# Knot Theory 

T. V. H. Prathamesh

September 13, 2023


#### 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 proper- ties ..... 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
$\mid$ 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 $\otimes 65$ ) where
concatenates-Nil: [] $\otimes y s=y s$
concatenates-Cons: $((x \# x s) \otimes y s)=x \#(x s \otimes y s)$
lemma empty-concatenate: $x s \otimes N i l=x s$
by (induction xs) (auto)

Associativity properties of Conscatenation
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 (infixr $\circ 66$ ) where
compose-Nil: $($ basic $x) \circ y s=\operatorname{prod} x y s$
compose-Cons: $((\operatorname{prod} x x s) \circ y s)=\operatorname{prod} x(x s \circ y s)$
Associativity properties of composition
lemma compose-leftassociativity: $(((x::$ wall $) \circ y) \circ z)=(x \circ y \circ z)$
by (induction $x$ ) (auto)
lemma compose-rightassociativity: $(x::$ 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 $\Rightarrow$ nat
where
block-length [] = 0
block-length (Cons xy) $=1+($ block-length $y)$

```
primrec domain::brick \(\Rightarrow\) int
where
domain vert \(=1 \mid\)
domain cup \(=0 \mid\)
domain cap \(=2 \mid\)
domain over \(=2 \mid\)
domain under \(=2\)
lemma domain-non-negative: \(\forall x\).(domain \(x) \geq 0\)
proof-
    have \(\forall x .(x=\) vert \() \vee(x=\) over \() \vee(x=\) under \() \vee(x=\) cap \() \vee(x=\) cup \()\)
    by (metis brick.exhaust)
moreover have
```

```
    \(\forall x .(((x=\) vert \() \vee(x=\) over \() \vee(x=\) under \() \vee(x=\) cap \() \vee(x=\) cup \()) \longrightarrow(\) 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 $=1 \mid$
codomain cup $=2 \mid$
codomain cap $=0$
codomain over $=2 \mid$
codomain under $=2$
primrec domain-block::block $\Rightarrow$ int
where
domain-block [] = 0
$\mid$ 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
$\mid$ 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 $(x * y s)=$ domain-block $x$
fun codomain-wall:: wall $\Rightarrow$ int where
codomain-wall (basic $x)=$ codomain-block $x$
|codomain-wall $(x * y s)=$ codomain-wall ys
lemma domain-wall-compose: domain-wall $(x s \circ y s)=$ domain-wall xs by (induction xs) (auto)

```
lemma codomain-wall-compose: codomain-wall (xsoys) = codomain-wall ys
    by (induction xs) (auto)
```

this lemma tells us the number of incoming and outgoing strands of a composition of two wall
absolute value
definition abs::int $\Rightarrow$ int where
abs $x \equiv$ if $(x \geq 0)$ then $x$ else $(0-x)$
theorems about abs
lemma abs-zero: assumes abs $x=0$ shows $x=0$
using abs-def assms eq-iff-diff-eq-0
by metis
lemma abs-zero-equality: assumes abs $(x-y)=0$ shows $x=y$
using assms abs-zero eq-iff-diff-eq-0
by blast
lemma abs-non-negative: abs $x \geq 0$
using abs-def diff-0 le-cases neg-0-le-iff-le
by auto
lemma abs-non-negative-sum: assumes abs $x+a b s y=0$
shows abs $x=0$ and abs $y=0$
using abs-def diff-0 abs-non-negative neg-0-le-iff-le
add-nonneg-eq-0-iff assms
apply (metis)
by (metis abs-non-negative add-nonneg-eq-0-iff assms)
The following lemmas tell us that the number of incoming and outgoing strands of every brick is a non negative integer
lemma domain-nonnegative: (domain $x) \geq 0$
using domain.simps brick.exhaust le-cases not-numeral-le-zero zero-le-one by (metis)
lemma codomain-nonnegative: (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: domain-block $x \geq 0$
by (induction x) (auto simp add: domain-nonnegative)
lemma codomain-block-nonnegative: (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: $(($ domain-block $(x \#$ Nil $))>0) \vee(($ domain-block $y)>0)$
$\Longrightarrow$ (domain-block $(x \# y)>0)$
proof-
have $($ domain-block $(x \# y))=($ domain $x)+($ domain-block $y)$ by auto
also have (domain $x)=($ domain-block $(x \#$ Nil $))$ by auto
then have (domain-block $(x \#$ Nil $)>0)=($ domain $x>0)$ by auto
then have $(($ domain $x>0) \vee($ domain-block $y>0)) \Longrightarrow($ domain $x+$ do-main-block y) $>0$
using domain-nonnegative add-nonneg-pos add-pos-nonneg domain-block-nonnegative
by metis
from this
show $(($ domain-block $(x \#$ Nil $))>0) \vee(($ domain-block $y)>0)$ $\Longrightarrow$ (domain-block $(x \# y)>0)$
by auto
qed
lemma domain-additive: (domain-block $(x \otimes y))=($ domain-block $x)+($ domain-block y)
by (induction $x$ )(auto)
lemma codomain-additive: $($ codomain-block $(x \otimes y))=($ codomain-block $x)+($ codomain-block y)
by (induction $x)$ (auto)
lemma domain-zero-sum: assumes (domain-block $x)+($ domain-block y) $=0$
shows domain-block $x=0$ and 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 domain-block $y>0$ or domain-block
$y>0$
shows (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 codomain-block $y>0$ or codomain-block $y>0$
shows (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 $\Rightarrow$ bool where
brick-is-cup vert $=$ False
brick-is-cup cup $=$ True
brick-is-cup cap $=$ False
brick-is-cup over $=$ False
brick-is-cup under $=$ False
primrec $i s$-cup::block $\Rightarrow$ bool
where
is-cup [] = True
is-cup $(x \# y)=($ if $(x=$ cup $)$ then (is-cup $y)$ else False $)$

```
lemma brickcount-zero-implies-cup:(domain \(x=0) \Longrightarrow(x=\) cup \()\)
    by (cases \(x\) ) (auto)
lemma brickcount-zero-implies-brick-is-cup:(domain \(x=0) \Longrightarrow(\) brick-is-cup \(x)\)
    by (cases \(x\) ) (auto)
lemma domain-zero-implies-is-cup:(domain-block \(x=0) \Longrightarrow(\) is-cup \(x)\)
proof \((\) induction \(x)\)
    case Nil
    show? case by auto
    next
    case (Cons a y)
    show ?case
    proof-
        have step 1: domain-block \((a \# y)=(\) domain \(a)+(\) domain-block \(y)\)
                by auto
    with domain-zero-sum havedomain-block \(y=0\)
                    by (metis (full-types) Cons.prems domain-block-nonnegative do-
main-positive leD neq-iff)
    then have step2: (is-cup y)
                using Cons.IH by (auto)
    with step1 and domain-zero-sum
                have domain \(a=0\)
                            using Cons.prems 〈domain-block \(y=0\rangle\) 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 step 2 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
$\mid$ is-tangle-diagram $(x * x s)=($ if is-tangle-diagram xs
then (codomain-block $x=$ domain-wall $x s$ )
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<br>imports Preliminaries<br>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 \(x=\operatorname{prod}(\) cup\#[]) \((\) basic \((\) cap\#[])) in exI) \((\) auto \()\)
typedef Link-Diagram \(=\{(x::\) wall \()\). is-link-diagram \(x\}\)
    by \((\) rule-tac \(x=\operatorname{prod}(\) 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)}=(\mathrm{ Abs-Tangle-Diagram y)
shows }x=
using Rep-Tangle-Diagram Abs-Tangle-Diagram-inverse assms mem-Collect-eq
by metis
```

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

```
lemma is-tangle-left-compose:
    is-tangle-diagram \((x \circ y) \Longrightarrow\) 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 \((\operatorname{prod} z z s)\)
    have \((z * z s) \circ y=(z *(z s \circ y))\) by auto
    then have is-tangle-diagram \((z *(z s o y))\) using prod by auto
    moreover then have 1: is-tangle-diagram zs
        using is-tangle-diagram.simps(2) prod.hyps prod.prems by metis
    ultimately have domain-wall (zs \(\circ y)=\) codomain-block \(z\)
                by (metis is-tangle-diagram.simps(2))
    moreover have domain-wall \((z s \circ y)=\) domain-wall zs
                using domain-wall-def domain-wall-compose by auto
    ultimately have domain-wall \(z s=\) codomain-block \(z\) by auto
    then have is-tangle-diagram \((z * z s)\)
                by (metis 1 is-tangle-diagram.simps(2))
    then show ?case by auto
qed
lemma is-tangle-right-compose:
    is-tangle-diagram \((x \circ y) \Longrightarrow\) is-tangle-diagram \(y\)
```

