uncross-positive-flip::relation$

where

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

definition $uncross-positive-straighten::relation$

where

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

definition $uncross\text{-}negative\text{-}flip::relation$

where

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

definition $uncross\text{-}negative\text{-}straighten::relation$

where

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

definition $uncross$

where

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

swing begins

abbreviation $r\text{-}over\text{-}braid::wall$

where

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

abbreviation $l\text{-}over\text{-}braid::wall$

where

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

abbreviation $r\text{-}under\text{-}braid::wall$

where

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

abbreviation $l\text{-}under\text{-}braid::wall$

where

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

definition $swing\text{-}pos::wall \Rightarrow wall \Rightarrow bool$

where

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

definition $swing\text{-}neg::wall \Rightarrow wall \Rightarrow bool$

where

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

definition *swing::relation*

where

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

pull begins

definition *pull-posneg::relation*

where

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

definition *pull-negpos::relation*

where

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

pull definition

definition *pull::relation*

where

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

linkrel-pull ends

linkrel-straighten

definition *straighten-topdown::relation*

where

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

definition *straighten-downtop::relation*

where

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

definition straighten

definition *straighten::relation*

where

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

straighten ends

rotate moves

definition *rotate-toppos::relation*

where

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

definition *rotate-topneg::wall* \Rightarrow *wall* \Rightarrow *bool*

where

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

definition *rotate-downpos::wall* \Rightarrow *wall* \Rightarrow *bool*

where

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

definition *rotate-downneg::wall* \Rightarrow *wall* \Rightarrow *bool*

where

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

rotate definition

definition *rotate::wall* \Rightarrow *wall* \Rightarrow *bool*

where

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

rotate ends

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

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

where

$$\begin{aligned} \text{compress-top1 } x y \equiv & \exists B. ((x = (\text{basic } (\text{make-vert-block } (\text{nat } (\text{domain-wall } B)))) \circ \\ & B) \\ & \wedge (y = B) \wedge (\text{codomain-wall } B \neq 0) \\ & \wedge (\text{is-tangle-diagram } B)) \end{aligned}$$

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

where

compress-bottom1 x y \equiv $\exists B.((x = B \circ (\text{basic } (\text{make-vert-block } (\text{nat } (\text{codomain-wall } B))))))$
 $\wedge (y = B) \wedge (\text{domain-wall } B \neq 0)$
 $\wedge (\text{is-tangle-diagram } B)$

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

where

compress-bottom x y \equiv $\exists B.((x = B \circ (\text{basic } (\text{make-vert-block } (\text{nat } (\text{codomain-wall } B))))))$
 $\wedge (y = ((\text{basic } (\square)) \circ B)) \wedge (\text{domain-wall } B = 0)$
 $\wedge (\text{is-tangle-diagram } B)$

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

where

compress-top x y \equiv $\exists B.((x = (\text{basic } (\text{make-vert-block } (\text{nat } (\text{domain-wall } B)))) \circ B)$
 $\wedge (y = (B \circ (\text{basic } (\square)))) \wedge (\text{codomain-wall } B = 0)$
 $\wedge (\text{is-tangle-diagram } B)$

definition *compress::wall* \Rightarrow *wall* \Rightarrow *bool*

where

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

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

definition *slide::wall* \Rightarrow *wall* \Rightarrow *bool*

where

slide x y \equiv $\exists B.((x = ((\text{basic } (\text{make-vert-block } (\text{nat } (\text{domain-block } B)))) \circ (\text{basic } B)))$
 $\wedge (y = ((\text{basic } B) \circ (\text{basic } (\text{make-vert-block } (\text{nat } (\text{codomain-block } B))))))$
 $\wedge ((\text{domain-block } B) \neq 0)$

linkrel-definition

definition *linkrel::wall* \Rightarrow *wall* \Rightarrow *bool*

where

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

definition *framed-uncross::wall* \Rightarrow *wall* \Rightarrow *bool*

where

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

definition *framed-linkrel::wall* \Rightarrow *wall* \Rightarrow *bool*

where

$\text{framed-linkrel } x \ y = ((\text{framed-uncross } x \ y) \vee (\text{pull } x \ y) \vee (\text{straighten } x \ y) \vee (\text{swing } x \ y) \vee (\text{rotate } x \ y) \vee (\text{compress } x \ y) \vee (\text{slide } x \ y))$

lemma *framed-uncross-implies-uncross*: $(\text{framed-uncross } x \ y) \implies (\text{uncross } x \ y)$
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* :: $\text{wall} \Rightarrow \text{wall} \Rightarrow \text{bool}$ (**infixl** \sim 64)

where

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

inductive *Framed-Tangle-Equivalence* :: $\text{wall} \Rightarrow \text{wall} \Rightarrow \text{bool}$ (**infixl** $\sim f$ 64)

where

$\text{refl } [\text{intro!}, \text{Pure.intro!}, \text{simp}]: a \sim f a$
 $\text{equality } [\text{Pure.intro}]: \text{framed-linkrel } a \ b \implies a \sim f b$
 $\text{domain-compose}: (\text{domain-wall } a = 0) \wedge (\text{is-tangle-diagram } a) \implies a \sim f ((\text{basic } \square) \circ a)$
 $\text{codomain-compose}: (\text{codomain-wall } a = 0) \wedge (\text{is-tangle-diagram } a) \implies a \sim f (a \circ (\text{basic } \square))$


```

|compose-eq:((B::wall) ~f D) ^ ((A::wall) ~f C)
  ^ (is-tangle-diagram A) ^ (is-tangle-diagram B)
  ^ (is-tangle-diagram C) ^ (is-tangle-diagram D)
  ^ (domain-wall B) = (codomain-wall A)
  ^ (domain-wall D) = (codomain-wall C)
  ==> ((A::wall) o B) ~f (C o D)
|trans: A ~f B ==> B ~f C ==> A ~f C
|sym: A ~f B ==> B ~f A
|tensor-eq: ((B::wall) ~f D) ^ ((A::wall) ~f C) ^ (is-tangle-diagram A) ^ (is-tangle-diagram
B)
  ^ (is-tangle-diagram C) ^ (is-tangle-diagram D) ==> ((A::wall) o B) ~f (C o D)

```

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

(infixl ~ T 64)

where

Tangle-Diagram-Equivalence T1 T2 \equiv

(*Rep-Tangle-Diagram* T1) ~ (*Rep-Tangle-Diagram* T2)

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

(infixl ~ L 64)

where

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

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

morphisms *Rep-Tangles* *Abs-Tangles*

proof (rule equivI)

show reflp *Tangle-Diagram-Equivalence*

unfolding reflp-def *Tangle-Diagram-Equivalence-def*

Tangle-Equivalence.refl

by auto

show symp *Tangle-Diagram-Equivalence*

unfolding *Tangle-Diagram-Equivalence-def* symp-def

using *Tangle-Diagram-Equivalence-def* *Tangle-Equivalence.sym*

by auto

show transp *Tangle-Diagram-Equivalence*

unfolding *Tangle-Diagram-Equivalence-def* transp-def

using *Tangle-Diagram-Equivalence-def* *Tangle-Equivalence.trans*

by metis

qed

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

morphisms *Rep-Links* *Abs-Links*

proof (rule equivI)

show reflp *Link-Diagram-Equivalence*

unfolding reflp-def *Link-Diagram-Equivalence-def*

Tangle-Equivalence.refl

```

    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  $\Rightarrow$  Tangle-Diagram
 $\Rightarrow$  bool
  (infixl  $\sim$  T 64)
where
  Framed-Tangle-Diagram-Equivalence T1 T2
     $\equiv$  (Rep-Tangle-Diagram T1)  $\sim$  (Rep-Tangle-Diagram T2)

definition Framed-Link-Diagram-Equivalence::Link-Diagram  $\Rightarrow$  Link-Diagram  $\Rightarrow$ 
bool
  (infixl  $\sim$  L 64)
where
  Framed-Link-Diagram-Equivalence T1 T2
     $\equiv$  (Rep-Link-Diagram T1)  $\sim$  (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 Framed-Tangle-Diagram-Equivalence-def symp-def
    using Framed-Tangle-Diagram-Equivalence-def
    Framed-Tangle-Equivalence.sym
    by (metis Tangle-Equivalence.sym)
  show transp Framed-Tangle-Diagram-Equivalence
    unfolding Framed-Tangle-Diagram-Equivalence-def transp-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  $a \sim b$  and  $b \sim c$  shows  $a \sim c$ 
  using Tangle-Equivalence.trans assms(1) assms(2) by metis

```

```

lemma prelim-cup-compress:

```

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

```

proof –

```

```

  have domain-wall (basic (cup # [])) = 0
    by auto

```

```

  moreover have codomain-wall (basic (cup # [])) = 2
    by auto

```

```

moreover

```

```

  have make-vert-block (nat (codomain-wall (basic (cup # []))))
    = (vert # vert # [])

```

```

    unfolding make-vert-block-def
    by auto

```

```

moreover have is-tangle-diagram ((basic (cup#[])) \circ (basic (vert # vert # [])))
  using is-tangle-diagram.simps by auto

```

```

ultimately

```

```

  have compress-bottom
    ((basic (cup#[])) \circ (basic (vert # vert # [])))
    ((basic [])\circ(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

$x \equiv (basic [cup,cup]) \circ (basic [vert,over,vert]) \circ (basic [cap,cap])$

abbreviation *y::wall*

where

$y \equiv (basic [cup]) \circ (basic [cap])$

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 linkrel right-over straight-line
  using linkrel-def by auto
then have 2:right-over ~ straight-line
  using Tangle-Equivalence.equality by auto
have (left-over ~ straight-line) ∧ (right-over ~ straight-line)
  ⇒ ?thesis
  using transitive by auto
then show ?thesis using 1 2 transitive by blast
qed

```

theorem *Example:*

```

 $x \sim y$ 
proof–
have 1:left-over ~ straight-line
  using Tangle-Equivalence.equality uncross-straighten-left-over by auto
moreover have 2:straight-line ~ straight-line
  using refl by auto
have 3:(left-over ⊗ straight-line) ~ (straight-line ⊗ straight-line)
proof–
have is-tangle-diagram (left-over)
  unfolding is-tangle-diagram-def by auto
moreover have is-tangle-diagram (straight-line)
  unfolding is-tangle-diagram-def by auto
ultimately show ?thesis using 1 2 by (metis Tangle-Equivalence.tensor-eq)
qed
then have 4:
  ((basic (cup#[])) ∘ (left-over ⊗ straight-line))
  ~ ((basic (cup#[])) ∘ (straight-line ⊗ straight-line))
proof–
have is-tangle-diagram (left-over ⊗ straight-line)
  by auto
moreover have is-tangle-diagram (straight-line ⊗ straight-line)
  by auto
moreover have is-tangle-diagram (basic (cup#[]))
  by auto
moreover have domain-wall (left-over ⊗ straight-line) = (codomain-wall (basic
(cup#[])))
  unfolding domain-wall-def by auto
moreover have domain-wall (straight-line ⊗ straight-line) = (codomain-wall
(basic (cup#[])))
  unfolding domain-wall-def by auto
moreover have (basic (cup#[])) ~ (basic (cup#[]))
  using refl by auto
ultimately show ?thesis
  using compose-eq 3 by (metis Tangle-Equivalence.compose-eq)
qed
moreover have 5: (basic [cup]) ∘ (straight-line ⊗ straight-line)

```

$\sim (\text{basic } [\text{cup}])$

proof-
have 0 :
 $(\text{basic } ([\text{cup}])) \circ (\text{straight-line} \otimes \text{straight-line}) = (\text{basic } [\text{cup}]) \circ (\text{basic } [\text{vert}, \text{vert}])$
 $\circ (\text{basic } [\text{vert}, \text{vert}]) \circ (\text{basic } [\text{vert}, \text{vert}])$
by auto
let $?x = (\text{basic } (\text{cup}\#\[]))$
 $\circ (\text{basic } (\text{vert}\#\text{vert}\#\[])) \circ (\text{basic } (\text{vert}\#\text{vert}\#\[]))$
 $\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}\#\[])) = \text{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 $(\text{metis } \text{Tangle-Equivalence.compose-eq is-tangle-diagram.simps}(1))$

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

```

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

```

```

(?y))
  using tensor-eq cup-compress Nil-right-tensor is-tangle-diagram.simps(1)
refl
  by (metis Tangle-Equivalence.tensor-eq)
then have ((?y)  $\otimes$  (?y)) = (basic ([ ])  $\otimes$  ([ ]))
   $\circ$  ((basic (cup#[ ])  $\otimes$  (basic (cup#[ ])))
  using tensor.simps(4) by (metis compose-Nil)
then have ((?y)  $\otimes$  (?y)) = (basic ([ ])  $\circ$  (basic (cup#cup#[ ])))
  using tensor.simps(1) concatenate-def by auto
then have (?y)  $\otimes$  ((basic (cup#[ ])  $\circ$  (basic (vert#vert#[ ])))
   $\sim$  (basic ([ ])  $\circ$  (basic (cup#cup#[ ])))
  using 7 by auto
moreover have (basic ([ ])  $\circ$  (basic (cup#cup#[ ]))  $\sim$  (basic (cup#cup#[ ])))
proof-
  have domain-wall (basic (cup#cup#[ ]) = 0
  by auto
  then show ?thesis using domain-compose sym
  by (metis Tangle-Equivalence.domain-compose Tangle-Equivalence.sym
is-tangle-diagram.simps(1))
qed
ultimately have (?y)  $\otimes$  ((basic (cup#[ ])  $\circ$  (basic (vert#vert#[ ])))
   $\sim$  (basic (cup#cup#[ ])))
  using trans by (metis (full-types) Example.transitive)
then have (basic(cup#[ ])  $\circ$  (basic(cup#vert#vert#[ ]))  $\sim$  (basic(cup#cup#[ ])))
  by auto
moreover have ?temp  $\sim$  ?temp
  using refl by auto
moreover have is-tangle-diagram ((basic(cup#[ ])  $\circ$  (basic(cup#vert#vert#[ ])))
  by auto
moreover have is-tangle-diagram (basic(cup#cup#[ ]))
  by auto
moreover have is-tangle-diagram (?temp)
  by auto
moreover have codomain-wall ((basic(cup#[ ])  $\circ$  (basic(cup#vert#vert#[ ])))
  = domain-wall ?temp
  by auto
moreover have codomain-wall (basic(cup#cup#[ ]) = domain-wall ?temp
  by auto
ultimately have 8: ((basic(cup#[ ])  $\circ$  (basic(cup#vert#vert#[ ])))  $\circ$  (?temp)
   $\sim$  (basic(cup#cup#[ ])  $\circ$  (?temp)
  using compose-eq by (metis Tangle-Equivalence.compose-eq)
then have ((basic [cup,cup]  $\circ$  (?temp))
   $\sim$  (basic [cup]  $\circ$  (left-over  $\otimes$  straight-line))
  using 55 compose-leftassociativity sym wall.simps
  by (metis Tangle-Equivalence.sym compose-Nil)
moreover have (basic [cup]  $\circ$  (left-over  $\otimes$  straight-line)
   $\sim$  (basic [cup]  $\circ$  (straight-line  $\otimes$  straight-line))
  using 4 by auto
ultimately have ((basic [cup,cup]  $\circ$  (?temp))