```
proof (induct x)
    case (basic z)
    have (basic z) ○ y = (z*y) using basic by auto
    then have is-tangle-diagram y
        unfolding is-tangle-diagram.simps(2) using basic.prems by (metis is-tangle-diagram.simps(2))
    then show ?case using basic.prems by auto
    next
    case (prod z zs)
    have }((z*zs)\circy)=(z*(zs\circy)) by aut
    then have is-tangle-diagram (z*(zs\circy)) using prod by auto
    then have is-tangle-diagram (zs ○ 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:
assumesis-tangle-diagram y
shows ((is-tangle-diagram $x$ )
$\wedge($ codomain-wall $x=$ domain-wall $y))$
$\Longrightarrow$ is-tangle-diagram $(x \circ y)$
$\operatorname{proof}($ 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.prems by auto
moreover have is-tangle-diagram y
using assms by auto
ultimately have is-tangle-diagram $(z * y)$
unfolding is-tangle-diagram-def by auto
then show? case by auto
next
case (prod $z z s)$
have is-tangle-diagram ( $z * z s$ )
using prod.prems by metis
then have codomain-block $z=$ domain-wall zs
using is-tangle-diagram.simps(2) prod.prems by metis
then have codomain-block $z=$ domain-wall ( $z s \circ y$ )
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.prems)
ultimately have is-tangle-diagram $(z *(z s \circ y))$
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 (domain-wall $y=$ 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:
is-tangle-diagram $(x \circ y) \Longrightarrow($ domain-wall $y)=($ codomain-wall $x)$
proof (induct $x$ )
case (basic z)
have (basic $z$ ) $\circ y=z * y$
using compose-def basic by auto
then have
is-tangle-diagram $(($ basic $z) \circ y) \Longrightarrow$
(is-tangle-diagram $y) \wedge($ codomain-block $z=$ domain-wall $y)$
using is-tangle-diagram.simps(2) by (metis)
then have (codomain-block $z)=($ domain-wall $y)$
using basic.prems by auto
moreover have codomain-wall (basic z) = codomain-block $z$
using domain-wall-compose by auto
ultimately have (codomain-wall (basic $z))=($ domain-wall $y)$
by auto
then show? case by simp
next
case $(\operatorname{prod} z z s)$
have codomain-wall $z s=$ domain-wall $y$ using prod.hyps prod.prems by (metis compose-Nil compose-leftassociativity is-tangle-right-compose)
moreover have codomain-wall $z s=$ codomain-wall $(z * z s)$
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 $x y=$ 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 domain-wall $x=$ codomain-wall $y$
shows (Abs-Tangle-Diagram $x) \circ($ Abs-Tangle-Diagram $y)$ $=($ Abs-Tangle-Diagram $(x \circ y))$
using Abs-Tangle-Diagram-inverse assms(1) assms(2) compose-Tangle-def mem-Collect-eq
by auto
definition domain-Tangle::Tangle-Diagram $\Rightarrow$ int
where
domain-Tangle $x=$ domain-wall(Rep-Tangle-Diagram $x)$
definition codomain-Tangle::Tangle-Diagram $\Rightarrow$ int where
codomain-Tangle $x=$ codomain-wall(Rep-Tangle-Diagram $x)$
end

## 3 Tangle_Algebra: Tensor product of tangles and its properties

theory Tangle-Algebra
imports Tangles
begin

## 4 Definition of tensor product of walls

the following definition is used to construct a block of n vert strands
primrec make-vert-block:: nat $\Rightarrow$ block
where
make-vert-block $0=[]$
$\mid$ make-vert-block (Suc n) $=$ vert\#(make-vert-block $n)$
lemma domain-make-vert:domain-block (make-vert-block $n$ ) $=$ int $n$
by (induction $n$ ) (auto)
lemma codomain-make-vert:codomain-block (make-vert-block $n$ ) $=$ int $n$ by (induction $n$ ) (auto)
fun tensor::wall => wall => wall (infixr $\otimes 65)$
where
1:tensor $($ basic $x)($ basic $y)=($ basic $(x \otimes y))$
|2:tensor $(x * x s)($ basic $y)=($

```
        if (codomain-block y = 0)
        then (x\otimesy)*xs
        else
        (x\otimesy)
        *(xs\otimes(basic (make-vert-block (nat (codomain-block y))))))
|3:tensor (basic x) (y*ys)=(
    if (codomain-block x = 0)
        then (x\otimesy)*ys
        else
        (x\otimesy)
        *((basic (make-vert-block (nat (codomain-block x))))\otimes ys))
|4:tensor }(x*xs)(y*ys)=(x\otimesy)*(xs\otimesys
```


## 5 Properties of tensor product of tangles

lemma Nil-left-tensor:xs $\otimes($ basic ([])) $=x s$
by (cases xs) (auto simp add:empty-concatenate)
lemma Nil-right-tensor:(basic ([])) $\otimes x s=x s$
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 $x y=$ Abs-Tangle-Diagram $(($ Rep-Tangle-Diagram $x) \otimes($ Rep-Tangle-Diagram y))
lemma tensor $($ basic $[$ vert $])($ basic $([$ vert $]))=($ basic $(([$ vert $]) \otimes([$ vert $])))$ by $\operatorname{simp}$
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 $(x s \otimes y s)=$ domain-wall xs + domain-wall ys proof(cases xs)
fix $x$
assume $A: x s=$ basic $x$
then have domain-wall $(x s \otimes y s)=$ domain-wall $x s+$ domain-wall ys
proof(cases ys)
fix $y$
assume $B: y s=$ basic $y$
have domain-block $(x \otimes y)=$ domain-block $x+$ domain-block $y$ using domain-additive by auto
then have domain-wall $(x s \otimes y s)=$ domain-wall $x s+$ domain-wall ys
using tensor.simps(1) A B by auto
thus ?thesis by auto
next
fix $z z s$
assume $C: y s=(z * z s)$
have domain-wall $(x s \otimes y s)=$ domain-wall $x s+$ domain-wall ys
proof (cases (codomain-block $x)=0$ )
assume codomain-block $x=0$
then have $(x s \otimes y s)=(x \otimes z) * z s$ using $A C$ tensor.simps(4) by auto
then have domain-wall $(x s \otimes y s)=$ domain-block $(x \otimes z)$ 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 $(x \otimes z)=$ domain-block $x+$ domain-block $z$ using domain-additive by auto
ultimately show ?thesis by auto
next
assume codomain-block $x \neq 0$
have ( $x s \otimes y s$ )

$$
=(x \otimes z)
$$

$*(($ basic (make-vert-block (nat (codomain-block $x)))) \otimes z s)$ using tensor.simps(3) A $C$ 〈codomain-block $x \neq 0\rangle$ by auto
then have domain-wall $(x s \otimes y s)=$ domain-block $(x \otimes z)$ 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 $(x \otimes z)=$ 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 $z z s$
assume $D: x s=z * z s$
then have domain-wall $(x s \otimes y s)=$ domain-wall $x s+$ domain-wall ys
proof (cases ys)
fix $y$
assume E:ys = basic $y$
then have domain-wall $(x s \otimes y s)=$ domain-wall $x s+$ domain-wall ys
proof (cases codomain-block $y=0$ )
assume codomain-block $y=0$
have $(x s \otimes y s)=(z \otimes y) * z s$
using tensor．simps（2）$D E$ 〈codomain－block $y=0$ 〉by auto
then have domain－wall $(x s \otimes y s)=$ domain－block $(z \otimes y)$
by auto
moreover have domain－wall $x s=$ 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 $(z \otimes y)=$ domain－block $z+$ domain－block $y$
using domain－additive by auto
ultimately show ？thesis by auto
next
assume codomain－block $y \neq 0$
have $(x s \otimes y s)$

$$
=
$$

$$
(z \otimes y)
$$

＊（zs $\otimes($ basic（make－vert－block（nat（codomain－block y）））））
using tensor．simps（3）$D E$ 〈codomain－block $y \neq 0$ 〉 by auto
then have domain－wall $(x s \otimes y s)=$ domain－block $(z \otimes y)$ 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 $(z \otimes y)=$ domain－block $z+$ domain－block $y$ using domain－additive by auto
ultimately show ？thesis by auto
qed
then show ？thesis by auto
next
fix $w w s$
assume $F: y s=w * w s$
have $(x s \otimes y s)=(z \otimes w) *(z s \otimes w s)$
using $D F$ by auto
then have domain－wall $(x s \otimes y s)=$ domain－block $(z \otimes w)$ 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 $(z \otimes w)=$ 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 \otimes ys) = codomain-wall xs + codomain-wall ys
proof(induction xs ys rule:tensor.induct)
fix xs ys
let ?case = (codomain-wall ((basic xs) \otimes (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 \not=0
        \Longrightarrow ~ c o d o m a i n - w a l l ~
            (xs \otimes 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)\otimes (basic y))
            =(codomain-wall (x*xs)) + (codomain-wall (basic y))
show ?case
proof(cases (codomain-block y=0))
    case True
    have }((x*xs)\otimes(\mathrm{ basic y)) = (x@y)*xs
                            using Tangle-Algebra.2 True by auto
    then have codomain-wall ((x*xs)\otimes (basic y))
                    = codomain-wall ((x\otimesy)*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)\otimes(basic y)
        =(x\otimesy)
                *(xs\otimes(basic (make-vert-block (nat (codomain-block y)))))
    using False by (metis Tangle-Algebra.2)
    moreover then have codomain-wall ((x*xs) \otimes (basic y))
                    = codomain-wall(...)
    by auto
moreover have ...
                = codomain-wall
                        (xs\otimes(basic (make-vert-block (nat (codomain-block y)))))
            using domain-wall.simps by auto
    moreover have ...
\[
\begin{aligned}
= & \text { codomain-wall xs } \\
& + \text { codomain-wall }
\end{aligned}
\]
```