```



```

      ~ (basic [cup]) ◦ (straight-line ⊗ straight-line)
proof–
  have ((basic [cup,cup]) ◦ (?temp))
    ~ (basic [cup] ◦ (left-over ⊗ straight-line))
    using 8 55 compose-leftassociativity sym wall.simps Tangle-Equivalence.sym
compose-Nil
    by (metis)
  moreover have (basic [cup]) ◦ (left-over ⊗ straight-line)
    ~ (basic [cup]) ◦ (straight-line ⊗ straight-line)
    using 4 by auto
  moreover have (((basic [cup,cup]) ◦ (?temp))
    ~ (basic [cup] ◦ (left-over ⊗ straight-line)))
  ∧ ((basic [cup]) ◦ (left-over ⊗ straight-line)
    ~ (basic [cup]) ◦ (straight-line ⊗ straight-line))
    ⇒ ?thesis
    using Example.transitive by auto
  ultimately show ?thesis by auto
qed
then have (basic ([cup,cup])) ◦ (?temp) ~ (basic (cup # []))
  using trans transp-def 5 by (metis Example.transitive)
moreover have (basic (cap#[[]])) ~ (basic (cap#[[]]))
  using refl by auto
moreover have is-tangle-diagram ((basic(cup#cup#[[]])) ◦ (?temp))
  by auto
moreover have is-tangle-diagram (basic (cup # []))
  by auto
moreover have is-tangle-diagram (basic (cap # []))
  by auto
moreover have codomain-wall ((basic(cup#cup#[[]])) ◦ (?temp))
  = domain-wall (basic (cap # []))
  by auto
moreover have codomain-wall (basic(cup#[[]])) = domain-wall (basic (cap # []))
  by auto
ultimately have 9:((basic(cup#cup#[[]])) ◦ (?temp)) ◦ (basic (cap#[[]]))
  ~ (basic (cup#[[]])) ◦ (basic (cap#[[]]))
  using Tangle-Equivalence.compose-eq by metis
let ?z = ((basic(cup#cup#[[]])) ◦ (basic(vert#over#vert#[[]])))
have 10:((basic(cup#cup#[[]])) ◦ (?temp)) ◦ (basic (cap#[[]]))
  = ?z ◦ ((basic(cap#vert#vert#[[]])) ◦ (basic (cap#[[]])))
  by auto
then have 11:((basic(cap#vert#vert#[[]])) ◦ (basic (cap#[[]])))
  = ((basic ((cap#[[]])⊗(vert#vert#[[]]))◦(basic (([]) ⊗(cap#[[]])))
  unfolding concatenate-def by auto
then have 12: ((basic(cap#vert#vert#[[]])) ◦ (basic (cap#[[]])))
  = ((basic (cap#[[]])◦(basic ([])))⊗((basic (vert#vert#[[]])◦(basic
(cap#[[]])))
  using tensor.simps by auto
let ?w = ((basic (cap#[[]])◦(basic ([])))
have 13:((basic (vert#vert#[[]])◦(basic (cap#[[]]))) ~ ?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#* [])) ◦ (*basic* (*cap#* []))) ?w
using *compress-top-def* **by** *auto*
then have *compress* ((*basic* (*vert#vert#* [])) ◦ (*basic* (*cap#* []))) ?w
using *compress-def* **by** *auto*
then have *linkrel* ((*basic* (*vert#vert#* [])) ◦ (*basic* (*cap#* []))) ?w
using *linkrel-def* **by** *auto*
then have ((*basic* (*vert#vert#* [])) ◦ (*basic* (*cap#* []))) ~ ?w
using *Tangle-Equivalence.equality* **by** *auto*
then show ?thesis **by** *simp*
qed
moreover have *is-tangle-diagram* ((*basic* (*vert#vert#* [])) ◦ (*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#* [])) ◦ (*basic* (*cap#* []))) ~ ((?w) ⊗
(?w))
using *Tangle-Equivalence.tensor-eq* **by** *metis*
then have ((*basic* (*cap#vert#vert#* [])) ◦ (*basic* (*cap#* []))) ~ ((?w) ⊗ (?w))
using 13 **by** *auto*
moreover have ((?w) ⊗ (?w)) = (*basic* (*cap#cap#* [])) ◦ (*basic* ([]))
using *tensor.simps* **by** *auto*
ultimately have ((*basic* (*cap#vert#vert#* [])) ◦ (*basic* (*cap#* []))) ~ (*basic* (*cap#cap#* [])) ◦ (*basic*
([]))
by *auto*
moreover have ?z ~ ?z
using *refl* **by** *auto*
moreover have *domain-wall* ((*basic* (*cap#cap#* [])) ◦ (*basic* ([])))
 = *codomain-wall* (?z)
by *auto*
moreover have *domain-wall* (((*basic* (*cap#vert#vert#* [])) ◦ (*basic* (*cap#* []))))
 = *codomain-wall* (?z)
by *auto*
moreover have *is-tangle-diagram* ((*basic* (*cap#vert#vert#* [])) ◦ (*basic* (*cap#* [])))
by *auto*
moreover have *is-tangle-diagram* (?z)
by *auto*
moreover have *is-tangle-diagram* ((*basic* (*cap#cap#* [])) ◦ (*basic* ([])))
by *auto*
ultimately have 14: (?z) ◦ ((*basic* (*cap#vert#vert#* [])) ◦ (*basic* (*cap#* [])))
 ~ (?z) ◦ ((*basic* (*cap#cap#* [])) ◦ (*basic* ([]))) (**is** ?aa ~ ?bb)

```

using Tangle-Equivalence.compose-eq by metis
moreover have 15: ((?z) ∘ ((basic(cap#cap#[]))) ∘ (basic ([])))
  ~ ((?z) ∘ (basic(cap#cap#[]))) (is ?bb ~ ?cc)
using Tangle-Equivalence.codomain-compose Tangle-Equivalence.sym
  ⟨is-tangle-diagram (basic [cap, cap] ∘ basic [])⟩ codomain-wall-compose
  compose-leftassociativity converse-composition-of-tangle-diagrams
  domain-block.simps(1) domain-wall.simps(1)
by (metis (opaque-lifting, mono-tags) Tangle-Equivalence.compose-eq
  Tangle-Equivalence.refl
  ⟨codomain-wall (basic [cup, cup])
    = domain-wall (basic [vert, over, vert] ∘ basic [cap, vert, vert])⟩

    ⟨domain-wall (basic [cap, cap] ∘ basic [])
  = codomain-wall (basic [cup, cup] ∘ basic [vert, over, vert])⟩
    comp-of-tangle-dgms domain-wall-compose is-tangle-diagram.simps(1))
ultimately have (?aa ~ ?bb) ∧ (?bb ~ ?cc) ⇒ ?aa ~ ?cc
  using transitive by auto
then have 16: ?aa ~ ?cc
  using 14 15 by auto
then have 17: ((basic (cup#[])) ∘ (basic (cap#[]))) ~ ?aa
  using 9 10 Tangle-Equivalence.trans Tangle-Equivalence.sym
  by (metis (opaque-lifting, no-types))
have (((basic (cup#[])) ∘ (basic (cap#[]))) ~ ?aa) ∧ (?aa ~ ?cc)
  ⇒ ((basic (cup#[])) ∘ (basic (cap#[]))) ~ ?cc
  using transitive by auto
then have ((basic (cup#[])) ∘ (basic (cap#[]))) ~ ?cc
  using 17 16 by auto
then show ?thesis using Tangle-Equivalence.sym by auto
qed
end

```

9 Kauffman Matrix and Kauffman Bracket- Definitions and Properties

```

theory Kauffman-Matrix
imports
  Matrix-Tensor.Matrix-Tensor
  Link-Algebra
  HOL-Computational-Algebra.Polynomial
  HOL-Computational-Algebra.Fraction-Field
begin

```

10 Rational Functions

intpoly is the type of integer polynomials

```

type-synonym intpoly = int poly

```

lemma *eval-pCons*: $\text{poly } (\text{pCons } 0 \ 1) \ x = x$
using *poly-1 poly-pCons* **by** *auto*

lemma *pCons2*: $(\text{pCons } 0 \ 1) \neq (1::\text{int } \text{poly})$
using *eval-pCons poly-1 zero-neq-one* **by** *metis*

definition *var-def*: $x = (\text{pCons } 0 \ 1)$

lemma *non-zero*: $x \neq 0$
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*: $A = \text{Fract } x \ 1$

definition *var-def2*: $B = \text{Fract } 1 \ x$

lemma *assumes* $b \neq 0$ **and** $d \neq 0$
shows $\text{Fract } a \ b = \text{Fract } c \ d \iff a * d = c * b$
using *eq-fract assms* **by** *auto*

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

unfolding *var-def1*

proof(*rule ccontr*)

assume $0: \neg (\text{Fract } x \ 1 \neq (0::\text{rat-poly}))$

then have $\text{Fract } x \ 1 = (0::\text{rat-poly})$

by *auto*

moreover have $(0::\text{rat-poly}) = \text{Fract } (0::\text{intpoly}) \ (1::\text{intpoly})$

by (*metis Zero-fract-def*)

ultimately have $\text{Fract } x \ (1::\text{intpoly}) = \text{Fract } (0::\text{intpoly}) \ (1::\text{intpoly})$

by *auto*

moreover have $(1::\text{intpoly}) \neq 0$

by *auto*

ultimately have $x * (1::\text{intpoly}) = (0::\text{intpoly}) * (1::\text{intpoly})$

using *eq-fract* **by** *metis*

then have $x = (0::\text{intpoly})$

by *auto*

then show *False* **using** *non-zero* **by** *auto*

qed

lemma *mult-inv-non-zero*:

assumes $(p::\text{rat-poly}) \neq 0$

and $p * q = (1::\text{rat-poly})$

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*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],[0,A]] [[A,0],[0,A]] = [[A*A,A],[0,A*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 \Rightarrow *rat-poly mat* \Rightarrow *rat-poly mat*
(infixl \otimes *65)*

where

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

lemma assumes (*j::nat*) *div a = i div a*
and *j mod a = i mod a*
shows *j = i*

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 $[[1]] \otimes M = M$
by (*metis rat-poly.Tensor-left-id*)

lemma $M \otimes [[1]] = M$
by (*metis rat-poly.Tensor-right-id*)

11 Kauffman matrices

We assign every brick to a matrix of rational polynomials

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

where

brickmat vert = $[[1,0],[0,1]]$
brickmat cup = $[[0],[A],[-B],[0]]$
brickmat cap = $[[0,-A,B,0]]$
brickmat over = $[[A,0,0,0],$
 $[0,0,B,0],$
 $[0,B,A-(B*B*B),0],$
 $[0,0,0,A]]$
brickmat under = $[[B,0,0,0],$
 $[0,B-(A*A*A),A,0],$
 $[0,A,0,0],$
 $[0,0,0,B]]$

lemma *inverse1:rat-poly-times A B = 1*
using *non-zero One-fract-def monoid-mult-class.mult.right-neutral*

mult-fract mult-fract-cancel var-def1 var-def2
by (*metis (opaque-lifting, no-types)*)

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)*
 $= (\text{rat-poly-times } p \ q) + (\text{rat-poly-times } p \ r)$
by (*metis rat-poly.plus-left-distributivity*)

lemma *minus-left-distributivity:*
 $\text{rat-poly-times } p \ (q - r)$
 $= (\text{rat-poly-times } p \ q) - (\text{rat-poly-times } p \ r)$
using *minus-mult-right right-diff-distrib* **by** *blast*

lemma *minus-right-distributivity:*
 $\text{rat-poly-times } (p - q) \ r = (\text{rat-poly-times } p \ r) - (\text{rat-poly-times } q \ r)$
using *minus-left-distributivity rat-poly.comm* **by** *metis*

lemma *equation:*
rat-poly-plus
 $(\text{rat-poly-times } B \ (B - \text{rat-poly-times } (\text{rat-poly-times } A \ A) \ A))$
 $(\text{rat-poly-times } (A - \text{rat-poly-times } (\text{rat-poly-times } B \ B) \ B) \ A)$
 $= 0$

proof–

have $\text{rat-poly-times } (\text{rat-poly-times } A \ A) \ A$
 $= ((A * A) * A)$

by *auto*

then have $\text{rat-poly-times } B \ (B - \text{rat-poly-times } (\text{rat-poly-times } A \ A) \ A)$
 $= B * B - B * ((A * A) * A)$

using *minus-left-distributivity* **by** *auto*

moreover have $\dots = B * B - (B * (A * (A * A)))$

by *auto*

moreover have $\dots = B * B - ((B * A) * (A * A))$

by *auto*

moreover have $\dots = B * B - A * A$

using *inverse2* **by** *auto*

ultimately have 1:

$\text{rat-poly-times } B \ (B - \text{rat-poly-times } (\text{rat-poly-times } A \ A) \ A)$
 $= B * B - A * A$

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 *minus-right-distributivity* **by auto**
moreover have $\dots = (A*A) - ((B*B)*(B*A))$
by auto
moreover have $\dots = (A*A) - (B*B)$
using *inverse2* **by auto**
ultimately have ? :
 $(\text{rat-poly-times } (A - \text{rat-poly-times } (\text{rat-poly-times } B B) B) A)$
 $= (A*A) - (B*B)$
by auto
have $B*B - A*A + (A*A) - (B*B) = 0$
by auto
with ? ?thesis **by auto**
qed

lemma *rat-poly.matrix-mult* (*brickmat over*) (*brickmat under*)
 $= [[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]]$
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:equation*)
done

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

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

using *rat-poly.row-length-def* **by auto**

lemma *cup-dim:rat-poly.row-length* (*brickmat cup*) $= 1$ **and** $\text{length } (\text{brickmat cup}) = 4$

using *rat-poly.row-length-def* **by auto**

lemma *cap-dim:rat-poly.row-length* (*brickmat cap*) $= 4$ **and** $\text{length } (\text{brickmat cap}) = 1$

using *rat-poly.row-length-def* **by auto**

lemma *over-dim:rat-poly.row-length* (*brickmat over*) $= 4$ **and** $\text{length } (\text{brickmat over}) = 4$

using *rat-poly.row-length-def* **by auto**

lemma *under-dim:rat-poly.row-length* (*brickmat under*) $= 4$ **and** $\text{length } (\text{brickmat under}) = 4$

using *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) = (2nat (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 (blockmat ls) = (2nat ((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) =  $2^{\text{nat}(\text{domain-block } ls)}$ 
    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 ... =  $2 * (2^{\text{nat}(\text{domain-block } ls)})$ 
    using rat-poly.row-length-def Cons vert by auto
  moreover have ... =  $2^{(1 + \text{nat}(\text{domain-block } ls))}$ 
    using domain-block.simps by auto
  moreover have ... =  $2^{(\text{nat}(\text{domain } l) + \text{nat}(\text{domain-block } ls))}$ 
    using domain.simps vert by auto
  moreover have ... =  $2^{(\text{nat}(\text{domain } l + \text{domain-block } ls))}$ 
    using Suc-eq-plus1-left Suc-nat-eq-nat-zadd1
      calculation(4) domain.simps(1) domain-block-non-negative
      vert
    by (metis)
  moreover have ... =  $2^{(\text{nat}(\text{domain-block } (l\#ls)))}$ 
    using domain-block.simps by auto
  ultimately show ?thesis by metis
next
case over
  have rat-poly.row-length (blockmat ls) =  $2^{\text{nat}(\text{domain-block } ls)}$ 
    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 ... =  $4 * (2^{\text{nat}(\text{domain-block } ls)})$ 
    using rat-poly.row-length-def Cons over by auto
  also have ... =  $2^{(2 + \text{nat}(\text{domain-block } ls))}$ 
    using domain-block.simps by auto
  also have ... =  $2^{(\text{nat}(\text{domain } l) + \text{nat}(\text{domain-block } ls))}$ 
    using domain.simps over by auto
  also have ... =  $2^{(\text{nat}(\text{domain } l + \text{domain-block } ls))}$ 
    by (simp add: nat-add-distrib domain-block-nonnegative over)
  also have ... =  $2^{(\text{nat}(\text{domain-block } (l\#ls)))}$ 
    by simp
  finally show ?thesis .
next
case under
  have rat-poly.row-length (blockmat ls) =  $2^{\text{nat}(\text{domain-block } ls)}$ 
    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 ... =  $4 * (2^{\text{nat}(\text{domain-block } ls)})$ 

```

```

    using rat-poly.row-length-def Cons under 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 under by auto
  also have ... = 2^(nat (domain l + domain-block ls))
    by (simp add: nat-add-distrib domain-block-nonnegative under)
  also have ... = 2^(nat (domain-block (l#ls)))
    using domain-block.simps by auto
  finally show ?thesis .
next
case cup
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 ... = 1*(2 ^ nat (domain-block ls))
  using rat-poly.row-length-def Cons cup by auto
moreover have ... = 2^(0 + nat (domain-block ls))
  using domain-block.simps by auto
moreover have ... = 2^(nat (domain l) + nat (domain-block ls))
  using domain.simps cup by auto
moreover have ... = 2^(nat (domain l + domain-block ls))
  using nat-sum cup domain.simps(2) nat-0 plus-int-code(2)
    plus-nat.add-0
  by (metis)
moreover have ... = 2^(nat (domain-block (l#ls)))
  using domain-block.simps by auto
ultimately show ?thesis by metis
next
case cap
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 ... = 4*(2 ^ nat (domain-block ls))
  using rat-poly.row-length-def Cons cap by auto
moreover have ... = 2^(2 + nat (domain-block ls))
  using domain-block.simps by auto
moreover have ... = 2^(nat (domain l) + nat (domain-block ls))
  using domain.simps cap by auto
moreover have ... = 2^(nat (domain l + domain-block ls))
  by (simp add: cap domain-block-nonnegative nat-add-distrib)
moreover have ... = 2^(nat (domain-block (l#ls)))
  using domain-block.simps by auto

```

ultimately show *?thesis* by *metis*
qed
qed

lemma *row-length-domain-block*:

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

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*) = $2^{\wedge} \text{nat} (\text{domain-block } ls)$
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 ... = $2 * (2^{\wedge} \text{nat} (\text{domain-block } ls))$

using *rat-poly.row-length-def Cons vert* **by** *auto*

moreover have ... = $2^{\wedge}(1 + \text{nat} (\text{domain-block } ls))$

using *domain-block.simps* **by** *auto*

moreover have ... = $2^{\wedge}(\text{nat} (\text{domain } l) + \text{nat} (\text{domain-block } ls))$

using *domain.simps vert* **by** *auto*

moreover have ... = $2^{\wedge}(\text{nat} (\text{domain } l + \text{domain-block } ls))$

using *Suc-eq-plus1-left Suc-nat-eq-nat-zadd1 calculation(4) domain.simps(1)*

domain-block-non-negative vert
by *metis*

moreover have ... = $2^{\wedge}(\text{nat} (\text{domain-block } (l\#ls)))$

using *domain-block.simps* **by** *auto*

ultimately show *?thesis* **by** *metis*

next

case *over*

have *rat-poly.row-length* (blockmat *ls*) = $2^{\wedge} \text{nat} (\text{domain-block } ls)$
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 ... = $4 * (2^{\wedge} \text{nat} (\text{domain-block } ls))$

using *rat-poly.row-length-def Cons over* **by** *auto*

moreover have ... = $2^{\wedge}(2 + \text{nat} (\text{domain-block } ls))$

using *domain-block.simps* **by** *auto*

moreover have ... = $2^{\wedge}(\text{nat} (\text{domain } l) + \text{nat} (\text{domain-block } ls))$

using *domain.simps over* **by** *auto*

```

moreover have ... = 2nat (domain l + domain-block ls)
  by (simp add: over domain-block-nonnegative nat-add-distrib)
moreover have ... = 2nat (domain-block (l#ls))
  using domain-block.simps by auto
ultimately show ?thesis by metis
next
case under
have rat-poly.row-length (blockmat ls) = 2nat (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 ... = 4*(2nat (domain-block ls))
  using rat-poly.row-length-def Cons under by auto
moreover have ... = 2(2 + nat (domain-block ls))
  using domain-block.simps by auto
moreover have ... = 2(nat (domain l) + nat (domain-block ls))
  using domain.simps under by auto
moreover have ... = 2(nat (domain l + domain-block ls))
  by (simp add: under domain-block-nonnegative nat-add-distrib)
moreover have ... = 2(nat (domain-block (l#ls))
  using domain-block.simps by auto
ultimately show ?thesis by metis
next
case cup
have rat-poly.row-length (blockmat ls) = 2nat (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 ... = 1*(2nat (domain-block ls))
  using rat-poly.row-length-def Cons cup by auto
moreover have ... = 2(0 + nat (domain-block ls))
  using domain-block.simps by auto
moreover have ... = 2(nat (domain l) + nat (domain-block ls))
  using domain.simps cup by auto
moreover have ... = 2(nat (domain l + domain-block ls))
  using nat-sum cup domain.simps(2)
    nat-0 plus-int-code(2) plus-nat.add-0 by (metis)
moreover have ... = 2(nat (domain-block (l#ls))
  using domain-block.simps by auto
ultimately show ?thesis by metis
next
case cap
have rat-poly.row-length (blockmat ls) = 2nat (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 ... = 4*(2 ^ nat (domain-block ls))
    using rat-poly.row-length-def Cons cap by auto
  moreover have ... = 2^(2 + nat (domain-block ls))
    using domain-block.simps by auto
  moreover have ... = 2^(nat (domain l) + nat (domain-block ls))
    using domain.simps cap by auto
  moreover have ... = 2^(nat (domain l + domain-block ls))
    by (simp add: cap domain-block-nonnegative nat-add-distrib)
  moreover have ... = 2^(nat (domain-block (l#ls)))
    using domain-block.simps by auto
  ultimately show ?thesis by metis
qed
qed

```

```

lemma length-codomain-block:length (blockmat ls)
      = (2 ^ (nat ((codomain-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 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 ... = 2*(2 ^ nat (codomain-block ls))
      using Cons vert by auto
    moreover have ... = 2^(1 + nat (codomain-block ls))
      by auto
    moreover have ... = 2^(nat (codomain l) + nat (codomain-block ls))
      using codomain.simps vert by auto
    moreover have ... = 2^(nat (codomain l + codomain-block ls))
      using nat-sum Suc-eq-plus1-left Suc-nat-eq-nat-zadd1
        codomain.simps(1) codomain-block-nonnegative nat-numeral
        numeral-One vert by (metis)
    moreover have ... = 2^(nat (codomain-block (l#ls)))
      by auto
    ultimately show ?thesis by metis
  next
  case over
    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 ... = 4*(2 ^ nat (codomain-block ls))
    using Cons over by auto
  moreover have ... = 2^(2 + nat (codomain-block ls))
    by auto
  moreover have ... = 2^(nat (codomain l) + nat (codomain-block ls))
    using codomain.simps over by auto
  moreover have ... = 2^(nat (codomain l + codomain-block ls))
    using nat-sum over codomain.simps codomain-block-nonnegative
    by auto
  moreover have ... = 2^(nat (codomain-block (l#ls)))
    by auto
  ultimately show ?thesis by metis
next
case under
  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 ... = 4*(2 ^ nat (codomain-block ls))
    using Cons under by auto
  moreover have ... = 2^(2 + nat (codomain-block ls))
    by auto
  moreover have ... = 2^(nat (codomain l) + nat (codomain-block ls))
    using codomain.simps under by auto
  moreover have ... = 2^(nat (codomain l + codomain-block ls))
    using nat-sum under codomain.simps codomain-block-nonnegative
    by auto
  moreover have ... = 2^(nat (codomain-block (l#ls)))
    by auto
  ultimately show ?thesis by metis
next
case cup
  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 ... = 4*(2 ^ nat (codomain-block ls))
    using Cons cup by auto
  moreover have ... = 2^(2 + nat (codomain-block ls))
    by auto
  moreover have ... = 2^(nat (codomain l) + nat (codomain-block ls))
    using codomain.simps cup by auto
  moreover have ... = 2^(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
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

```



```

moreover have rat-poly.row-length (blockmat (a#ls))
  = (rat-poly.row-length (brickmat a))
    *(rat-poly.row-length (blockmat ls))
  using calculation rat-poly.row-length-mat by (metis)
moreover have length (blockmat (a#ls))
  = (length (brickmat a))
    *(length (blockmat ls))
  using blockmat.simps(2) rat-poly.length-Tensor by (metis)
ultimately have Cons-3:mat
  (rat-poly.row-length (brickmat a))
  (length (brickmat a))
  (brickmat a)
   $\implies$  ?case
  using rat-poly.well-defined-Tensor Cons by auto
then show ?case
proof(cases a)
case vert
  have mat
    (rat-poly.row-length (brickmat a))
    (length (brickmat a))
    (brickmat a)
    using vert-dim mat-vert rat-poly.matrix-row-length vert
    by metis
  thus ?thesis using Cons-3 by auto
next
case over
  have mat
    (rat-poly.row-length (brickmat a))
    (length (brickmat a))
    (brickmat a)
    using mat-over rat-poly.matrix-row-length over
    by metis
  thus ?thesis using Cons-3 by auto
next
case under
  have mat
    (rat-poly.row-length (brickmat a))
    (length (brickmat a))
    (brickmat a)
    using mat-under rat-poly.matrix-row-length under by metis
  thus ?thesis using Cons-3 by auto
next
case cap
  have mat
    (rat-poly.row-length (brickmat a))
    (length (brickmat a))
    (brickmat a)
    using mat-cap rat-poly.matrix-row-length cap by metis
  thus ?thesis using Cons-3 by auto

```

```

next
case cup
  have mat
    (rat-poly.row-length (brickmat a))
    (length (brickmat a))
      (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 1×1 matrix whose entry is the Kauffman bracket.

```

primrec kauff-mat::wall  $\Rightarrow$  rat-poly mat
where
  kauff-mat (basic w) = (blockmat w)
  |kauff-mat (w*ws) = rat-poly.matrix-mult (blockmat w) (kauff-mat ws)

```

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

```

theorem matrix-kauff-mat:
  ((is-tangle-diagram ws)
   $\implies$  (rat-poly.row-length (kauff-mat ws)) =  $2^{\wedge}$ (nat (domain-wall ws))
   $\wedge$  (length (kauff-mat ws)) =  $2^{\wedge}$ (nat (codomain-wall ws))
   $\wedge$  (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))
  =  $2^{\wedge}(\text{nat } (\text{domain-wall } ws))$ 
using prod.hyps prod-1 by auto
moreover have prod-4:length (kauff-mat ws)
  =  $2^{\wedge}(\text{nat } (\text{codomain-wall } ws))$ 
using prod.hyps prod-1 by auto
moreover have prod-5:
  mat
  (rat-poly.row-length (blockmat w))
  (length (blockmat w))
  (blockmat w)
using matrix-blockmat by auto
moreover have prod-6:
  rat-poly.row-length (blockmat w)
  =  $2^{\wedge}(\text{nat } (\text{domain-block } w))$ 
  and length (blockmat w) =  $2^{\wedge}(\text{nat } (\text{codomain-block } w))$ 
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.matrix-mult (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 ... =  $2^{\wedge}(\text{nat } (\text{domain-wall } (w*ws)))$ 
using prod-6 domain-wall.simps by auto
ultimately have res2:
  rat-poly.row-length (kauff-mat (w*ws))
  =  $2^{\wedge}(\text{nat } (\text{domain-wall } (w*ws)))$ 
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 ... =  $2^{\wedge}(\text{nat } (\text{codomain-wall } (w*ws)))$ 
using prod-4 codomain-wall.simps(2) by auto

```

ultimately have $res3: length (kauff\text{-}mat (w*ws))$
 $= 2^{\wedge}(nat (codomain\text{-}wall (w*ws)))$
by *auto*
with *res1 res2 show ?case*
using $\langle length (kauff\text{-}mat ws) = 2^{\wedge} nat (codomain\text{-}wall (w * ws)) \rangle$
 $\langle rat\text{-}poly.\text{row-length} (blockmat w) = 2^{\wedge} nat (domain\text{-}wall (w * ws)) \rangle$
by (*metis*)
qed

theorem *effective-matrix-kauff-mat:*
assumes *is-tangle-diagram ws*
shows $(rat\text{-}poly.\text{row-length} (kauff\text{-}mat ws)) = 2^{\wedge}(nat (domain\text{-}wall ws))$
and $length (kauff\text{-}mat ws) = 2^{\wedge}(nat (codomain\text{-}wall ws))$
and $mat (rat\text{-}poly.\text{row-length} (kauff\text{-}mat ws)) (length (kauff\text{-}mat ws))$
 $(kauff\text{-}mat ws)$
apply (*auto simp add:matrix-kauff-mat assms*)
using *assms matrix-kauff-mat by metis*

lemma *mat-mult-equiv:*
 $rat\text{-}poly.\text{matrix-mult } m1 m2 = mat\text{-}mult (rat\text{-}poly.\text{row-length } m1) m1 m2$
by *auto*

theorem *associative-rat-poly-mat:*
assumes $mat (rat\text{-}poly.\text{row-length } m1) (rat\text{-}poly.\text{row-length } m2) m1$
and $mat (rat\text{-}poly.\text{row-length } m2) (rat\text{-}poly.\text{row-length } m3) m2$
and $mat (rat\text{-}poly.\text{row-length } m3) nc m3$
shows $rat\text{-}poly.\text{matrix-mult } m1 (rat\text{-}poly.\text{matrix-mult } m2 m3)$
 $= rat\text{-}poly.\text{matrix-mult} (rat\text{-}poly.\text{matrix-mult } m1 m2) m3$

proof–
have $(rat\text{-}poly.\text{matrix-mult } m2 m3)$
 $= mat\text{-}mult (rat\text{-}poly.\text{row-length } m2) m2 m3$
using *mat-mult-equiv by auto*
then have $rat\text{-}poly.\text{matrix-mult } m1 (rat\text{-}poly.\text{matrix-mult } m2 m3)$
 $= mat\text{-}mult (rat\text{-}poly.\text{row-length } m1) m1$
 $(mat\text{-}mult (rat\text{-}poly.\text{row-length } m2) m2 m3)$
using *mat-mult-equiv by auto*
moreover have $\dots = mat\text{-}mult (rat\text{-}poly.\text{row-length } m1)$
 $(mat\text{-}mult (rat\text{-}poly.\text{row-length } m1) m1 m2) m3$
using *assms mat-mult-assoc by metis*
moreover have $\dots = rat\text{-}poly.\text{matrix-mult} (rat\text{-}poly.\text{matrix-mult } m1 m2) m3$
proof–

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