```
                                    (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}\not=0
        codomain-wall
            (basic (make-vert-block (nat (codomain-block x)))\otimes ys)
            = codomain-wall
                    (basic (make-vert-block (nat (codomain-block x))))
                        + codomain-wall ys)
let ?case = codomain-wall ((basic x)\otimes (y*ys))
                = codomain-wall (basic x) + codomain-wall (y*ys)
show ?case
    proof(cases codomain-block x = 0)
    case True
        have (basic x)\otimes(y*ys)=(x\otimesy)*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)\otimes(y*ys)
        =(x\otimesy)
                *((basic (make-vert-block (nat (codomain-block x))))\otimes 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))))\otimes 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 $x \otimes y * y s=(x \otimes y) *$ (basic $($ make-vert-block $($ nat $($ codomain-block $x))) \otimes y s)\rangle\langle c o d o m a i n-w a l l ~((x \otimes y) *($ basic $($ make-vert-block $($ nat $($ codomain-block $x))) \otimes y s))=$ codomain-wall $($ basic $($ make-vert-block
 $x))) \otimes y s)=$ codomain-block $x+$ codomain-wall $y s\rangle)$
qed
next
fix $x$ xs $y$ ys
assume case-4:codomain-wall $(x s \otimes y s)=$ codomain-wall $x s+$ codomain-wall
let ?case $=$ codomain-wall $((x * x s) \otimes(y * y s))$ $=$ codomain-wall $(x * x s)+$ codomain-wall $(y * y s)$
have $((x * x s) \otimes(y * y s))=(x \otimes y) *(x s \otimes y s)$
using 4 by auto
then have codomain-wall (...) = codomain-wall (...)
by auto
then have $\ldots=$ codomain-wall $(x s \otimes y s)$
using codomain-wall.simps(2) by auto
then have $\ldots=$ codomain-wall xs + codomain-wall ys using case-4 by auto
then have $\ldots=$ codomain-wall $(x * x s)+($ codomain-wall $(y * y s))$
using codomain-wall.simps(2) by auto
then show? case by (metis «codomain-wall $((x \otimes y) *(x s \otimes y s))=$ codomain-wall $(x s \otimes y s)\rangle\langle x * x s \otimes y * y s=(x \otimes y) *(x s \otimes y s)\rangle$ case-4 $)$
qed
theorem is-tangle-make-vert-right:
(is-tangle-diagram xs)
$\Longrightarrow$ is-tangle-diagram $(x s \otimes$ (basic (make-vert-block $n))$ )
proof (induct xs)
case (basic xs)
show ?case by auto
next
case (prod $x x s$ )
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 $((x \otimes($ make-vert-block 0$)) * x s)$ using is-tangle-diagram.simps(2) by (metis calculation prod.prems)
then have is-tangle-diagram $((x * x s) \otimes($ basic (make-vert-block 0$)))$ by auto
then show ?thesis using 0 by (metis)
next
case (Suc k)
have codomain-block (make-vert-block $(k+1))=\operatorname{int}(k+1)$
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 $(k+1)$ )))

$$
=\text { make-vert-block }(k+1)
$$

by auto
moreover have codomain-wall (basic (make-vert-block $(k+1)$ )) $>0$
using make-vert-block.simps codomain-wall.simps Suc-eq-plus1
codomain-make-vert of-nat-0-less-iff zero-less-Suc
by metis
ultimately have $1:(x * x s) \otimes($ basic (make-vert-block $(k+1)))$
$=(x \otimes($ make-vert-block $(k+1))) *(x s \otimes($ basic $($ make-vert-block $(k+1))))$
using tensor.simps(2) by $\operatorname{simp}$
have domain-wall (xs $\otimes($ basic (make-vert-block $(k+1)))$ )

$$
=\text { domain-wall xs }+ \text { domain-wall (basic }(\text { make-vert-block }(k+1)))
$$

using tensor-domain-wall-additivity by auto
then have 2:
domain-wall $(x s \otimes($ basic (make-vert-block $(k+1))))$

$$
=(\text { domain-wall xs })+\operatorname{int}(k+1)
$$

using domain-make-vert domain-wall.simps(1) by auto
moreover have 3: codomain-block ( $x \otimes$ (make-vert-block $(k+1))$ ) $=$ codomain-block $x+\operatorname{int}(k+1)$
using codomain-additive codomain-make-vert by (metis)
have is-tangle-diagram ( $x * x s$ )
using prod.prems by auto
then have 4 :codomain-block $x=$ domain-wall $x s$
using is-tangle-diagram.simps(2) by metis
from 234 have
domain-wall $(x s \otimes$ (basic (make-vert-block $(k+1))))$
$=$ codomain-block $(x \otimes($ make-vert-block $(k+1)))$
by auto
moreover have is-tangle-diagram (xs $\otimes$ (basic (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 $((x * x s) \otimes$ (basic (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 $x s) \Longrightarrow$ is-tangle-diagram $(($ basic $($ make-vert-block $n)) \otimes x s)$
proof (induct xs)
case (basic xs)
show ?case by auto
next
case $(\operatorname{prod} x x s)$
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$) \otimes x) * x s)$ using is-tangle-diagram.simps(2) by (metis calculation prod.prems)
then have is-tangle-diagram $(($ basic (make-vert-block 0$)) \otimes(x * x s))$ by auto
then show ?thesis using 0 by (metis)
next
case (Suc k)
have codomain-block (make-vert-block $(k+1))=\operatorname{int}(k+1)$
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 $(k+1))$ ))

$$
=\text { make-vert-block }(k+1)
$$

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 $(k+1))) \otimes(x * x s)$

```
=((make-vert-block (k+1)) \otimesx)*((basic (make-vert-block (k+1)))
```

xs)
using tensor.simps(3) by simp
have domain-wall ((basic (make-vert-block $(k+1))) \otimes x s)$
$=$ 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))) \otimesxs)
                =(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)) \otimesx)
        = 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.prems by auto
    then have 4:codomain-block x= domain-wall xs
            using is-tangle-diagram.simps(2) by metis
        from 2 3 4 have
        domain-wall ((basic (make-vert-block (k+1))) \otimesxs)
                = codomain-block ((make-vert-block (k+1))\otimesx)
            by auto
    moreover have is-tangle-diagram ((basic (make-vert-block (k+1))) \otimesxs)
    using prod.hyps prod.prems 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 $y) \neq 0 \Longrightarrow$
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
$y())))) \Longrightarrow$
(is-tangle-diagram $(x * x s) \wedge$ is-tangle-diagram (basic $y) \longrightarrow$ is-tangle-diagram $(x$