```

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 1×1 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) ∧ (is-tangle-diagram ws2)
 ∧ (domain-wall ws2 = codomain-wall ws1))  $\implies$ 
kauff-mat (ws1 ∘ ws2) = rat-poly.matrix-mult (kauff-mat ws1) (kauff-mat ws2)
proof(induct ws1)
case (basic w1)
  have (basic w1) ∘ (ws2) = (w1)*(ws2)
    using compose.simps by auto
  moreover have kauff-mat ((basic w1) ∘ ws2) =rat-poly.matrix-mult (blockmat
w1) (kauff-mat ws2)
    using kauff-mat.simps(2) by auto
  then show ?case using kauff-mat.simps(1) by auto
next
case (prod w1 ws1)
  have 1:is-tangle-diagram (w1*ws1)
    using prod.prem by (rule conjE)
  then have 2:(is-tangle-diagram ws1)
    ∧ (codomain-block w1 = domain-wall ws1)
    using is-tangle-diagram.simps(2) by metis
  then have
    mat (2nat (domain-wall ws1)) (2nat (codomain-wall ws1)) (kauff-mat
ws1)
    and mat (2nat (domain-block w1)) (2nat (codomain-block w1)) (2nat (codomain-wall ws1))
(blockmat w1)
    using effective-matrix-kauff-mat matrix-blockmat length-codomain-block

```

```

      row-length-domain-block
    by (auto) (metis)
  with 2 have 3:mat
      (rat-poly.row-length (blockmat w1))
      (2nat (domain-wall ws1))
      (blockmat w1)

    and mat
      (2nat (domain-wall ws1))
      (2nat (domain-wall ws2))
      (kauff-mat ws1)

    and (2nat (domain-wall ws1))
      = (rat-poly.row-length (kauff-mat ws1))
  using effective-matrix-kauff-mat prod.premis matrix-blockmat
      row-length-domain-block by auto
  then have mat
      (rat-poly.row-length (blockmat w1))
      (rat-poly.row-length (kauff-mat ws1))
      (blockmat w1)

    and mat
      (rat-poly.row-length (kauff-mat ws1))
      (2nat (domain-wall ws2))
      (kauff-mat ws1)

    by auto
  moreover have mat
      (2nat (domain-wall ws2))
      (2nat (codomain-wall ws2))
      (kauff-mat ws2)

    and (2nat (domain-wall ws2))
      = rat-poly.row-length (kauff-mat ws2)
  using prod.premis effective-matrix-kauff-mat
      effective-matrix-kauff-mat
  by (auto) (metis prod.premis)
  ultimately have mat
      (rat-poly.row-length (blockmat w1))
      (rat-poly.row-length (kauff-mat ws1))
      (blockmat w1)

    and mat
      (rat-poly.row-length (kauff-mat ws1))
      (rat-poly.row-length (kauff-mat ws2))
      (kauff-mat ws1)

    and mat
      (rat-poly.row-length (kauff-mat ws2))
      (2nat (codomain-wall ws2))
      (kauff-mat ws2)

    by auto
  with 3 have rat-poly.matrix-mult
      (blockmat w1)
      (rat-poly.matrix-mult (kauff-mat ws1)
        (kauff-mat ws2))

```

```

      = rat-poly.matrix-mult
        (rat-poly.matrix-mult
          (blockmat w1)
          (kauff-mat ws1))
          (kauff-mat ws2)
    using associative-rat-poly-mat by auto
  then show ?case
    using 2 codomain-wall.simps(2) compose-Cons
    prod.hyps prod.premis kauff-mat.simps(2) by (metis)
qed

theorem left-mat-compose:
  assumes is-tangle-diagram ws
    and codomain-wall ws = 0
  shows kauff-mat ws = (kauff-mat (ws ∘ (basic [])))
proof -
  have mat (rat-poly.row-length (kauff-mat ws)) 1 (kauff-mat ws)
    using effective-matrix-kauff-mat assms nat-0 power-0 by metis
  moreover have (kauff-mat (basic [])) = mat1 1
    using kauff-mat.simps(1) blockmat.simps(1) mat1-equiv by auto
  moreover then have 1:(kauff-mat (ws ∘ (basic [])))
    = rat-poly.matrix-mult
      (kauff-mat ws)
      (kauff-mat (basic []))
    using tangle-compose-matrix assms is-tangle-diagram.simps by auto
  ultimately have rat-poly.matrix-mult (kauff-mat ws) (kauff-mat (basic []))
    = (kauff-mat ws)
    using mat-mult-equiv mat1-mult-right by auto
  then show ?thesis using 1 by auto
qed

theorem right-mat-compose:
  assumes is-tangle-diagram ws and domain-wall ws = 0
  shows kauff-mat ws = (kauff-mat ((basic []) ∘ ws))
proof -
  have mat 1 (length (kauff-mat ws)) (kauff-mat ws)
    using effective-matrix-kauff-mat assms nat-0 power-0 by metis
  moreover have (kauff-mat (basic [])) = mat1 1
    using kauff-mat.simps(1) blockmat.simps(1) mat1-equiv by auto
  moreover then have 1:(kauff-mat ((basic []) ∘ ws))
    = rat-poly.matrix-mult
      (kauff-mat (basic []))
      (kauff-mat ws)
    using tangle-compose-matrix assms is-tangle-diagram.simps by auto
  ultimately have rat-poly.matrix-mult (kauff-mat (basic [])) (kauff-mat ws)
    = (kauff-mat ws)
    using effective-matrix-kauff-mat(3) is-tangle-diagram.simps(1)
    mat1 mat1-mult-left one-neq-zero rat-poly.mat-empty-column-length

```

$\text{rat-poly.unique-row-col}(1)$
by *metis*
then show *?thesis* **using** 1 **by** *auto*
qed

lemma *left-id-blockmat:blockmat* $\square \otimes \text{blockmat } b = \text{blockmat } b$
unfolding *blockmat.simps(1)* *rat-poly.Tensor-left-id* **by** *auto*

lemma *tens-assoc*:

$\forall a \text{ } xs \text{ } ys. (\text{brickmat } a \otimes (\text{blockmat } xs \otimes \text{blockmat } ys))$
 $= (\text{brickmat } a \otimes \text{blockmat } xs) \otimes \text{blockmat } ys$

proof –

have $\forall a. (\text{mat}$
 $\quad (\text{rat-poly.row-length } (\text{brickmat } a))$
 $\quad (\text{length } (\text{brickmat } a))$
 $\quad (\text{brickmat } a))$
using *brickmat.simps*
unfolding *mat-def rat-poly.row-length-def Ball-def vec-def*
apply(*auto*)
by (*case-tac a*) (*auto*)

moreover have $\forall xs. (\text{mat}$
 $\quad (\text{rat-poly.row-length } (\text{blockmat } xs))$
 $\quad (\text{length } (\text{blockmat } xs))$
 $\quad (\text{blockmat } xs))$

using *matrix-blockmat* **by** *auto*

moreover have $\forall ys. \text{mat}$
 $\quad (\text{rat-poly.row-length } (\text{blockmat } ys))$
 $\quad (\text{length } (\text{blockmat } ys))$
 $\quad (\text{blockmat } ys)$

using *matrix-blockmat* **by** *auto*

ultimately show *?thesis* **using** *rat-poly.associativity* **by** *auto*
qed

lemma *kauff-mat-tensor-distrib*:

$\forall xs. \forall ys. (\text{kauff-mat } (\text{basic } xs \otimes \text{basic } ys))$
 $= \text{kauff-mat } (\text{basic } xs) \otimes \text{kauff-mat } (\text{basic } ys)$
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*:

$(\text{blockmat } (a \otimes b)) = (\text{blockmat } a) \otimes (\text{blockmat } b)$

proof –

have $\text{blockmat } (a \otimes b) = \text{kauff-mat } (\text{basic } (a \otimes b))$


```

    using kauff-mat.simps(1) by auto
  moreover have ... = kauff-mat (basic a)  $\otimes$  kauff-mat (basic b)
    using kauff-mat-tensor-distrib by auto
  moreover have ... = (blockmat a)  $\otimes$  (blockmat b)
    using kauff-mat.simps(1) by auto
  ultimately show ?thesis by auto
qed

```

```

lemma blockmat-non-empty:  $\forall$  bs. (blockmat bs  $\neq$  [])
  apply (rule allI)
  apply (induct-tac bs)
  apply (auto)
  apply (case-tac a)
  apply (auto)
  apply (metis length-0-conv rat-poly.vec-mat-Tensor-length)
  apply (metis length-0-conv rat-poly.vec-mat-Tensor-length)
  apply (metis length-0-conv rat-poly.vec-mat-Tensor-length)
  apply (metis length-0-conv rat-poly.vec-mat-Tensor-length)
  apply (metis length-0-conv rat-poly.vec-mat-Tensor-length)
done

```

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  $\neq$  []
proof -
  have (length (kauff-mat ws) =  $2^{\wedge}(\text{nat}(\text{codomain-wall ws}))$ )
    using effective-matrix-kauff-mat assms by auto
  then have (length (kauff-mat ws)  $\geq$  1)
    by auto
  then show ?thesis by auto
qed

```

```

lemma is-tangle-diagram-length-rowlength:
  assumes is-tangle-diagram (w*ws)
  shows length (blockmat w) = rat-poly.row-length (kauff-mat ws)
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)
    =  $2^{\wedge}(\text{nat}(\text{domain-wall ws}))$ 
    using effective-matrix-kauff-mat by (metis assms is-tangle-diagram.simps(2))
  moreover have length (blockmat w)
    =  $2^{\wedge}(\text{nat}(\text{codomain-block w}))$ 
    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  $0: \text{blockmat } w1 = []$ 
              show False using  $0$ 
                by (metis blockmat-non-empty)
            next
              assume  $1: \text{kauff-mat } ws1 = []$ 
              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  $0: \text{blockmat } w2 = []$ 
              show False using  $0$ 
                by (metis blockmat-non-empty)
            next
              assume  $1: \text{kauff-mat } ws2 = []$ 

```

```

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  $\Rightarrow$  rat-poly mat
where
make-vert-equiv 0 = [[1]]
|make-vert-equiv (Suc k) = ((mat1 2)  $\otimes$  (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:( $\forall i < 2. \forall j < 2.$ 
    ((?r)!i!j = (if i = j then 1 else 0)))

```

```

proof-
  have 1:(?r ! 0! 0) = 1
    by auto
  moreover have 2:(?r ! 0! 1) = 0
    by auto
  moreover have 3:(?r ! 1! 0) = 0
    by auto
  moreover have 5:(?r ! 1! 1) = 1
    by auto
  ultimately show ?thesis
    by (auto dest!: less-2-cases)
qed
have 3:mat 2 2 (mat1 2)
  by (metis mat1)
have 4:( $\forall i < 2. \forall j < 2.$  ((?l)!i!j = (if i = j then 1 else 0)))
  by (metis mat1-index)

```

then have $(\forall i < 2. \forall j < 2. ((?l) ! i ! j = (?r ! i ! j)))$
using 2 **by** *auto*
with 1 3 **have** $?l = ?r$
by (*metis mat-eqI*)
then show $?thesis$ **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 $rat-poly.row-length (make-vert-equiv n) = 2^{\hat{n}}$
and $length (make-vert-equiv n) = 2^{\hat{n}}$
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 $rat-poly.row-length (make-vert-equiv n) = 2^{\hat{n}}$
and $length (make-vert-equiv n) = 2^{\hat{n}}$
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 $nat-mult::nat \Rightarrow nat \Rightarrow nat$ (**infixl** $*n$ 65)
where
 $nat-mult a b \equiv ((a::nat)*b)$

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 $j = a*(i \text{ div } a) + (i \text{ mod } a)$
using *assms* **by** *auto*
then show *?thesis* **by** *auto*
qed

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

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

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

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

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

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

lemma $((P \wedge Q) \longrightarrow R) \implies (P \longrightarrow Q \longrightarrow R)$
by *auto*

lemma *length* (*mat1* 2) = 2
apply (*simp add:mat1I-def*)
done

theorem *make-vert-equiv-mat*:
make-vert-equiv n = (*mat1* (2ⁿ))
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))!j!i)
    = 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.Tensor (mat1 2) (mat1 (2^k))!j!i)
    = 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))

```

```

      *n(rat-poly.row-length (mat1 (2^k)))
        = 2^(k+1)
    using 3 6 by auto
  moreover have (length (mat1 2))
    *n(length (mat1 (2^k)))
      = 2^(k+1)
    using 4 7 by (metis 3 6 calculation(2))
  ultimately show ?thesis by metis
qed
have 10:∀ i j.((i div (rat-poly.row-length (mat1 (2^k))) < 2)
  ∧(j div length (mat1 (2^k)) < 2)
  → (((mat1 2)!(j div (length (mat1 (2^k))))
    !(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 mat1-index by (metis 6 7)
have 11:∀ j.(j < (2^(k+1)) → j div (length (mat1 (2^k))) < 2)
proof-
  have 2^(k+1) = (2 *n (2^k))
    by auto
  then show ?thesis
    using 7 allI Suc.IH prop-make-vert-equiv(1)
    rat-poly.div-left-ineq by (metis)

qed
moreover have 12:
  ∀ i.(i < (2^(k+1))
  → (i div (rat-poly.row-length (mat1 (2^k)))) < 2)
proof-
  have 2^(k+1) = (2 *n (2^k))
    by auto
  then show ?thesis using 7 allI by (metis Suc.IH prop-make-vert-equiv(1)
rat-poly.div-left-ineq)
qed
ultimately have 13:
  ∀ i j.((i < (2^(k+1))) ∧ j < (2^(k+1))) →
    ((i div (rat-poly.row-length (mat1 (2^k)))) < 2)
    ∧((j div (length (mat1 (2^k)))) < 2)

  by auto
have 14:∀ i j.(i < (2^(k+1))) ∧ (j < (2^(k+1))) →
  (((mat1 2)
  !(j div (length (mat1 (2^k))))
  !(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))
apply(rule allI)
apply(rule allI)
proof
  fix i j
  assume 0:(i::nat) < 2 ^ (k + 1) ^ (j::nat) < 2 ^ (k + 1)
  have ((i div (rat-poly.row-length (mat1 (2^k)))) < 2)
    ^ ((j div (length (mat1 (2^k)))) < 2)
    using 0 13 by auto
  then show (((mat1 2)
    !(j div (length (mat1 (2^k))))
    !(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:∀ i j.((i mod (rat-poly.row-length (mat1 (2^k)))) < 2^k)
  ^ (j mod length (mat1 (2^k)) < 2^k)
  → (((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)))
    using mat1-index by (metis 6 7)
have 16:∀ j.(j < (2^(k+1))) → j mod (length (mat1 (2^k))) < 2^k)
proof-
  have 2^(k+1) = (2 * 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:∀ i.(i < (2^(k+1)))
  → (i mod (rat-poly.row-length (mat1 (2^k)))) < 2^k)
proof-
  have 2^(k+1) = (2 * n (2^k))
    by auto
  then show ?thesis using 7 allI by (metis 6 calculation)
qed
ultimately have 18:
  ∀ i j.((i < (2^(k+1))) ^ (j < (2^(k+1)))) →
    ((i mod (rat-poly.row-length (mat1 (2^k)))) < 2^k)
    ^ ((j mod (length (mat1 (2^k)))) < 2^k)

```



```

by (metis 7)
have 19:  $\forall i j. (i < (2^{k+1})) \wedge (j < (2^{k+1})) \longrightarrow$ 
  (((mat1 (2k))
    !(j mod (length (mat1 (2k))))
    !(i mod (rat-poly.row-length (mat1 (2k))))))
  = (if
    ((j mod (length (mat1 (2k))))
     = (i mod (rat-poly.row-length (mat1 (2k))))))
    then 1
    else 0))
apply(rule allI)
apply(rule allI)
proof
fix i j
assume 0:  $(i::nat) < 2^{k+1} \wedge (j::nat) < 2^{k+1}$ 
have ((i mod (rat-poly.row-length (mat1 (2k)))) < 2k)
   $\wedge$  ((j mod (length (mat1 (2k)))) < 2k)
  using 0 18 by auto
then show (((mat1 (2k))
    !(j mod (length (mat1 (2k))))
    !(i mod (rat-poly.row-length (mat1 (2k))))))
  = (if
    ((j mod (length (mat1 (2k))))
     = (i mod (rat-poly.row-length (mat1 (2k))))))
    then 1
    else 0))
  using 15 by (metis 6)
qed
have ( $\forall i. \forall j.$ 
   $(i < (2^{k+1})) \wedge (j < (2^{k+1}))$ 
   $\longrightarrow$  rat-poly-times
    ((mat1 2)
      !(j div (length (mat1 (2k))))
      !(i div (rat-poly.row-length (mat1 (2k))))))
    ((mat1 (2k))
      !(j mod length (mat1 (2k)))
      !(i mod (rat-poly.row-length (mat1 (2k))))))
  =
  (if
    (((j div (length (mat1 (2k))))
      = (i div (rat-poly.row-length (mat1 (2k))))))
      $\wedge$  ((j mod (length (mat1 (2k))))
      = (i mod (rat-poly.row-length (mat1 (2k))))))
    then 1
    else 0))
apply(rule allI)
apply(rule allI)
proof
fix i j

```

```

assume 0: ((i::nat) < (2^(k+1))) ∧ ((j::nat) < (2^(k+1)))
have s1: ((mat1 2)
  !(j div (length (mat1 (2^k))))
  !(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 0 14 by metis
moreover have s2: ((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)
  using 0 19 by metis
show 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))))
    ∧ ((j mod (length (mat1 (2^k))))
     = (i mod (rat-poly.row-length (mat1 (2^k))))
    then 1
    else 0)

  apply(simp)
  apply(rule conjI)
  proof–
  show j div length (mat1 (2 ^ k)) = i div rat-poly.row-length (mat1 (2 ^ k))
    ∧ (j mod length (mat1 (2 ^ k)) = i mod rat-poly.row-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))))
  = 1
  proof–

```

have
 $j \text{ div length } (mat1 (2 \wedge k))$
 $\quad = i \text{ div rat-poly.row-length } (mat1 (2 \wedge k))$
 $\wedge j \text{ mod length } (mat1 (2 \wedge k)) = i \text{ mod rat-poly.row-length } (mat1 (2 \wedge k))$
 $\implies \text{rat-poly-times}$
 $(mat1 2 ! (j \text{ div length } (mat1 (2 \wedge k))) ! (i \text{ div rat-poly.row-length } (mat1 (2 \wedge k))))$
 $(mat1 (2 \wedge k) ! (j \text{ mod length } (mat1 (2 \wedge k)))$
 $\quad ! (i \text{ mod rat-poly.row-length } (mat1 (2 \wedge k)))) = 1$
proof-
assume local-assms:
 $j \text{ div length } (mat1 (2 \wedge k)) = i \text{ div rat-poly.row-length } (mat1 (2 \wedge k))$
 $\wedge j \text{ mod length } (mat1 (2 \wedge k)) = i \text{ mod rat-poly.row-length } (mat1 (2 \wedge k))$
have $(mat1 2 ! (j \text{ div length } (mat1 (2 \wedge k))) ! (i \text{ div rat-poly.row-length } (mat1 (2 \wedge k))))$
 $\quad = 1$
using s1 local-assms by metis
moreover have $(mat1 (2 \wedge k)$
 $\quad ! (j \text{ mod length } (mat1 (2 \wedge k))) ! (i \text{ mod rat-poly.row-length } (mat1 (2$
 $\wedge k)))) = 1$
using s2 local-assms by metis
ultimately show ?thesis
by $(metis 3 6 7 \text{ Suc.IH local-assms mve1 prop-make-vert-equiv}(1)$
 $\quad \text{prop-make-vert-equiv}(2) \text{ rat-poly.right-id})$
qed
then show ?thesis by auto
qed
show
 $(j \text{ div length } (mat1 (2 \wedge k)) = i \text{ div rat-poly.row-length } (mat1 (2 \wedge k)) \longrightarrow$
 $\quad j \text{ mod length } (mat1 (2 \wedge k)) \neq i \text{ mod rat-poly.row-length } (mat1 (2 \wedge$
 $k))) \longrightarrow$
 $\quad mat1 2 ! (j \text{ div length } (mat1 (2 \wedge k))) ! (i \text{ div rat-poly.row-length } (mat1 (2$
 $\wedge k))) = 0 \vee$
 $\quad mat1 (2 \wedge k) ! (j \text{ mod length } (mat1 (2 \wedge k))) ! (i \text{ mod rat-poly.row-length } (mat1 (2 \wedge k))) = 0$
proof-
have $(j \text{ div length } (mat1 (2 \wedge k)) = i \text{ div rat-poly.row-length } (mat1 (2 \wedge k))$
 $\quad \wedge j \text{ mod length } (mat1 (2 \wedge k)) \neq i \text{ mod rat-poly.row-length } (mat1 (2 \wedge$
 $k))) \implies$
 $\quad mat1 2 ! (j \text{ div length } (mat1 (2 \wedge k))) ! (i \text{ div rat-poly.row-length } (mat1 (2 \wedge k))) = 0$
 $\quad \vee mat1 (2 \wedge k) ! (j \text{ mod length } (mat1 (2 \wedge k))) ! (i \text{ mod rat-poly.row-length } (mat1 (2 \wedge k))) = 0$
proof-
assume local-assms:
 $(j \text{ div length } (mat1 (2 \wedge k)) = i \text{ div rat-poly.row-length } (mat1 (2 \wedge k))$
 $\quad \wedge j \text{ mod length } (mat1 (2 \wedge k)) \neq i \text{ mod rat-poly.row-length } (mat1 (2$

$\wedge k)))$
have $mat1 (2 \wedge k) ! (j \text{ mod } length (mat1 (2 \wedge k))) ! (i \text{ mod } rat\text{-}poly.\text{row-length} (mat1 (2 \wedge k))) = 0$
using $s2 \text{ local-assms}$ **by** $metis$
then show $?thesis$ **by** $auto$
qed
then have l :
 $(j \text{ div } length (mat1 (2 \wedge k)) = i \text{ div } rat\text{-}poly.\text{row-length} (mat1 (2 \wedge k))$
 $\wedge j \text{ mod } length (mat1 (2 \wedge k)) \neq i \text{ mod } rat\text{-}poly.\text{row-length} (mat1 (2 \wedge k)))$
 \longrightarrow
 $mat1 2 ! (j \text{ div } length (mat1 (2 \wedge k))) ! (i \text{ div } rat\text{-}poly.\text{row-length} (mat1 (2 \wedge k))) = 0$
 $\vee mat1 (2 \wedge k) ! (j \text{ mod } length (mat1 (2 \wedge k))) ! (i \text{ mod } rat\text{-}poly.\text{row-length} (mat1 (2 \wedge k))) = 0$
by $auto$
show $(j \text{ div } length (mat1 (2 \wedge k)) = i \text{ div } rat\text{-}poly.\text{row-length} (mat1 (2 \wedge k)))$
 \longrightarrow
 $j \text{ mod } length (mat1 (2 \wedge k)) \neq i \text{ mod } rat\text{-}poly.\text{row-length} (mat1 (2 \wedge k)) \longrightarrow$
 $mat1 2 ! (j \text{ div } length (mat1 (2 \wedge k))) ! (i \text{ div } rat\text{-}poly.\text{row-length} (mat1 (2 \wedge k))) = 0 \vee$
 $mat1 (2 \wedge k) ! (j \text{ mod } length (mat1 (2 \wedge k))) ! (i \text{ mod } rat\text{-}poly.\text{row-length} (mat1 (2 \wedge k))) = 0$
proof–
have
 $(j \text{ div } length (mat1 (2 \wedge k)) = i \text{ div } rat\text{-}poly.\text{row-length} (mat1 (2 \wedge k)) \longrightarrow$
 $j \text{ mod } length (mat1 (2 \wedge k)) \neq i \text{ mod } rat\text{-}poly.\text{row-length} (mat1 (2 \wedge k)))$
 \implies
 $mat1 2 ! (j \text{ div } length (mat1 (2 \wedge k))) ! (i \text{ div } rat\text{-}poly.\text{row-length} (mat1 (2 \wedge k))) = 0 \vee$
 $mat1 (2 \wedge k) ! (j \text{ mod } length (mat1 (2 \wedge k))) ! (i \text{ mod } rat\text{-}poly.\text{row-length} (mat1 (2 \wedge k))) = 0$
proof–
assume $local\text{-}assm1$:
 $(j \text{ div } length (mat1 (2 \wedge k)) = i \text{ div } rat\text{-}poly.\text{row-length} (mat1 (2 \wedge k)) \longrightarrow$
 $j \text{ mod } length (mat1 (2 \wedge k)) \neq i \text{ mod } rat\text{-}poly.\text{row-length} (mat1 (2 \wedge k)))$
have $(j \text{ div } length (mat1 (2 \wedge k)) = i \text{ div } rat\text{-}poly.\text{row-length} (mat1 (2 \wedge k)))$
 \implies
 $mat1 (2 \wedge k) ! (j \text{ mod } length (mat1 (2 \wedge k)))$
 $\quad ! (i \text{ mod } rat\text{-}poly.\text{row-length} (mat1 (2 \wedge k)))$
 $= 0$
using $s2 \text{ local-assm1}$ **by** $(metis \ 7)$
then have $l1$: $(j \text{ div } length (mat1 (2 \wedge k)) = i \text{ div } rat\text{-}poly.\text{row-length} (mat1 (2 \wedge k)))$
 $\implies ?thesis$
by $auto$
moreover have $\neg(j \text{ div } length (mat1 (2 \wedge k)) = i \text{ div } rat\text{-}poly.\text{row-length} (mat1 (2 \wedge k)))$

$\implies \text{mat1 } 2 ! (j \text{ div length } (\text{mat1 } (2^k)))$
 $\quad ! (i \text{ div rat-poly.row-length } (\text{mat1 } (2^k)))$
 $\quad = 0$

using *s1* **by** *metis*
then have $\neg(j \text{ div length } (\text{mat1 } (2^k)) = i \text{ div rat-poly.row-length } (\text{mat1 } (2^k)))$
 $\implies ?thesis$
by *auto*
then show *?thesis* **using** *l1* **by** *auto*
qed
then show *?thesis* **by** *auto*
qed
qed
qed
then have $(\forall i. \forall j. (i < (2^{k+1})) \wedge (j < (2^{k+1}))) \longrightarrow$
 $((\text{rat-poly.Tensor } (\text{mat1 } 2) (\text{mat1 } (2^k))!j!i) = (\text{if}$
 $((j \text{ div } (\text{length } (\text{mat1 } (2^k)))) = (i \text{ div } (\text{rat-poly.row-length } (\text{mat1 } (2^k))))$
 $\wedge (j \text{ mod } (\text{length } (\text{mat1 } (2^k)))) = (i \text{ mod } (\text{rat-poly.row-length } (\text{mat1 } (2^k))))$
 $\text{then } 1$
 $\text{else } 0)))$
using *9* **by** *metis*
then have $(\forall i. \forall j. (i < (2^{k+1})) \wedge (j < (2^{k+1}))) \longrightarrow$
 $((\text{rat-poly.Tensor } (\text{mat1 } 2) (\text{mat1 } (2^k))!j!i) = (\text{if}$
 $((j \text{ div } (2^k)) = (i \text{ div } (2^k)))$
 $\wedge (j \text{ mod } (2^k) = (i \text{ mod } (2^k)))$
 $\text{then } 1$
 $\text{else } 0)))$
by (*metis* (*opaque-lifting*, *no-types*) *6 7*)
then have $20: (\forall i. \forall j. (i < (2^{k+1})) \wedge (j < (2^{k+1}))) \longrightarrow$
 $((\text{rat-poly.Tensor } (\text{mat1 } 2) (\text{mat1 } (2^k))!j!i) = (\text{if } (j = i)$
 $\text{then } 1$
 $\text{else } 0)))$
using *equal-div-mod2* **by** *auto*
with *2* **have** $(\forall i. \forall j. (i < (2^{k+1})) \wedge (j < (2^{k+1}))) \longrightarrow$
 $((\text{rat-poly.Tensor } (\text{mat1 } 2) (\text{mat1 } (2^k))!j!i) = (\text{mat1 } (2^{k+1}))!j!i)$
by *metis*
then have $(\forall i < (2^{k+1}). \forall j < (2^{k+1}).$
 $((\text{rat-poly.Tensor } (\text{mat1 } 2) (\text{mat1 } (2^k))!j!i) = (\text{mat1 } (2^{k+1}))!j!i)$
by *auto*
moreover have $\text{mat } (2^{k+1}) (2^{k+1}) (\text{rat-poly.Tensor } (\text{mat1 } 2) (\text{mat1 } (2^k)))$
using $\langle \text{make-vert-equiv } (k + 1) = \text{mat1 } 2 \otimes \text{mat1 } (2^k) \rangle$
by (*metis* *prop-make-vert-equiv*(*1*) *prop-make-vert-equiv*(*2*)
prop-make-vert-equiv(*3*))
ultimately have $(\text{rat-poly.Tensor } (\text{mat1 } 2) (\text{mat1 } (2^k))) = (\text{mat1 } (2^{k+1}))$