* xs $\otimes$ basic y))
proof -
assume $A$ : (codomain-block $y) \neq 0$
assume $B$ :
is-tangle-diagram (xs)
$\wedge$ is-tangle-diagram ((basic (make-vert-block (nat (codomain-block y)))))
is-tangle-diagram $(x s \otimes(($ basic $($ make-vert-block $($ nat $($ codomain-block $y))))))$
have is-tangle-diagram $(x * x s) \wedge$ is-tangle-diagram (basic $y) \longrightarrow$ is-tangle-diagram
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)
^(is-tangle-diagram (basic (make-vert-block (nat (codomain-block y)))))
    \longrightarrow \text { is-tangle-diagram (xs @ basic (make-vert-block (nat (codomain-block y)))))}
    \Longrightarrow
    is-tangle-diagram (x*xs)^ is-tangle-diagram (basic y)}
    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 $(x * x s) \wedge$ 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 $(x * x s) \wedge$ is-tangle-diagram (basic $y$ )
$\longrightarrow$
domain-wall $(x s \otimes($ basic $($ make-vert-block $($ nat $($ codomain-block $y)))))$
$=$ codomain-block $(x \otimes y)$
by (metis 12 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))))
$\Longrightarrow$
is-tangle-diagram $(x * x s) \wedge$ is-tangle-diagram (basic $y) \longrightarrow$
is-tangle-diagram $((x \otimes y) *(x s \otimes$ (basic (make-vert-block (nat (codomain-block
y)))) ))
using is-tangle-diagram.simps(2) by auto
then have
is-tangle-diagram $(x * x s) \wedge$ is-tangle-diagram (basic $y) \longrightarrow$
is-tangle-diagram $((x * x s) \otimes($ basic $y))$
by (metis Tangle-Algebra. 2 〈
codomain-block $y \neq 0$ > is-tangle-make-vert-right)
then show ?thesis by auto
qed
lemma simp2:

```
(codomain-block \(x) \neq 0\)
\(\Longrightarrow\)
    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 \((\operatorname{codomain-block~} x)))) \otimes y s)\)
    \(\Longrightarrow\)
    (is-tangle-diagram (basic \(x\) )
    \(\wedge\) is-tangle-diagram ( \(y * y s\) )
        \(\longrightarrow\) is-tangle-diagram \(((\) basic \(x) \otimes(y * y s)))\)
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 \((y s) \longrightarrow\)
        is-tangle-diagram
                            \(((\) basic \((\) make-vert-block \((\) nat (codomain-block \(x)))) \otimes y s)\)
    have is-tangle-diagram (basic \(x) \wedge\) is-tangle-diagram \((y * y s)\)
                                    \(\longrightarrow\) 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 \((\operatorname{codomain-block} x)))))\)
    \(\longrightarrow\) is-tangle-diagram \(((\) basic (make-vert-block \((\) nat \((\) codomain-block \(x)))) \otimes\)
ys))
    \(\Longrightarrow\)
    is-tangle-diagram \((\) basic \(x) \wedge\) is-tangle-diagram \((y * y s) \longrightarrow\)
        is-tangle-diagram
                        \(((\) basic \((\) make-vert-block \((\) nat \((\operatorname{codomain-block} x)))) \otimes y s)\)
        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 \((y * y s) \longrightarrow\)
            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 \(x)\) )))
\(\otimes y s)\)
```

```
            \(=\) domain-wall (basic (make-vert-block (nat (codomain-block \(x))\) )
```

            \(=\) domain-wall (basic (make-vert-block (nat (codomain-block \(x))\) )
                        + domain-wall ys
                        + domain-wall ys
    using tensor-domain-wall-additivity by auto
    moreover then have is-tangle-diagram (basic $x) \wedge$ is-tangle-diagram $(y * y s) \longrightarrow$
domain-wall $(($ basic (make-vert-block $($ nat $($ codomain-block $x)))) \otimes y s)$
$=$ codomain-block $(x \otimes y)$
by (metis 12 calculation(4))
ultimately have

```

\section*{(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)
```

$\Longrightarrow$
is-tangle-diagram $($ basic $x) \wedge i s$-tangle-diagram $(y * y s)$
is-tangle-diagram $((x \otimes y) *(($ basic (make-vert-block (nat (codomain-block

```
\(x)))(\otimes y s))\)
    using is-tangle-diagram.simps(2) by auto
then have
    is-tangle-diagram (basic \(x) \wedge\) is-tangle-diagram \((y * y s) \longrightarrow\)
    is-tangle-diagram \(((\) basic \(x) \otimes(y * y s))\)
    by (metis Tangle-Algebra. 3 A B)
then show?thesis by auto
qed
lemma simp3:
    is-tangle-diagram \(x s \wedge\) is-tangle-diagram ys \(\longrightarrow\) is-tangle-diagram \((x s \otimes y s)\)
    \(\Longrightarrow\)
    is-tangle-diagram \((x * x s) \wedge\) is-tangle-diagram \((y * y s)\)
    \(\longrightarrow\) is-tangle-diagram \((x * x s \otimes y * y s)\)
proof-
    assume \(A\) : is-tangle-diagram xs \(\wedge\) is-tangle-diagram ys \(\longrightarrow\) is-tangle-diagram (xs
\(\otimes y s)\)
    have is-tangle-diagram \((x * x s) \longrightarrow\) (codomain-block \(x=\) domain-wall \(x s)\)
    using is-tangle-diagram.simps(2) by auto
    moreover have is-tangle-diagram \((y * y s) \longrightarrow\) (codomain-block \(y=\) domain-wall
ys)
    using is-tangle-diagram.simps(2) by auto
ultimately have is-tangle-diagram \((x * x s) \wedge\) is-tangle-diagram ( \(y * y s\) )
            \(\longrightarrow\) codomain-block \((x \otimes y)=\) domain-wall \((x s \otimes y s)\)
        using codomain-additive tensor-domain-wall-additivity by auto
moreover have is-tangle-diagram \((x * x s) \wedge\) is-tangle-diagram ( \(y * y s\) )
            \(\longrightarrow\) is-tangle-diagram ( \(x s \otimes y s\) )
        using \(A\) is-tangle-diagram.simps(2) by auto
    moreover have \((x * x s) \otimes(y * y s)=(x \otimes y) *(x s \otimes y s)\)
        using tensor.simps(4) by auto
    ultimately have is-tangle-diagram \((x * x s) \wedge\) is-tangle-diagram \((y * y s)\)
            \(\longrightarrow\) is-tangle-diagram \(((x * x s) \otimes(y * y s))\)
        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 $z)) \wedge($ is-tangle-diagram $($ basic $w))$
$\longrightarrow$ is-tangle-diagram $(($ basic $z) \otimes($ basic $w))$
show ? case by auto
next
fix $x$ xs $y$
let ?case $=($ is-tangle-diagram $(x * x s)) \wedge($ is-tangle-diagram (basic $y))$
$\longrightarrow$ is-tangle-diagram $((x * x s) \otimes($ 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 $x)) \wedge($ is-tangle-diagram $(y * y s))$
$\longrightarrow$ is-tangle-diagram $(($ basic $x) \otimes(y * y s))$
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 $x y x s$ ys
assume $A$ : is-tangle-diagram $x s \wedge$ is-tangle-diagram ys $\longrightarrow$ is-tangle-diagram (xs
$\otimes y s)$
let ?case $=$
is-tangle-diagram $(x * x s) \wedge$ is-tangle-diagram $(y * y s) \longrightarrow i s$-tangle-diagram
$(x * x s \otimes y * y s)$
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 $(x \otimes y)$
using assms is-tangle-diagramness by auto
definition Tensor-Tangle::Tangle-Diagram $\Rightarrow$ Tangle-Diagram $\Rightarrow$ Tangle-Diagram
(infixl $\circ 65$ )
where
Tensor-Tangle $x y=$
Abs-Tangle-Diagram $(($ Rep-Tangle-Diagram $x) \otimes($ Rep-Tangle-Diagram y) $)$
theorem well-defined-compose:
assumes is-tangle-diagram $x$
and is-tangle-diagram y
shows $($ Abs-Tangle-Diagram $x) \otimes($ Abs-Tangle-Diagram $y)=($ Abs-Tangle-Diagram

```
```

$(x \otimes y))$
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 \(\forall x y .(x, y) \in\{(x, y) . R x y\}^{*} \longrightarrow(y, x) \in\{(x, y) . R x y\}^{*}\)
proof-
have \(R x y \longrightarrow R y x\) by (metis assms symp \(D\) )
then have \((x, y) \in\{(x, y) . R x y\} \longrightarrow(y, x) \in\{(x, y) . R x y\}\) by auto
then have \(2: \forall x y .(x, y) \in\{(x, y) . R x y\} \longrightarrow(y, x) \in\{(x, y) . R x y\}\)
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 \(\forall x y .(x, y) \in\{(x, y) . R x y\}^{*} \longrightarrow(y, x) \in\{(x\), y). \(R x y\}^{*}\)
shows symp \(R^{\text {** }}\)
unfolding symp-def Enum.rtranclp-rtrancl-eq assms by (metis assms)
lemma symmetry3: assumes symp \(R\) shows symp \(R^{\wedge}\) ** using assms symmetry 1 symmetry2 by metis
lemma symm-trans: assumes symp \(R\) shows symp \(R \wedge++\) by (metis assms rtranclpD symmetry3 symp-def tranclp-into-rtranclp)
end

\section*{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\# []\())\)
- (basic (cap\#vert\#[])))
abbreviation right-under::wall
where
right-under \(\equiv((\) basic \((\) vert\#cup\# []\()) \circ(\) basic \((\) under\#vert\# []\())\)
- (basic (vert\#cap\#[])))
abbreviation left-under::wall
where
left-under \(\equiv((\) basic (cup\#vert\#[])) \(\circ(\) basic \((\) vert\#under\#[]))
- (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-positive-straighten \(x y \equiv((x=\) right-over \() \wedge(y=\) straight-line \())\)
definition uncross-negative-flip::relation
where
uncross-negative-flip \(x y \equiv((x=\) right-under \() \wedge(y=\) left-under \())\)
definition uncross-negative-straighten::relation
where
uncross-negative-straighten \(x y \equiv((x=\) left-under \() \wedge(y=\) straight-line \())\)
definition uncross
where
uncross \(x y \equiv((\) uncross-positive-straighten \(x y) \vee(\) uncross-positive-flip \(x y)\)
\(\vee(\) uncross-negative-straighten \(x y) \vee(\) uncross-negative-flip \(x y))\)
swing begins
abbreviation \(r\)-over-braid:: wall
where
```