```

      using 1 mat-eq1 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*::
 $rat-poly\ mat \Rightarrow rat-poly\ mat \Rightarrow rat-poly\ mat \Rightarrow bool$

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

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

proof –
assume *assms:weak-matrix-match* $A1\ A2\ B1$
have $1:\text{length } B1 = 1$
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 $\text{mat } 1\ 1\ [[1]]$
unfolding *mat-def Ball-def vec-def* **by** *auto*
moreover have $\text{rat-poly.row-length } [[1]] = \text{length } B1$
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 $((\text{rat-poly.matrix-mult } A1\ A2) \otimes (\text{rat-poly.matrix-mult } B1\ [[1]]))$
 $= (\text{rat-poly.matrix-mult } (A1 \otimes B1) (A2 \otimes [[1]]))$
using *rat-poly.distributivity* **by** *auto*
moreover have $(\text{rat-poly.matrix-mult } B1\ [[1]]) = B1$
using *weak-matrix-match-def assms mat1-equiv mat1-mult-right*
by (*metis*)
moreover have $(A2 \otimes [[1]]) = A2$
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
 $\text{weak-matrix-match2 } A1\ B1\ B2 \equiv (\text{mat } (\text{rat-poly.row-length } A1)\ 1\ A1)$
 $\wedge (\text{mat } (\text{rat-poly.row-length } B1)\ (\text{length } B1)\ B1)$
 $\wedge (\text{mat } (\text{rat-poly.row-length } B2)\ (\text{length } B2)\ B2)$
 $\wedge (A1 \neq []) \wedge (B1 \neq []) \wedge (B2 \neq [])$
 $\wedge (\text{length } B1 = \text{rat-poly.row-length } B2)$

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

proof –
assume *assms:weak-matrix-match2* $A1\ B1\ B2$
have $1:\text{length } A1 = 1$
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  $[[1]]$ 
    unfolding mat-def Ball-def vec-def by auto
  moreover have rat-poly.row-length  $[[1]] = \text{length } A1$ 
    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
      1 blockmat.simps(1) matrix-blockmat
    by (metis (opaque-lifting, no-types))
  then have  $((\text{rat-poly.matrix-mult } A1 \ [[1]]) \otimes (\text{rat-poly.matrix-mult } B1 \ B2))$ 
    =  $(\text{rat-poly.matrix-mult } (A1 \otimes B1) \ ([[1]] \otimes B2))$ 
    using rat-poly.distributivity by auto
  moreover have  $(\text{rat-poly.matrix-mult } A1 \ [[1]]) = A1$ 
    using weak-matrix-match2-def
      assms mat1-equiv mat1-mult-right
    by (metis)
  moreover have  $([[1]] \otimes B2) = B2$ 
    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
       $(\text{rat-poly.row-length } (\text{blockmat } w1))$ 
       $(\text{length } (\text{blockmat } w1))$ 
      (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
       $(\text{rat-poly.row-length } (\text{kauff-mat } ws1))$ 
       $(\text{length } (\text{kauff-mat } ws1))$ 
      (kauff-mat  $ws1$ )
    using matrix-kauff-mat by metis
  next
  have mat
       $(\text{rat-poly.row-length } (\text{blockmat } w2))$ 
       $(\text{length } (\text{blockmat } w2))$ 
      (blockmat  $w2$ )
    using matrix-blockmat by auto

```



```

then have mat
  (rat-poly.row-length (blockmat w2)) 1 (blockmat w2)
  using assms(2) length-codomain-block by auto
then show mat (rat-poly.row-length (blockmat w2)) (Suc 0) (blockmat w2)
  by auto
next
show length (blockmat w1) = rat-poly.row-length (kauff-mat ws1)
  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 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)
qed

```

```

lemma is-tangle-diagram-weak-matrix-match2:
assumes is-tangle-diagram (w2*ws2)
  and codomain-block w1 = 0
shows weak-matrix-match2 (blockmat w1) (blockmat w2) (kauff-mat ws2)
unfolding weak-matrix-match2-def
apply(auto)
proof–
  have mat
    (rat-poly.row-length (blockmat w1))
    (length (blockmat w1))
    (blockmat w1)
  using matrix-blockmat by auto
then have mat
    (rat-poly.row-length (blockmat w1)) 1 (blockmat w1)
  using assms(2) length-codomain-block by auto
then show mat (rat-poly.row-length (blockmat w1)) (Suc 0) (blockmat w1)
  by auto
next
have is-tangle-diagram ws2
  using assms(1) 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 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 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 ⊗ b * bs)

```

```

      = kauff-mat (a * as) ⊗ kauff-mat (b * bs)
have 0:rat-poly.matrix-match
      (blockmat a)
      (kauff-mat as)
      (blockmat b)
      (kauff-mat bs)
      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 * as) ⊗ (b * bs))
      = kauff-mat ((a ⊗ b) * (as ⊗ bs))
      using tensor.simps by auto
moreover have ... = rat-poly.matrix-mult
      (blockmat (a ⊗ b))
      (kauff-mat (as ⊗ bs))
      using kauff-mat.simps(2) by auto
moreover have ... = rat-poly.matrix-mult
      ((blockmat a) ⊗ (blockmat b))
      ((kauff-mat as) ⊗ (kauff-mat bs))
      using hyps 1 kauff-mat-tensor-distrib by auto
moreover have ... = (rat-poly.matrix-mult (blockmat a) (kauff-mat as))
      ⊗ (rat-poly.matrix-mult (blockmat b) (kauff-mat bs))
      using 0 rat-poly.distributivity by auto
moreover have ... = kauff-mat (a*as) ⊗ kauff-mat (b*bs)
      by auto
ultimately show ?case by metis
next
fix a b as bs
assume hyps:codomain-block b ≠ 0
      ⇒ is-tangle-diagram as
      ∧ is-tangle-diagram
      (basic (make-vert-block (nat (codomain-block b))))
      ⇒ kauff-mat
      (as ⊗ basic (make-vert-block (nat (codomain-block b))))
      = kauff-mat as
      ⊗ kauff-mat
      (basic (make-vert-block (nat (codomain-block b))))

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

let ?case = kauff-mat (a * as ⊗ basic b)
      = kauff-mat (a * as) ⊗ kauff-mat (basic b)
show ?case
proof(cases codomain-block b = 0)
case True
      have ((a * as) ⊗ (basic b)) = ((a ⊗ b) * as)
      using tensor.simps True by auto
      then have kauff-mat ((a * as) ⊗ (basic b))
      = kauff-mat ((a ⊗ b) * 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 T1:
    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 T1 by metis
next
case False
let ?bs = (basic (make-vert-block (nat (codomain-block b))))
have F0: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 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

```

$(as \otimes basic \ (make\text{-}vert\text{-}block \ (nat \ (codomain\text{-}block \ b)))) =$
 $kauff\text{-}mat \ as \ \otimes \ kauff\text{-}mat \ ?bs$

using *F1 hyps* **by** *auto*
moreover have $((a * as) \ \otimes \ (basic \ b)) = (a \ \otimes \ b) * (as \ \otimes \ ?bs)$
using *False tensor.simps* **by** *auto*
moreover then have $kauff\text{-}mat \ ((a * as) \ \otimes \ (basic \ b))$
 $= kauff\text{-}mat((a \ \otimes \ b) * (as \ \otimes \ ?bs))$

by *auto*
moreover then have $\dots = rat\text{-}poly.matrix\text{-}mult$
 $(blockmat \ (a \ \otimes \ b))$
 $(kauff\text{-}mat \ (as \ \otimes \ ?bs))$

using *kauff-mat.simps* **by** *auto*
moreover then have $\dots = rat\text{-}poly.matrix\text{-}mult$
 $((blockmat \ a) \ \otimes \ (blockmat \ b))$
 $((kauff\text{-}mat \ as) \ \otimes \ (kauff\text{-}mat \ ?bs))$

using *F3 blockmat-tensor-distrib* **by** *(metis)*
moreover then have

\dots
 $= (rat\text{-}poly.matrix\text{-}mult \ (blockmat \ a) \ (kauff\text{-}mat \ as))$
 $\ \otimes \ (rat\text{-}poly.matrix\text{-}mult \ (blockmat \ b) \ (kauff\text{-}mat \ ?bs))$

using *rat-poly.distributivity F0* **by** *auto*
moreover then have \dots
 $= (rat\text{-}poly.matrix\text{-}mult$
 $(blockmat \ a)$
 $(kauff\text{-}mat \ as))$
 $\ \otimes \ (blockmat \ b)$

using *mat1-vert-block* **by** *auto*
moreover then have $\dots = (kauff\text{-}mat \ (a * as))$
 $\ \otimes \ (kauff\text{-}mat \ (basic \ b))$

using *kauff-mat.simps* **by** *auto*
ultimately show *?thesis* **by** *metis*

qed

next

fix *a b as bs*

assume *hyps*:

$codomain\text{-}block \ b \neq 0$
 $\implies is\text{-}tangle\text{-}diagram$
 $(basic \ (make\text{-}vert\text{-}block \ (nat \ (codomain\text{-}block \ b))))$
 $\wedge (is\text{-}tangle\text{-}diagram \ as)$
 $\implies kauff\text{-}mat \ (basic \ (make\text{-}vert\text{-}block \ (nat \ (codomain\text{-}block \ b)))) \ \otimes \ as$
 $= kauff\text{-}mat \ (basic \ (make\text{-}vert\text{-}block \ (nat \ (codomain\text{-}block \ b))))$
 $\ \otimes \ kauff\text{-}mat \ as$

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

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

show *?case*

proof(*cases codomain-block b = 0*)

case *True*

have $((basic \ b) \ \otimes \ (a * as)) = ((b \ \otimes \ a) * as)$

```

    using tensor.simps True by auto
  then have kauff-mat ((basic b)  $\otimes$  (a * as))
    = kauff-mat ((b  $\otimes$  a) * as)
    by auto
  moreover have ... = rat-poly.matrix-mult
    (blockmat (b  $\otimes$  a))
    (kauff-mat as)
    by auto
  moreover have ... = rat-poly.matrix-mult
    ((blockmat b)  $\otimes$  (blockmat a))
    (kauff-mat as)
    using blockmat-tensor-distrib by (metis)
  ultimately have T1:kauff-mat ((basic b)  $\otimes$  (a*as))
    = rat-poly.matrix-mult
    ((blockmat b)  $\otimes$  (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
  then have rat-poly.matrix-mult
    ((blockmat b)  $\otimes$  (blockmat a))
    (kauff-mat as)
    = (blockmat b)
       $\otimes$  (rat-poly.matrix-mult (blockmat a)(kauff-mat as))
    using weak-distributivity2 by auto
  moreover have ... = (kauff-mat (basic b))  $\otimes$  (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-vert-block
  is-tangle-diagram-matrix-match by metis
have F1:codomain-block b  $\neq$  0
  using False by auto
have F2: is-tangle-diagram as
   $\wedge$  is-tangle-diagram ?bs
  using is-tangle-diagram.simps prems by metis
then have F3:kauff-mat (?bs  $\otimes$  as) = kauff-mat ?bs  $\otimes$  kauff-mat as
  using F1 hyps by auto

```

```

moreover have ((basic b) ⊗ (a*as)) = (b ⊗ a) * (?bs ⊗ as)
using False tensor.simps by auto
moreover then have
    kauff-mat ((basic b) ⊗ (a*as))
      = kauff-mat((b ⊗ a) * (?bs ⊗ as))
by auto
moreover then have ...
    = rat-poly.matrix-mult
      (blockmat (b ⊗ a))
      (kauff-mat (?bs ⊗ as))
using kauff-mat.simps by auto
moreover then have ...
    = rat-poly.matrix-mult
      ((blockmat b) ⊗ (blockmat a))
      ((kauff-mat ?bs) ⊗ (kauff-mat as))
using F3 by (metis blockmat-tensor-distrib)
moreover then have ...
    = (rat-poly.matrix-mult
      (blockmat b)
      (kauff-mat ?bs))
      ⊗ (rat-poly.matrix-mult
      (blockmat a)
      (kauff-mat as))
using rat-poly.distributivity F0 by auto
moreover then have ... = (blockmat b)
      ⊗ (rat-poly.matrix-mult
      (blockmat a)
      (kauff-mat as))
using mat1-vert-block by auto
moreover then have ... = (kauff-mat (basic b))
      ⊗ (kauff-mat (a*as))
using kauff-mat.simps by auto
ultimately show ?thesis by metis
qed
qed

end

```

12 Computations: This section can be skipped

```

theory Computations
imports Kauffman-Matrix
begin

```

```

lemma unlink-computation:
  rat-poly-plus (rat-poly-times (rat-poly-times A A) (rat-poly-times A A))

```

$(\text{rat-poly-plus } (\text{rat-poly-times } 2 (\text{rat-poly-times } A (\text{rat-poly-times } A (\text{rat-poly-times } B B))))$
 $(\text{rat-poly-times } (\text{rat-poly-times } B B) (\text{rat-poly-times } B B))) =$
 $((A^4) + (B^4) + 2)$

proof –

have $(\text{rat-poly-times } (\text{rat-poly-times } A A) (\text{rat-poly-times } A A)) = A^4$
by $(\text{simp add: numeral-Bit0})$

moreover have $(\text{rat-poly-times } (\text{rat-poly-times } B B) (\text{rat-poly-times } B B))$
 $= B^4$
by $(\text{simp add: numeral-Bit0})$

moreover have $(\text{rat-poly-times } 2 (\text{rat-poly-times } A (\text{rat-poly-times } A (\text{rat-poly-times } B B))))$
 $= 2$
using inverse1 **by** $(\text{metis mult-2-right one-add-one rat-poly.assoc rat-poly.comm})$

ultimately show $?thesis$ **by** auto

qed

lemma *computation-swingpos:*

$\text{rat-poly-plus } (\text{rat-poly-times } B (\text{rat-poly-times } (A - \text{rat-poly-times } (\text{rat-poly-times } B B) B) B) B))$

$(\text{rat-poly-times } (A - \text{rat-poly-times } (\text{rat-poly-times } B B) B) (\text{rat-poly-times } A (A - \text{rat-poly-times } (\text{rat-poly-times } B B) B))) =$

$\text{rat-poly-times } A (\text{rat-poly-times } (A - \text{rat-poly-times } (\text{rat-poly-times } B B) B) A)$

(**is** $?l = ?r$)

proof –

have $1: (A - \text{rat-poly-times } (\text{rat-poly-times } B B) B)$
 $= A - (B^3)$

by $(\text{metis power3-eq-cube})$

then have $2: (\text{rat-poly-times } (A - \text{rat-poly-times } (\text{rat-poly-times } B B) B) B)$
 $= A * B - (B^3) * B$

by $(\text{metis minus-right-distributivity})$

then have $\dots = 1 - (B^4)$

by $(\text{simp add: inverse1 numeral-Bit0 power3-eq-cube})$

then have $(\text{rat-poly-times } B (\text{rat-poly-times } (A - \text{rat-poly-times } (\text{rat-poly-times } B B) B) B) B))$

$= B - (B^4) * B$

using 2

by $(\text{metis minus-right-distributivity mult.commute mult.right-neutral})$

then have $3: (\text{rat-poly-times } B (\text{rat-poly-times } (A - \text{rat-poly-times } (\text{rat-poly-times } B B) B) B))$

$= B - (B^5)$

by $(\text{metis (no-types, lifting) inverse1 minus-right-distributivity mult.left-commute mult.right-neutral power2-eq-square power-numeral-odd})$

have $(\text{rat-poly-times } (A - \text{rat-poly-times } (\text{rat-poly-times } B B) B) (\text{rat-poly-times } A (A - \text{rat-poly-times } (\text{rat-poly-times } B B) B)))$

$= (A - (B^3)) * (A * (A - (B^3)))$

using 1 **by** auto

moreover then have $\dots = (A - (B^3)) * (A * A - (A * (B^3)))$

by $(\text{metis minus-left-distributivity})$