r-over-braid \equiv((basic ((over\#vert\#[]))\circ(basic ((vert\#over\#[])))
\circ(basic (over\# vert\#[]))))

```
abbreviation l-over-braid::wall
where
\(\begin{aligned} & l \text {-over-braid } \equiv(\text { basic }(\text { vert \#over\#[])) } \circ(\text { basic }(\text { over \#vert\#[]) }) \\ & \circ(\text { basic }(\text { vert\#over\#[])) })\end{aligned}\)
```

abbreviation r-under-braid::wall
where
$r$-under-braid $\equiv(($ basic $(($ under\#vert\# []$)) \circ($ basic $(($ vert\#under\#[]))) $\circ($ basic $($ under\# vert\#[]))))

```
abbreviation l-under-braid::wall
where
\(l\)-under-braid \(\equiv(\) basic \((\) vert\#under\#[]))○(basic (under\#vert\#[])) \(\circ\) (basic (vert\#under\#[]))
definition swing-pos::wall \(\Rightarrow\) wall \(\Rightarrow\) bool
where
swing-pos \(x y \equiv(x=r\)-over-braid \() \wedge(y=l\)-over-braid \()\)
definition swing-neg::wall \(\Rightarrow\) wall \(\Rightarrow\) bool
where
swing-neg \(x y \equiv(x=r\)-under-braid \() \wedge(y=l\)-under-braid \()\)
definition swing::relation
where
swing \(x y \equiv(\) swing-pos \(x y) \vee(\) swing-neg \(x y)\)
pull begins
definition pull-posneg::relation
where
pull-posneg \(x y \equiv((x=((\) basic \((\) over\# \(\#])) \circ(\) basic \(\quad(\) under \(\#[]))))\)
\(\wedge(y=((\) basic \((\) vert\#vert\#[]))) \()\)
definition pull-negpos::relation
where
pull-negpos \(x y \equiv((x=((\) basic \((\) under \(\#[])) \circ(\) basic \(\quad(\) over \(\#[]))))\)
\[
\wedge(y=((\text { basic }(\text { vert } \# \text { vert } \#[])))
\]
\[
\circ(\text { basic }((\text { vert \#vert\#[]))))) }
\]
pull definition
definition pull::relation
where
pull \(x y \equiv((\) pull-posneg \(x y) \vee(\) pull-negpos \(x y))\)
linkrel-pull ends
linkrel-straighten
definition straighten-topdown:: relation
where
straighten-topdown \(x\) y \(\equiv((x=((\) basic \(((v e r t \#\) cup \(\#])))\)
\(\circ(\) basic \(((\) cap\#vert\#[])))))
\(\wedge(y=((\) basic \((\) vert\#[]))○(basic \((\) vert\#[])))))
definition straighten-downtop::relation
where
straighten-downtop \(x y \equiv((x=((\) basic \(((\) cup\# vert \(\#[])))\)
\(\circ(\) basic \(((\) vert\# cap\#[])))))
\(\wedge(y=((\) basic \((\) vert\# []\()) \circ(\) basic \((\) vert\#[]) \())))\)
definition straighten
definition straighten::relation
where
straighten \(x y \equiv((\) straighten-topdown \(x y) \vee(\) straighten-downtop \(x y))\)
straighten ends
rotate moves
definition rotate-toppos::relation

\section*{where}
\[
\begin{array}{r}
\text { rotate-toppos } x y \equiv((x=((\text { basic }((\text { vert \#over\#[]))))} \\
\circ(\text { basic }((\text { cap\# vert \#[]))))))} \\
\wedge(y=((\text { basic }((\text { under } \# \text { vert } \#[])) \\
\circ(\text { basic }((v e r t \# \text { cap } \#[])))))))
\end{array}
\]
definition rotate-topneg::wall \(\Rightarrow\) wall \(\Rightarrow\) bool where
\[
\begin{array}{r}
\text { rotate-topneg } x y \equiv((x=((\text { basic }((\text { vert \#under\# } \#])))) \\
\circ(\text { basic }((\text { cap\# vert } \#[]))))) \\
\wedge(y=((\text { basic }((\text { over\#vert\#[])) } \\
\circ(\text { basic }((\text { vert } \# \text { cap } \#[])))))))
\end{array}
\]
```

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

```
definition rotate-downneg:: wall \(\Rightarrow\) wall \(\Rightarrow\) bool where
\[
\begin{array}{r}
\text { rotate-downneg } x y \equiv((x=((\text { basic }(\text { cup\#vert } \#[])) \\
\circ(\text { basic }(\text { vert\#under } \#[])))) \\
\wedge(y=((\text { basic }((\text { vert } \# \text { cup } \#[]))) \\
\circ(\text { basic }((\text { over\#vert } \#[]))))))
\end{array}
\]
rotate definition
definition rotate:: wall \(\Rightarrow\) wall \(\Rightarrow\) bool

\section*{where}
rotate \(x y \equiv((\) rotate-toppos \(x y) \vee(\) rotate-topneg \(x y)\)
\(\vee(\) rotate-downpos \(x y) \vee(\) rotate-downneg \(x y))\)
rotate ends
Compress - Compress has two levels of equivalences. It is a composition of Compress-null, compbelow and compabove. compbelow and compabove are further written as disjunction of many other relations. Compbelow refers to when the bottom row is extended or compressed. Compabove refers to when the row above is extended or compressed
definition compress-top \(1::\) wall \(\Rightarrow\) wall \(\Rightarrow\) bool
where
compress-top1 \(x y \equiv \exists B .((x=(\) basic \((\) make-vert-block \((\) nat \((\) domain-wall \(B)))) \circ\) B)
```

    \(\wedge(y=B) \wedge(\) codomain-wall \(B \neq 0)\)
    \(\wedge(\) is-tangle-diagram \(B))\)
    ```
definition compress-bottom \(1::\) wall \(\Rightarrow\) wall \(\Rightarrow\) bool where
compress-bottom1 \(x y \equiv \exists B .((x=B \circ\) (basic (make-vert-block (nat (codomain-wall B))) ))
\[
\begin{aligned}
& \wedge(y=B)) \wedge(\text { domain-wall } B \neq 0) \\
& \wedge(\text { is-tangle-diagram } B)
\end{aligned}
\]
definition compress-bottom::wall \(\Rightarrow\) wall \(\Rightarrow\) bool
where
compress-bottom \(x y \equiv \exists B .((x=B \circ\) (basic (make-vert-block (nat (codomain-wall B) )) ))
\(\wedge(y=((\) basic \(([]) \circ B))) \wedge(\) domain-wall \(B=0)\)
\(\wedge(\) is-tangle-diagram \(B))\)
definition compress-top::wall \(\Rightarrow\) wall \(\Rightarrow\) bool
where
compress-top \(x y \equiv \exists B .((x=(\) basic \((\) make-vert-block \((\) nat \((\) domain-wall \(B)))) \circ\) B)
\[
\begin{aligned}
& \wedge(y=(B \circ(\text { basic }([])))) \wedge(\text { codomain-wall } B=0) \\
& \wedge(\text { is-tangle-diagram } B))
\end{aligned}
\]
definition compress::wall \(\Rightarrow\) wall \(\Rightarrow\) bool
where
compress \(x y=((\) compress-top \(x y) \vee(\) compress-bottom \(x y))\)
slide relation refer to the relation where a crossing is slided over a vertical strand
definition slide:: wall \(\Rightarrow\) wall \(\Rightarrow\) bool where
slide \(x y \equiv \exists B .((x=((\) basic \((\) make-vert-block \((\) nat \((\) domain-block \(B)))) \circ(\) basic B)))
\[
\wedge(y=((\text { basic B }) \circ(\text { basic }(\text { make-vert-block }(\text { nat }(\text { codomain-block B) })))))
\]
\(\wedge((\) domain-block \(B) \neq 0))\)
linkrel-definition
definition linkrel::wall \(=>\) wall \(\Rightarrow\) bool
where
linkrel \(x y=((\) uncross \(x y) \vee(\) pull \(x y) \vee(\) straighten \(x y)\)
\(\vee(\) swing \(x y) \vee(\) rotate \(x y) \vee(\) compress \(x y) \vee(\) slide \(x y))\)
definition framed-uncross::wall \(\Rightarrow\) wall \(\Rightarrow\) bool
where
framed-uncross \(x\) y \(\equiv((\) uncross-positive-flip \(x y) \vee(\) uncross-negative-flip \(x y))\)
definition framed-linkrel::wall \(=>\) wall \(\Rightarrow\) bool

\section*{where}
framed-linkrel \(x y=((\) framed-uncross \(x y) \vee(\) pull \(x y) \vee(\) straighten \(x y)\)
\(\vee(\) swing \(x y) \vee(\) rotate \(x y) \vee(\) compress \(x y) \vee(\) slide \(x y))\)
lemma framed-uncross-implies-uncross: (framed-uncross \(x y) \Longrightarrow(u n c r o s s ~ x y)\) by (auto simp add: framed-uncross-def uncross-def)
end

\section*{7 Link_Algebra: Defining equivalence of tangles and links}
```

theory Link-Algebra
imports Tangles Tangle-Algebra Tangle-Moves
begin
inductive Tangle-Equivalence :: wall }=>\mathrm{ wall }=>\mathrm{ bool (infixl ~ 64)
where
refl [intro!, Pure.intro!, simp]: a ~ a
|equality [Pure.intro]: linkrel a b \Longrightarrow a ~ b
|domain-compose:(domain-wall a = 0)^(is-tangle-diagram a)\Longrightarrow < < ((basic
[])○a)
|codomain-compose:(codomain-wall a = 0)^(is-tangle-diagram a)\Longrightarrowa~ (a\circ
(basic []))
|compose-eq:((B::wall) ~ D) ^ ((A::wall ) ~ C)
\wedge(is-tangle-diagram A)^(is-tangle-diagram B)
\wedge(is-tangle-diagram C)^(is-tangle-diagram D)
\wedge ( domain-wall B)=(codomain-wall A)
\wedge ( domain-wall D)=(codomain-wall C)
\Longrightarrow ( ( A : : w a l l ) \circ B ) \sim ( C \circ D )
|trans: }\mp@subsup{A}{}{~}B\Longrightarrow\mp@subsup{B}{}{~}C\LongrightarrowA~
|sym:A~}B\LongrightarrowB~~
|tensor-eq: ((B::wall) ~ D) ^((A::wall ) ~ C) ^(is-tangle-diagram A )^(is-tangle-diagram
B)
\wedge(is-tangle-diagram C)^(is-tangle-diagram D)\Longrightarrow((A::wall) \otimesB) ~}(C\otimesD
inductive Framed-Tangle-Equivalence :: wall }=>\mathrm{ wall }=>\mathrm{ bool (infixl ~f 64)
where
refl [intro!, Pure.intro!, simp]: a ~f a
|equality [Pure.intro]: framed-linkrel a b\Longrightarrowa~fb
|domain-compose:(domain-wall }a=0)\wedge(\mathrm{ is-tangle-diagram a) "a
[])○a)
|codomain-compose:(codomain-wall a = 0) ^(is-tangle-diagram a)\Longrightarrowa}~f(a
(basic []))

```
```

|compose-eq:((B::wall) ~f D) ^ ((A::wall) ~f C)
\wedge(is-tangle-diagram A)^(is-tangle-diagram B)
\wedge(is-tangle-diagram C)}\wedge(\mathrm{ is-tangle-diagram D)
\wedge ( domain-wall B)=(codomain-wall A)
\wedge ( domain-wall D)=(codomain-wall C)
\Longrightarrow ( ( A : : wall ) ○B) ~f (C○D)
|trans: A~fB\Longrightarrow 林 fC \Longrightarrow A ~
|sym:A~f B\LongrightarrowB~}
|tensor-eq: ((B::wall) ~ f D ) ^((A::wall ) ~fC) ^(is-tangle-diagram A )^(is-tangle-diagram
B)
\wedge(is-tangle-diagram C)^(is-tangle-diagram D )\Longrightarrow((A::wall) \otimesB) ~
definition Tangle-Diagram-Equivalence::Tangle-Diagram }=>\mathrm{ Tangle-Diagram }
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 }=>\mathrm{ Link-Diagram }=>\mathrm{ 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 equivpI)
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 equivpI)
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
\[
\equiv(\text { Rep-Tangle-Diagram T1 }) \sim(\text { Rep-Tangle-Diagram T2 })
\]
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

```

\section*{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:
\(((\) basic \((\) cup\# []\()) \circ(\) basic (vert \# vert \# [])) \() \sim\)
\(((\) basic [])○(basic (cup\#[])))
proof -
have domain-wall (basic (cup \# [])) \(=0\)
by auto
moreover have codomain-wall (basic \((\) cup \# [1)) \(=2\) 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\#[])) ○ (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\# []\()) \circ(\) 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\# []\()) \circ(\) basic \((\) vert \# vert \# [])) \() \sim\)
\(((\) basic []) \(\circ(\) 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 \([c u p]) \circ(\) basic \([c a p])\)
lemma uncross-straighten-left-over:left-over \(\sim\) 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 \(\sim\) left-over
using Tangle-Equivalence.equality by auto
then have 1:left-over \(\sim\) 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 12 transitive by blast
qed
theorem Example:
x~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 \otimes straight-line) ~ (straight-line \otimes 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 12 by (metis Tangle-Equivalence.tensor-eq)
qed
then have 4:
((basic (cup\#[])) ○ (left-over \otimes straight-line))
~ ((basic (cup\#[])) ○(straight-line \otimes straight-line))
proof-
have is-tangle-diagram (left-over \otimes straight-line)
by auto
moreover have is-tangle-diagram (straight-line \otimes straight-line)
by auto
moreover have is-tangle-diagram (basic (cup\#[]))
by auto
moreover have domain-wall (left-over \otimes straight-line) = (codomain-wall (basic
(cup\#[])))
unfolding domain-wall-def by auto
moreover have domain-wall (straight-line }\otimes\mathrm{ straight-line) }=(\mathrm{ 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])\circ (straight-line \otimes straight-line)

```
proof-
have 0 :
\((\) basic \(([\) cup \(])) \circ(\) straight-line \(\otimes\) straight-line \()=(\) basic \([\) cup \(]) \circ(\) basic \([\) vert, vert \(])\)
- (basic \([\) vert,\(v e r t]) \circ(\) basic \([\) vert,vert \(])\)
by auto
let \(? x=(\) basic \((\operatorname{cup} \#[]))\)
\(\circ(\) basic \((\) vert\#vert\#[])) \(\circ(\) basic (vert\#vert\#[]))
- (basic (vert\#vert\#[]))
let \(? x 1=(\) basic \((\) vert\#vert\#[])) \((\) basic \((\) vert\#vert\#[]))
have 1:? \(x^{\sim}((\) basic (cup\#[])) \(\circ\) ? \(x 1)\)
proof-
have \((\) basic \((\) cup\# []\()) \circ(\) basic (vert \# vert \# []) \() \sim(\) basic (cup\#[])) using cup-compress by auto
moreover have is-tangle-diagram (basic (cup\#[]))
using is-tangle-diagram-def by auto
moreover have is-tangle-diagram ((basic (cup\#[]))○(basic (vert \# vert \# []))) using is-tangle-diagram-def by auto
moreover have is-tangle-diagram (?x1) by auto
moreover have ? \(x 1 \sim\) ? \(x 1\)
using refl by auto
moreover have
codomain-wall \((\) basic \((\) cup\# []\())=\) domain-wall \((\) basic \((\) vert\#vert\#[]))
by auto
moreover have (basic (cup\#[])) ~ (basic (cup\#[]))
using refl by auto
ultimately show ?thesis
using compose-eq codomain-wall-compose compose-leftassociativity converse-composition-of-tangle-diagrams domain-wall-compose
by (metis Tangle-Equivalence.compose-eq is-tangle-diagram.simps(1))
qed
have 2: \(((\) basic \((\) cup\# [] \()) \circ\) ?x1 \() \sim(\) basic \((\) cup\# []\())\)
proof-
have
\(((\) basic \((\) cup \# []))○(basic (vert \# vert \# []))) \()(\) basic (vert \# vert \# [])) ~ ((basic(cup\#[]))○(basic(vert\#vert\#[])))
proof-
have \((\) basic \((\) cup\# []\()) \circ(\) basic (vert \# vert \# [])) \(\sim(\) basic (cup\#[])) using cup-compress by auto
moreover have (basic(vert\#vert\#[])) \(\sim(\operatorname{basic}(\) vert\#vert\#[])) using refl by auto
moreover have is-tangle-diagram (basic (cup\#[])) using is-tangle-diagram-def by auto
moreover have is-tangle-diagram ((basic (cup\#[]))○(basic (vert \# vert \# []))) using is-tangle-diagram-def by auto
moreover have is-tangle-diagram ((basic(vert\#vert\#[]))) by auto

\section*{moreover have}
codomain-wall ((basic (cup\#[]))○ (basic(vert\#vert\#[])))
\[
=\text { domain-wall }(\text { basic }(\text { vert\#vert\#[])) }
\]
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\# []\()) \circ\) ? \(x 1)\) ~
\(((\operatorname{basic}(\operatorname{cup} \#[])) \circ(\operatorname{basic}(\) vert\#vert\#[])))
by auto
then show ?thesis using cup-compress trans
by (metis (full-types) Example.transitive)
qed
from 012 show ?thesis using trans transp-def trans compose-Nil
by (metis (opaque-lifting, no-types) Example.transitive)
qed
let \(? y=((\) basic \(([])) \circ(\) basic \((\) cup\# []\()))\)
let ? temp \(=(\) basic \((\) vert\#over\#vert\#[]))○(basic (cap\#vert\#vert\#[]))
have 45 : (left-over \(\otimes\) straight-line \()=\)
((basic (cup\#vert\#vert\#[])) ○?temp)
using tensor.simps by (metis compose-Nil concatenates-Cons concate-
nates-Nil)
then have 55:(basic (cup\#[])) \(\circ\) (left-over \(\otimes\) straight-line \()\)
\(=(\) basic \((\) cup\# []\()) \circ(\) basic \((\) cup\#vert\#vert\#[])) \(\circ\) ?temp
by auto

\section*{then have}
(basic (cup\#[])) ○ (basic (cup\#vert\#vert\#[]))
```