```

moreover then have ... = (A - (B3))* (A*A - (B2))
  using inverse1
  by (simp add: power2-eq-square power3-eq-cube)
moreover then have ... = A*(A*A - (B2)) - (B3)*(A*A - (B2))
  by (metis minus-right-distributivity)
moreover then have ... = ((A3) - B) - B + (B5)
proof-
  have A*(A*A - (B2)) = (A*A*A - A*(B2))
    by (simp add: right-diff-distrib)
  moreover have ... = (A*A*A - A*(B*B))
    by (metis power2-eq-square)
  moreover have ... = ((A3) - ((A::rat-poly)*B)*B)
    by (simp add: power3-eq-cube)
  moreover have ... = ((A3) - ((1::rat-poly)*B))
    by (metis inverse1)
  moreover have ... = (A3) - B
    by auto
  ultimately have s1:(A::rat-poly)*(A*A - (B2)) = (A3) - (B::rat-poly)
    by metis
have s2:((B::rat-poly)3)*(A*A - (B2)) = (B3)*(A*A) - (B3::nat)*(B2)
  by (metis minus-left-distributivity power3-eq-cube)
  moreover then have ... = (((B::rat-poly)3)*(A*A) - (B5))
    using power-add
  proof-
    have (B3::nat)*(B2) = (B5)
      by (metis One-nat-def Suc-1 numeral-3-eq-3 power-Suc
power-numeral-odd)
    then show ?thesis using s2 by auto
  qed
  moreover then have ... = (((B::rat-poly)*B*B)*(A*A)) - (B5)
    by (metis power3-eq-cube)
moreover then have ... = (((B::rat-poly)*(B*(B*A)*A))) - (B5)
  by auto
  moreover then have ... = (((B::rat-poly)*(B*(1)*A))) - (B5)
    using inverse2 by auto
  moreover then have ... = (((B::rat-poly)*(B*A))) - (B5)
    by auto
  moreover then have ... = (((B::rat-poly))) - (B5)
    using inverse2
    by simp
ultimately have ((B::rat-poly)3)*(A*A - (B2)) = ((B::rat-poly) - (B5))

  by auto
then have A*(A*A - (B2)) - (B3)*(A*A - (B2))
  = (A3) - (B::rat-poly) - ((B::rat-poly) - (B5))
  using s1 by auto
then show ?thesis by auto
qed
ultimately have (rat-poly-times (A - rat-poly-times (rat-poly-times B B) B)

```

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

lemma *computation2*:

$\text{rat-poly-plus } (\text{rat-poly-times } A (\text{rat-poly-times } (B - \text{rat-poly-times } (\text{rat-poly-times } A A) A) A) A))$

$(\text{rat-poly-times } B (\text{rat-poly-times } (\text{rat-poly-times } A A) A) A)$

$(\text{rat-poly-times } B (B - \text{rat-poly-times } (\text{rat-poly-times } A A) A))) =$

$\text{rat-poly-times } B (\text{rat-poly-times } (B - \text{rat-poly-times } (\text{rat-poly-times } A A) A) A) B$

(**is** ?l = ?r)

proof–

have 1: $(B - \text{rat-poly-times } (\text{rat-poly-times } A A) A)$
 $= B - (A^{\wedge}3)$

by (*metis power3-eq-cube*)

then have 2: $(\text{rat-poly-times } (B - \text{rat-poly-times } (\text{rat-poly-times } A A) A) A) A)$
 $= B * A - (A^{\wedge}3) * A$

by (*metis minus-right-distributivity*)

then have ... = $1 - (A^{\wedge}4)$

using *inverse2*

by (*metis mult.commute one-plus-numeral power-add power-one-right semiring-norm(2)*)

semiring-norm(4))

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

using *inverse1* **by** *auto*
moreover then have ... = (((*A::rat-poly*)*(*A*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*)
(*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*)))

$(\text{rat-poly-times } A (\text{rat-poly-times } (B - \text{rat-poly-times } (\text{rat-poly-times } A A) A) A) A)$
 $A))$

using *computation2* **by** *auto*

lemma *computation-toppos:rat-poly-inv* $(\text{rat-poly-times } (A - \text{rat-poly-times } (\text{rat-poly-times } B B) B) A) =$

$\text{rat-poly-times } (B - \text{rat-poly-times } (\text{rat-poly-times } A A) A) B$ (**is** $?l = ?r$)

proof –

have $(\text{rat-poly-times } (A - \text{rat-poly-times } (\text{rat-poly-times } B B) B) A)$

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

by *auto*

moreover then have $\dots = (A*A) - ((B*B)*B)*A$

by *(metis minus-left-distributivity rat-poly.comm)*

moreover then have $\dots = (A*A) - (B*B)*(B*A)$

by *auto*

moreover then have $\dots = (A*A) - (B*B)$

using *inverse2* **by** *auto*

ultimately have $?l = \text{rat-poly-inv } ((A*A) - (B*B))$

by *auto*

then have $1: ?l = (B*B) - (A*A)$

by *auto*

have $?r = (B - ((A*A)*A))*B$

by *auto*

moreover have $\dots = B*B - ((A*A)*A)*B$

by *(metis minus-left-distributivity rat-poly.comm)*

moreover have $\dots = (B*B) - ((A*A)*(A*B))$

by *auto*

moreover have $\dots = ((B::\text{rat-poly})*B) - (A*A)$

using *inverse1* **by** *auto*

ultimately have $?r = (B*B) - (A*A)$

by *auto*

then show $?thesis$ **using** 1 **by** *auto*

qed

lemma *computation-downpos-prelim*:

$\text{rat-poly-inv } (\text{rat-poly-times } (B - \text{rat-poly-times } (\text{rat-poly-times } A A) A) B) =$

$\text{rat-poly-times } (A - \text{rat-poly-times } (\text{rat-poly-times } B B) B) A$ (**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 $\dots = (B*B) - ((A*A)*A)*B$

by *(metis minus-left-distributivity rat-poly.comm)*

moreover then have $\dots = (B*B) - (A*A)*(A*B)$

by *auto*

moreover then have $\dots = (B*B) - (A*A)$

using *inverse1* **by** *auto*

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

lemma *computation-downpos:rat-poly-times* $A (A - \text{rat-poly-times } (\text{rat-poly-times } B B) B) =$
 $\text{rat-poly-inv } (\text{rat-poly-times } B (B - \text{rat-poly-times } (\text{rat-poly-times } A A) A))$
 using *computation-downpos-prelim* **by** (*metis rat-poly.comm*)

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

using *inverse1* **by** *auto*
moreover then have ... = $(A*(A*A)) - B$
by *auto*
ultimately have (*rat-poly-times* A (*rat-poly-times* ($A - \text{rat-poly-times}$ (*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* ($A - \text{rat-poly-times}$ (*rat-poly-times* B B) B) A))
= $B - (A*A*A)$
by *auto*
then have $3: ?l = - (A*A*A)$
using 1 **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-inv* (*rat-poly-times* B (*rat-poly-times* ($B - \text{rat-poly-times}$ (*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* 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 1))
using *inverse2* **by** *auto*
moreover have ... = $- A$
by *auto*
ultimately have $1: (\text{rat-poly-inv } (\text{rat-poly-times } A \text{ (rat-poly-times } B \text{ } A))) = - A$
by *auto*
have (*rat-poly-times* B (*rat-poly-times* ($B - \text{rat-poly-times}$ (*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 ... = $B*((B*B) - ((A*A)*1))$
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* ($B - \text{rat-poly-times}$ (*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* ($B - \text{rat-poly-times}$ (*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 ($B - \text{rat-poly-times}$ (*rat-poly-times* $A \ A) \ A))$)
 $(\text{rat-poly-times}$ ($A - \text{rat-poly-times}$ (*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 ($B - \text{rat-poly-times}$ (*rat-poly-times* $A \ A) \ A)$)
 $= B*B - B*((A*A)*A)$
using minus-left-distributivity by auto
moreover have $\dots = B*B - (B*(A*(A*A)))$
by auto
moreover have $\dots = B*B - ((B*A)*(A*A))$
by auto
moreover have $\dots = B*B - A*A$
using inverse2 by auto
ultimately have $1:\text{rat-poly-times}$ B ($B - \text{rat-poly-times}$ (*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* ($A - \text{rat-poly-times}$ (*rat-poly-times* $B \ B) \ B) \ A)$)
 $= (A*A) - ((B*B)*B)*A$
using minus-right-distributivity by auto
moreover have $\dots = (A*A) - ((B*B)*(B*A))$
by auto
moreover have $\dots = (A*A) - (B*B)$
using inverse2 by auto
ultimately have $2:(\text{rat-poly-times}$ ($A - \text{rat-poly-times}$ (*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*: $(A - \text{rat-poly-times } (\text{rat-poly-times } B B) B)$
 $= A - (B^{\wedge}3)$
 using *power3-eq-cube* by (*metis*)

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

proof–

have 1: $((p::\text{rat-poly}) - (q::\text{rat-poly}))^{\wedge}2$
 $= (p - q)*(p - q)$
 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 $\dots = (p*p) - q*p - p*q + q*q$
 by *auto*
 moreover have $\dots = (p^{\wedge}2) - p*q - p*q + (q^{\wedge}2)$
 using *power2-eq-square* by (*simp add: power2-eq-square*)

ultimately show *?thesis* using 1 by *auto*
 qed

lemma *cube-minus*: $\forall p q.(((p::\text{rat-poly}) - (q::\text{rat-poly}))^{\wedge}3)$
 $= (p^{\wedge}3) - 3*(p^{\wedge}2)*(q) + 3*(p)*(q^{\wedge}2) - (q^{\wedge}3)$

apply(*rule allI*)

apply(*rule allI*)

proof–

fix *p q*

have 1: $((p::\text{rat-poly}) - (q::\text{rat-poly}))^{\wedge}3 = (p - q)*(p - q)^{\wedge}2$
 by (*metis One-nat-def Suc-1 numeral-3-eq-3 power-Suc*)

then have $(p - q)^{\wedge}2 = (p^{\wedge}2) - (2*p*q) + (q^{\wedge}2)$

using *square-subtract* by *auto*

then have 2: $(p - q)*(p - q)^{\wedge}2 = (p - q)*((p^{\wedge}2) - (2*p*q) + (q^{\wedge}2))$
 by *auto*

moreover have 3: $(p - q)*((p^{\wedge}2) - (2*p*q) + (q^{\wedge}2))$

$= p*((p^{\wedge}2) - (2*p*q) + (q^{\wedge}2))$

$- (q*((p^{\wedge}2) - (2*p*q) + (q^{\wedge}2)))$

by (*metis minus-right-distributivity*)

moreover have $p*((p^{\wedge}2) - (2*p*q) + (q^{\wedge}2))$

$= p*(p^{\wedge}2) - p*(2*p*q) + (p*(q^{\wedge}2))$

using *minus-left-distributivity* by (*simp add: distrib-left*)

moreover have $p*(p^{\wedge}2) = p^{\wedge}3$
by (*metis One-nat-def Suc-1 numeral-3-eq-3 power-Suc*)
moreover have $p*(2*p*q) = 2*(p^{\wedge}2)*q$
by (*metis (no-types, lifting) distrib-left mult-2 power2-eq-square semigroup-mult-class.mult.assoc*)
ultimately have $4:p*((p^{\wedge}2) - (2*p*q) + (q^{\wedge}2))$
 $= (p^{\wedge}3) - (2*(p^{\wedge}2)*q) + (p*(q^{\wedge}2))$
by auto
have $q*((p^{\wedge}2) - (2*p*q) + (q^{\wedge}2))$
 $= q*(p^{\wedge}2) - q*(2*p*q) + (q*(q^{\wedge}2))$
by (*simp add: distrib-left minus-left-distributivity*)
moreover have $q*(p^{\wedge}2) = (p^{\wedge}2)*q$
by simp
moreover have $q*(2*p*q) = 2*p*(q^{\wedge}2)$
by (*simp add: power2-eq-square*)
ultimately have $5:q*((p^{\wedge}2) - (2*p*q) + (q^{\wedge}2))$
 $= (p^{\wedge}2)*q - 2*p*(q^{\wedge}2) + (q^{\wedge}3)$
by (*metis One-nat-def Suc-1 numeral-3-eq-3 power-Suc*)
with 1 2 3 4 have $(p - q)^{\wedge}3$
 $= (p^{\wedge}3) - (2*(p^{\wedge}2)*q) + (p*(q^{\wedge}2))$
 $- ((p^{\wedge}2)*q - 2*p*(q^{\wedge}2) + (q^{\wedge}3))$
by auto
moreover have $\dots = (p^{\wedge}3) - (2*(p^{\wedge}2)*q) + (p*(q^{\wedge}2))$
 $- (p^{\wedge}2)*q + 2*p*(q^{\wedge}2) - (q^{\wedge}3)$
by auto
moreover have $\dots = (p^{\wedge}3) - 3*(p^{\wedge}2)*(q) + 3*(p)*(q^{\wedge}2) - (q^{\wedge}3)$
by auto
ultimately show
 $(p - q)^{\wedge}3$
 $= \text{rat-poly-plus } (p^{\wedge}3 -$
 rat-poly-times
 $\text{rat-poly-times } 3 \text{ (p}^2\text{)) } q$
 $\text{rat-poly-times (rat-poly-times } 3 \text{ p) (q}^2\text{))}$
 $- q^{\wedge}3$
by auto

qed

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

lemma cube-minus2:

fixes $p \ q$

shows $((p::\text{rat-poly}) - (q::\text{rat-poly}))^{\wedge}3$
 $= (p^{\wedge}3) - 3*(p^{\wedge}2)*(q) + 3*(p)*(q^{\wedge}2) - (q^{\wedge}3)$

using *cube-minus* **by auto**

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

lemma *sub1*:
assumes $p * q = 1$
shows $r * (p * q) = r * 1$
using *assms* **by** *metis*

lemma *n-distrib*: $(A \wedge^{n::nat}) * (B \hat{n}) = (A * B) \hat{n}$
by (*induct n*)(*auto*)

lemma *rat-poly-id-pow*: $(1::rat-poly) \hat{n} = 1$
by (*induct n*)(*auto*)

lemma *power-prod*: $(A \wedge^{n::nat}) * (B \hat{n}) = (1::rat-poly)$
apply (*simp add:n-distrib*)
apply (*simp add:inverse1*)
done
lemma (*pCons 0 1*) $\neq 0$
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

$rat-poly.matrix-mult (blockmat (make-vert-block (nat (domain-wall b)))) (kauff-mat b)$
 $= (kauff-mat b)$

proof –

have $mat (2 \wedge^{nat (domain-wall b)}) (length (kauff-mat b)) (kauff-mat b)$
by (*metis assms matrix-kauff-mat*)

moreover have $(blockmat (make-vert-block (nat (domain-wall b))))$
 $= mat1 (2 \wedge^{nat (domain-wall b)})$

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

$rat-poly.matrix-mult (kauff-mat b) (blockmat (make-vert-block (nat (codomain-wall b))))$

$$= (\text{kauff-mat } b)$$

proof –
have $\text{mat } (\text{rat-poly.row-length } (\text{kauff-mat } b)) (2^\sim(\text{nat } (\text{codomain-wall } b))) (\text{kauff-mat } b)$
by $(\text{metis assms matrix-kauff-mat})$
moreover have $(\text{blockmat } (\text{make-vert-block } (\text{nat } (\text{codomain-wall } b))))$
 $= \text{mat1 } (2^\sim(\text{nat } (\text{codomain-wall } b)))$
using $\text{make-vert-block-map-blockmat}$ **by** auto
ultimately show $?thesis$ **by** $(\text{metis mat1-rt-mult})$
qed

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

proof –
assume $\text{asm}:\text{compress-top } w1 \ w2$
have $\exists B.((w1 = (\text{basic } (\text{make-vert-block } (\text{nat } (\text{domain-wall } B)))) \circ B)$
 $\wedge (w2 = (B \circ (\text{basic } (\square)))) \wedge (\text{codomain-wall } B = 0)$
 $\wedge (\text{is-tangle-diagram } B)$
using compress-top-def **asm** **by** auto
then obtain B **where** $(w1 = (\text{basic } (\text{make-vert-block } (\text{nat } (\text{domain-wall } B)))) \circ B)$
 $\wedge (w2 = (B \circ (\text{basic } (\square)))) \wedge (\text{codomain-wall } B =$
 $0) \wedge (\text{is-tangle-diagram } B)$
by auto
then have $1:(w1 = (\text{basic } (\text{make-vert-block } (\text{nat } (\text{domain-wall } B)))) \circ B)$
 $\wedge (w2 = (B \circ (\text{basic } (\square)))) \wedge (\text{codomain-wall } B =$
 $0) \wedge (\text{is-tangle-diagram } B)$
by auto
then have $\text{kauff-mat}(w1) = (\text{kauff-mat } ((\text{basic } (\text{make-vert-block } (\text{nat } (\text{domain-wall } B)))) \circ B))$
by auto
moreover then have $\dots = \text{kauff-mat } ((\text{make-vert-block } (\text{nat } (\text{domain-wall } B)))) * B$
by auto
moreover then have $\dots = \text{rat-poly.matrix-mult } (\text{blockmat } (\text{make-vert-block } (\text{nat } (\text{domain-wall } B))))$
 $(\text{kauff-mat } B)$
by auto
moreover then have $\dots = (\text{kauff-mat } B)$
using 1 $\text{mat1-vert-wall-left}$ **by** (metis)
ultimately have $\text{kauff-mat}(w1) = \text{kauff-mat } B$
by auto
moreover have $\text{kauff-mat } w2 = \text{kauff-mat } B$
using 1 **by** $(\text{metis left-mat-compose})$
ultimately show $?thesis$ **by** auto
qed

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

$$= n$$

using domain-make-vert **by** auto

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

proof –

assume *asm*: *compress-bottom* $w1 \ w2$

have $\exists B. ((w1 = B \circ (\text{basic } (\text{make-vert-block } (\text{nat } (\text{codomain-wall } B))))))$
 $\wedge (w2 = ((\text{basic } (\square) \circ B)) \wedge (\text{domain-wall } B = 0))$
 $\wedge (\text{is-tangle-diagram } B)$

using *compress-bottom-def* *asm* **by** *auto*

then obtain B **where** $((w1 = B \circ (\text{basic } (\text{make-vert-block } (\text{nat } (\text{codomain-wall } B))))))$
 $\wedge (w2 = ((\text{basic } (\square) \circ B)) \wedge (\text{domain-wall } B = 0))$
 $\wedge (\text{is-tangle-diagram } B)$

by *auto*

then have $1: ((w1 = B \circ (\text{basic } (\text{make-vert-block } (\text{nat } (\text{codomain-wall } B))))))$
 $\wedge (w2 = ((\text{basic } (\square) \circ B)) \wedge (\text{domain-wall } B = 0))$
 $\wedge (\text{is-tangle-diagram } B)$

by *auto*

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

by *auto*

moreover then have $\dots = \text{rat-poly.matrix-mult } (\text{kauff-mat } B)$
 $(\text{kauff-mat } (\text{basic } (\text{make-vert-block } (\text{nat } (\text{codomain-wall } B))))))$

proof –

have *is-tangle-diagram* B

using 1 **by** *auto*

moreover have *is-tangle-diagram* $(\text{basic } (\text{make-vert-block } (\text{nat } (\text{codomain-wall } B))))$

using *is-tangle-diagram.simps* **by** *auto*

moreover have $\text{codomain-wall } B = \text{domain-wall } (\text{basic } (\text{make-vert-block } (\text{nat } (\text{codomain-wall } B))))$

proof –

have $\text{codomain-wall } B \geq 0$

apply *(induct B)*

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

then have $\text{domain-block } (\text{make-vert-block } (\text{nat } (\text{codomain-wall } B)))$
 $= \text{codomain-wall } B$

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 $\dots = \text{rat-poly.matrix-mult } (\text{kauff-mat } B)$
 $(\text{blockmat } (\text{make-vert-block } (\text{nat } (\text{codomain-wall } B))))$

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

moreover then have $\dots = (\text{kauff-mat } B)$

using 1 *mat1-vert-wall-right* **by** *auto*

ultimately have $\text{kauff-mat}(w1) = \text{kauff-mat } B$

by *auto*
moreover have $\text{kauff-mat } w2 = \text{kauff-mat } B$
 using 1 by (*metis right-mat-compose*)
ultimately show *?thesis* by *auto*
qed

theorem *compress-inv:compress* $w1\ w2 \implies (\text{kauff-mat } w1 = \text{kauff-mat } w2)$
unfolding *compress-def* using *compress-bottom-inv compress-top-inv*
 by *auto*

lemma *straghten-topdown-computation: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-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*)
done

theorem *straighten-topdown-inv:straighten-topdown* $w1\ w2 \implies (\text{kauff-mat } w1) =$
 $(\text{kauff-mat } w2)$
unfolding *straighten-topdown-def* using *straghten-topdown-computation* by *auto*

lemma *straghten-downtop-computation: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-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*)
done

theorem *straighten-downtop-inv:straighten-downtop* $w1\ w2 \implies (\text{kauff-mat } w1) =$
 $(\text{kauff-mat } w2)$
unfolding *straighten-downtop-def* using *straghten-downtop-computation* by *auto*

theorem *straighten-inv:straighten* $w1\ w2 \implies (\text{kauff-mat } w1) = (\text{kauff-mat } w2)$
unfolding *straighten-def* using *straighten-topdown-inv straighten-downtop-inv* by *auto*

lemma *kauff-mat-swingpos:*
 $\text{kauff-mat } (r\text{-over-braid}) = \text{kauff-mat } (l\text{-over-braid})$
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* $w1\ w2 \implies (kauff\text{-}mat\ w1) = (kauff\text{-}mat\ w2)$
unfolding *swing-pos-def* **using** *kauff-mat-swingpos* **by** *auto*

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

lemma *swing-neg-inv:swing-neg* $w1\ w2 \implies (kauff\text{-}mat\ w1) = (kauff\text{-}mat\ w2)$
unfolding *swing-neg-def* **using** *kauff-mat-swingneg* **by** *auto*

theorem *swing-inv*:
 $swing\ w1\ w2 \implies (kauff\text{-}mat\ w1) = (kauff\text{-}mat\ w2)$
unfolding *swing-def* **using** *swing-pos-inv* *swing-neg-inv* **by** *auto*

lemma *rotate-toppos-kauff-mat*: $kauff\text{-}mat\ ((basic\ [vert,over])\circ(basic\ [cap,vert]))$
 $= kauff\text{-}mat\ ((basic\ [under,vert])\circ(basic\ [vert,cap]))$
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-toppos-inv:rotate-toppos* $w1\ w2 \implies (kauff\text{-}mat\ w1) = (kauff\text{-}mat\ w2)$
unfolding *rotate-toppos-def* **using** *rotate-toppos-kauff-mat* **by** *auto*

lemma *rotate-topneg-kauff-mat*: $kauff\text{-}mat\ ((basic\ [vert,under])\circ(basic\ [cap,vert]))$
 $= kauff\text{-}mat\ ((basic\ [over,vert])\circ(basic\ [vert,cap]))$
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* $w1\ w2 \implies (\text{kauff-mat } w1) = (\text{kauff-mat } w2)$
unfolding *rotate-topneg-def* **using** *rotate-topneg-kauff-mat* **by** *auto*

lemma *rotate-downpos-kauff-mat:*
 $\text{kauff-mat } ((\text{basic } [cup,vert]) \circ (\text{basic } [vert,over])) = \text{kauff-mat } ((\text{basic } [vert,cup]) \circ (\text{basic } [under,vert]))$
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* $w1\ w2 \implies (\text{kauff-mat } w1) = (\text{kauff-mat } w2)$
unfolding *rotate-downpos-def* **using** *rotate-downpos-kauff-mat* **by** *auto*

lemma *rotate-downneg-kauff-mat:*
 $\text{kauff-mat } ((\text{basic } [cup,vert]) \circ (\text{basic } [vert,under])) = \text{kauff-mat } ((\text{basic } [vert,cup]) \circ (\text{basic } [over,vert]))$
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* $w1\ w2 \implies (\text{kauff-mat } w1) = (\text{kauff-mat } w2)$
unfolding *rotate-downneg-def* **using** *rotate-downneg-kauff-mat* **by** *auto*

theorem *rotate-inv:rotate* $w1\ w2 \implies (\text{kauff-mat } w1) = (\text{kauff-mat } w2)$
unfolding *rotate-def* **using** *rotate-downneg-inv* *rotate-downpos-inv* *rotate-topneg-inv*

rotate-toppos-inv **by** *auto*

lemma *positive-flip-kauff-mat:*
 $\text{kauff-mat } (\text{left-over}) = \text{kauff-mat } (\text{right-over})$


```

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])  $\circ$  (basic [under]))
= kauff-mat ((basic [vert,vert])  $\circ$  (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])  $\circ$  (basic [over]))
= kauff-mat ((basic [vert,vert])  $\circ$  (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)
using computation-pull-pos-neg by (simp add: computation-downpos)

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

theorem pull-inv:pull w1 w2  $\implies$  (kauff-mat w1) = (kauff-mat w2)
unfolding pull-def using pull-posneg-inv pull-negpos-inv by auto

theorem slide-inv:slide w1 w2  $\implies$  (kauff-mat w1 = kauff-mat w2)
proof -
  assume assm:slide w1 w2
  have  $\exists B.((w1 = ((basic (make-vert-block (nat (domain-block B)))) \circ (basic B)))$ 
     $\wedge (w2 = ((basic B) \circ (basic (make-vert-block (nat (codomain-block B))))))$ 
     $\wedge ((domain-block B) \neq 0))$ 
    using slide-def assm by auto
  then obtain B where  $((w1 = ((basic (make-vert-block (nat (domain-block$ 
    B)))) \circ (basic B)))
     $\wedge (w2 = ((basic B) \circ (basic (make-vert-block (nat (codomain-block B))))))$ 
     $\wedge ((domain-block B) \neq 0)$  by auto
  then have 1: $((w1 = ((basic (make-vert-block (nat (domain-block B)))) \circ (basic$ 
    B)))
     $\wedge (w2 = ((basic B) \circ (basic (make-vert-block (nat (codomain-block B))))))$ 
     $\wedge ((domain-block B) \neq 0))$ 
    by auto
  have kauff-mat w1 = kauff-mat (basic B)
  proof -
    have s1:mat ( $2^{\sim}$ (nat (domain-block B))) (length (blockmat B)) (blockmat B)
      by (metis matrix-blockmat row-length-domain-block)
    have w1 =  $((basic (make-vert-block (nat (domain-wall (basic B)))) \circ (basic B)))$ 
      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 ... = rat-poly.matrix-mult (mat1 ( $2^{\sim}$ (nat (domain-block
    B)))) (blockmat B)
      using kauff-mat.simps(1) domain-wall.simps(1) by (metis make-vert-block-map-blockmat)
    moreover have ... = (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 -

```

```

have s1:mat (2^(nat (domain-block B))) (2^(nat (codomain-block B))) (blockmat
B)
  by (metis length-codomain-block matrix-blockmat row-length-domain-block)
  have w2 = ((basic B) o(basic (make-vert-block (nat (codomain-wall (basic
B))))))
    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 tangle-compose-matrix is-tangle-diagram.simps
    by (metis compose-Nil kauff-mat.simps(1) kauff-mat.simps(2))
  moreover then have ... = rat-poly.matrix-mult (blockmat B) (mat1 (2^(nat
(codomain-block B))))
    using kauff-mat.simps(1) domain-wall.simps(1)
    by (metis blockmat-make-vert codomain-wall.simps(1) make-vert-equiv-mat)
  moreover have ... = (blockmat B)
    using s1 by (metis mat1-rt-mult)
  ultimately show ?thesis by auto
qed
  ultimately show ?thesis by auto
qed

```

```

theorem framed-linkrel-inv:framed-linkrel w1 w2  $\implies$  (kauff-mat w1) = (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::wall)  $\sim$  f w2  $\implies$  kauff-mat w1 = 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])) = [[-(A^2) - (B^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)
  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])) = [(((A^4)+(B^4)) + 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

trefoil-polynomial \equiv

rat-poly-plus

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

kauffman bracket of trefoil

lemma *trefoil*:

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

by(*simp add: mat-multI-def matT-vec-multI-def rat-poly.row-length-def*
scalar-prod trefoil-polynomial-def)

end

theory *Knot-Theory*

imports *Kauffman-Invariance Example*

begin

end