    =(basic }(([])\otimes(\mathrm{ cup#[])))०(basic ((cup#[])}\otimes(vert#vert#[])))
    ```
        using concatenate.simps by auto
then have 6 :
(basic (cup\#[])) ○ (basic (cup\#vert\#vert\#[]))
\(=((\) basic \(([])) \circ(\) basic \((\) cup\# []\()))\)
\(\otimes((\) basic \((\) cup\# []\()) \circ(\) basic \((\) vert\#vert\#[]) \())\)
using tensor.simps by auto
then have \(((\) basic \((\) cup\#[])) \(\circ(\) basic (vert\#vert\#[])))
\(\sim(\) basic ([])) \()(\) basic (cup\#[]))
using prelim-cup-compress by auto
moreover have \(((\) basic \(([])) \circ(\) basic (cup\# []\()))\)
\[
\sim((\text { basic }([])) \circ(\text { basic }(\text { cup\# }[])))
\]
using refl by auto
moreover have is-tangle-diagram ((basic (cup\#[])) \(\circ(\) basic \((\) vert\#vert\#[])))
by auto
moreover have is-tangle-diagram ((basic ([]))○(basic (cup\#[]))) by auto
ultimately have 7:?y \(\otimes((\) basic \((\) cup\# []\()) \circ(\) basic \((\) vert\#vert\#[])) \() \sim((? y) \otimes\)
(? \(?\) ) )
using tensor-eq cup-compress Nil-right-tensor is-tangle-diagram.simps(1)
reft
by (metis Tangle-Equivalence.tensor-eq)
then have \(((? y) \otimes(? y))=(\) basic \((([]) \otimes([])))\)
- ((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\#[])))
~ (basic ([])) ○(basic (cup\#cup\#[]))
using 7 by auto
moreover have (basic ([]))॰(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\#[])))
~ (basic (cup\#cup\#[]))
using trans by (metis (full-types) Example.transitive)
then have \((\operatorname{basic}(\) cup\#[])) \()(\) basic(cup\#vert\#vert\#[])) \(\sim(\) basic(cup\#cup\#[]))
by auto
moreover have ?temp ~ ?temp
using refl by auto
moreover have is-tangle-diagram ((basic(cup\#[]))o(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\#[]))o(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\#[]))○(basic(cup\#vert\#vert\#[]))) ○(?temp) ~ (basic(cup\#cup\#[])) \((\) ?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]) \(\circ\left(\right.\) ? \(\left.{ }^{\text {temp })}\right)\)
\[
\sim(\text { basic }[\text { cup }]) \circ(\text { straight-line } \otimes \text { straight-line })
\]
proof-
have ((basic [cup,cup]) ○ (?temp))
\[
\sim(\text { basic }[\text { cup }] \circ(\text { left-over } \otimes \text { straight-line }))
\]
using 855 compose-leftassociativity sym wall.simps Tangle-Equivalence.sym
compose-Nil
by (metis)
moreover have (basic [cup]) \(\circ(\) left-over \(\otimes\) straight-line \()\)
\[
\sim(\text { basic }[\text { cup }]) \circ(\text { straight-line } \otimes \text { straight-line })
\]
using 4 by auto
moreover have \((((\) basic \([\) cup,cup \(]) \circ(\) ?temp \())\)
\(\sim(\) basic \([c u p] \circ(\) left-over \(\otimes\) straight-line \()))\)
\(\wedge((\) basic \([\) cup \(]) \circ(\) left-over \(\otimes\) straight-line \()\) \(\sim(\) basic \([\) cup \(]) \circ(\) straight-line \(\otimes\) straight-line \())\)
\(\Longrightarrow\) ?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\#[])) \(\sim(\) basic (cap\#[]))
using refl by auto
moreover have is-tangle-diagram ((basic(cup\#cup\#[])) \(\circ(\) ?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))
\[
=\text { domain-wall (basic (cap \# [])) }
\]
by auto
moreover have codomain-wall (basic(cup\#[])) =domain-wall (basic (cap \# []))
by auto
ultimately have \(9:((\) basic (cup\#cup\#[])) \(\circ(\) ?temp \()) \circ(\) basic \((\) cap\# []\())\)
~ (basic (cup\#[])) ○ (basic (cap\#[]))
using Tangle-Equivalence.compose-eq by metis
let \(? z=((\operatorname{basic}(\) cup\#cup\#[])) \(\circ(\) basic \((v e r t \#\) over\#vert \#[]) \())\)
have 10:((basic(cup\#cup\#[])) \(\circ(\) ?temp \()) \circ(\) basic (cap\#[]))
\[
=? z \circ((\text { basic }(\text { cap\#vert } \# \text { vert \# }[])) \circ(\text { basic }(\text { cap\# }[])))
\]
by auto
then have 11:((basic(cap\#vert\#vert\#[])) \(\circ(\) basic \((\) cap\#[])))
\[
=((\text { basic }((\text { cap } \#[]) \otimes(\text { vert\#vert } \#[]))) \circ(\text { basic }(([]) \otimes(\text { cap\#[]))))})
\]
unfolding concatenate-def by auto
then have 12: ((basic(cap\#vert\#vert\#[])) \(\circ(\) basic \((\) cap\#[]) \())\)
\[
=((\text { basic }(\text { cap\# }[])) \circ(\text { basic }([]))) \otimes((\text { basic }(\text { vert } \# \text { vert\# } \#])) \circ(\text { basic }
\]
(cap\#[])))
using tensor.simps by auto
let \(? w=((\) basic \((\) cap\# [] \()) \circ(\) 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\#[]))) \(\sim\) ? \(w\) 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 ? \(w \sim\) ?w
using refl by auto
ultimately have \(14:(? w) \otimes((\) basic \((\) vert\#vert\#[]) \() \circ(\) basic \((\) cap\# []\())) \sim((? w) \otimes\) (?w))
using Tangle-Equivalence.tensor-eq by metis
then have \(((\) basic \((\) cap\#vert\#vert\#[])) \(\circ(\) basic \((\) cap\# []\())) \sim((? w) \otimes(? w))\)
using 13 by auto
moreover have \(((? w) \otimes(? w))=(\) basic \((\) cap\#cap\# []\()) \circ(\) basic \(([]))\)
using tensor.simps by auto
ultimately have \(\left(\left(\right.\right.\) basic \((\text { cap\#vert\#vert\#[]))○(basic }(\text { cap\#[]) }))^{\sim}(\) basic \((\) cap\#cap\#[]))○(basic ([]))
by auto
moreover have ?z ~ ? \(z\)
using refl by auto
moreover have domain-wall ((basic(cap\#cap\#[])) ○ (basic ([])))
\[
=\text { codomain-wall }(? z)
\]
by auto
moreover have domain-wall \((((\) basic(cap\#vert\#vert\#[])) \(\circ(\) basic \((\) cap\#[]) \()))\)
\[
=\text { codomain-wall }(? z)
\]
by auto
moreover have is-tangle-diagram ((basic(cap\#vert\#vert\#[])) \(\circ(\) basic \((\) cap\# []\()))\) by auto
moreover have is-tangle-diagram (?z)
by auto
moreover have is-tangle-diagram ((basic(cap\#cap\#[])) \(\circ(\) basic ([])))
by auto
ultimately have 14: (?z) ○ ((basic(cap\#vert\#vert\#[])) ○(basic (cap\#[])))
\[
\sim(? z) \circ((\text { basic }(\text { cap\#cap\# }[])) \circ(\text { basic }([])))(\text { is ? } a a \sim ? b b)
\]
using Tangle－Equivalence．compose－eq by metis
moreover have 15：（（？z）○（（basic（cap\＃cap\＃［］）））\(\circ(\) basic（［］））） \(\sim((? z) \circ(\) basic \((c a p \# c a p \#[])))\left(\right.\) is ？\(b b{ }^{\sim}\) ？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］）
\[
=\text { domain-wall (basic }[\text { vert, over, vert }] \circ \text { basic }[\text { cap, vert, vert }])>
\]

〈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 \((? a a \sim ? b b) \wedge(? b b \sim ? c c) \Longrightarrow ? a a \sim\) ？cc
using transitive by auto
then have 16：？aa \({ }^{\sim}\) ？cc
using 1415 by auto
then have 17：（（basic（cup\＃［］））○（basic（cap\＃［］）））～？aa
using 910 Tangle－Equivalence．trans Tangle－Equivalence．sym
by（metis（opaque－lifting，no－types））
have \((((\) basic \((\) cup\＃［］））○（basic \((\) cap\＃［］）））\(\sim\) ？aa \() \wedge(? a a \sim ~ ? c c)\)
\(\Longrightarrow\left((\right.\) basic \((\) cup\＃［］\()) \circ(\text { basic }(\text { cap\＃［］）}))^{\sim}\) ？cc
using transitive by auto
then have \(((\text { basic }(\text { cup\＃}[])) \circ(\text { basic }(\text { cap\＃}[])))^{\sim}\) ？cc using 1716 by auto
then show ？thesis using Tangle－Equivalence．sym by auto qed
end

\section*{9 Kauffman Matrix and Kauffman Bracket－Defi－ nitions and Properties}

\author{
theory Kauffman－Matrix \\ imports \\ Matrix－Tensor．Matrix－Tensor \\ Link－Algebra \\ HOL－Computational－Algebra．Polynomial \\ HOL－Computational－Algebra．Fraction－Field \\ begin
}

\section*{10 Rational Functions}
intpoly is the type of integer polynomials
type－synonym intpoly \(=\) int poly
lemma eval-pCons: poly (pCons 0 1) \(x=x\) using poly-1 poly-pCons by auto
lemma \(p\) Cons \(2:(p\) Cons 01\() \neq(1::\) int poly \()\)
using eval-pCons poly-1 zero-neq-one by metis
definition var-def: \(x=(p\) Cons 01\()\)
lemma non-zero: \(x \neq 0\) using var-def \(p\) Cons-eq-0-iff zero-neq-one by (metis)
rat_poly is the fraction field of integer polynomials. In other words, it is the type of rational functions
type-synonym rat-poly \(=\) intpoly fract
\(A\) is defined to be \(x / 1\), while \(B\) is defined to be \(1 / x\)
definition var-def1: \(A=\) Fract \(x 1\)
definition var-def2: \(B=\) Fract \(1 x\)
lemma assumes \(b \neq 0\) and \(d \neq 0\)
shows Fract \(a b=\) Fract \(c d \longleftrightarrow a * d=c * b\)
using eq-fract assms by auto
lemma \(A\)-non-zero: \(A \neq(0::\) rat-poly \()\)
unfolding var-def1
proof (rule ccontr)
assume \(0: \neg(\) Fract \(x 1 \neq(0::\) rat-poly \())\)
then have Fract \(x 1=(0::\) rat-poly \()\) by auto
moreover have \((0::\) rat-poly \()=\) Fract \((0::\) intpoly \()(1::\) intpoly \()\) by (metis Zero-fract-def)
ultimately have Fract \(x(1::\) intpoly \()=\) Fract \((0::\) intpoly \()(1::\) intpoly \()\) by auto
moreover have ( \(1::\) intpoly \() \neq 0\) by auto
ultimately have \(x *(1::\) intpoly \()=(0::\) intpoly \() *(1::\) intpoly \()\) using eq-fract by metis
then have \(x=(0::\) intpoly \()\)
by auto
then show False using non-zero by auto
qed
lemma mult-inv-non-zero:
assumes \((p::\) rat-poly \() \neq 0\)
and \(p * q=(1::\) rat-poly \()\)
```

shows q}\not=
using assms by auto
abbreviation rat-poly-times::rat-poly }=>\mathrm{ rat-poly }=>\mathrm{ rat-poly
where
rat-poly-times p q \equivp*q
abbreviation rat-poly-plus::rat-poly }=>\mathrm{ rat-poly }=>\mathrm{ rat-poly
where
rat-poly-plus p q\equivp+q
abbreviation rat-poly-inv::rat-poly }=>\mathrm{ 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 pr)(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)
    \(\operatorname{apply}\) (auto simp add:replicate-def rat-poly.row-length-def)
```

apply(auto simp add:scalar-prod)
done

```

\section*{abbreviation}
rat-polymat-tensor::rat-poly mat \(\Rightarrow\) rat-poly mat \(\Rightarrow\) rat-poly mat (infixl \(\otimes 65)\)

\section*{where}
rat-polymat-tensor \(p q \equiv\) rat-poly.Tensor \(p q\)
lemma assumes \((j:: n a t)\) div \(a=i\) div \(a\)
and \(j \bmod a=i \bmod a\)
shows \(j=i\)
proof -
have \(a *(j\) div \(a)+(j \bmod a)=j\)
using mult-div-mod-eq by simp
moreover have \(a *(i \operatorname{div} a)+(i \bmod a)=i\)
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)

\section*{11 Kauffman matrices}

We assign every brick to a matrix of rational polynmials
primrec brickmat::brick \(\Rightarrow\) rat-poly mat
where
brickmat vert \(=[[1,0],[0,1]]\)
\(\mid\) brickmat cup \(=[[0],[A],[-B],[0]]\)
|brickmat cap \(=[[0,-A, B, 0]]\)
\(\mid\) brickmat over \(=[[A, O, O, O]\),
\[
\begin{aligned}
& {[0,0, B, 0]} \\
& {[0, B, A-(B * B * B), 0]} \\
& [0,0,0, A]]
\end{aligned}
\]
\(\mid\) brickmat under \(=[[B, 0,0,0]\),
\[
\begin{aligned}
& {[0, B-(A * A * A), A, 0]} \\
& {[0, A, 0,0]} \\
& [0,0,0, B]]
\end{aligned}
\]
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: \(B \neq 0\)
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:
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:
rat-poly-times \((p-q) r=(\) rat-poly-times \(p r)-(\) rat-poly-times \(q r)\)
using minus-left-distributivity rat-poly.comm by metis
lemma equation:
rat-poly-plus
(rat-poly-times \(B(B-\) rat-poly-times (rat-poly-times \(A A) A))\)
(rat-poly-times \((A\) - rat-poly-times (rat-poly-times B B) B) A)
\[
=0
\]
proof -
have rat-poly-times (rat-poly-times \(A\) A) A
\[
=((A * A) * A)
\]
by auto
then have rat-poly-times \(B\) ( \(B\) - rat-poly-times (rat-poly-times \(A A\) ) \(A\) )
\[
=B * B-B *((A * A) * A)
\]
using minus-left-distributivity by auto
moreover have \(\ldots=B * B-(B *(A *(A * A)))\)
by auto
moreover have \(\ldots=B * B-((B * A) *(A * A))\)
by auto
moreover have \(\ldots=B * B-A * A\)
using inverse2 by auto
ultimately have 1 :
rat-poly-times \(B\) ( \(B\) - rat-poly-times (rat-poly-times \(A A) A\) )
\(=B * B-A * A\)
by auto
have rat-poly-times (rat-poly-times \(B\) ) \(B=(B * B) * B\)
by auto
then have
(rat-poly-times \((A-r a t-p o l y-t i m e s ~(r a t-p o l y-t i m e s ~ B ~ B) B) ~ A) ~\)
\[
=(A * A)-((B * B) * B) * A
\]
using minus-right-distributivity by auto
moreover have \(\ldots=(A * A)-((B * B) *(B * A))\)
by auto
moreover have \(\ldots=(A * A)-(B * B)\)
using inverse2 by auto
ultimately have 2:
(rat-poly-times \((A-r a t-p o l y-t i m e s ~(r a t-p o l y-t i m e s ~ B ~ B) ~ B) ~ A) ~\)
\[
=(A * A)-(B * B)
\]
by auto
have \(B * B-A * A+(A * A)-(B * B)=0\)
by auto
with 12 show ?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) \(=2 \wedge\) length (brickmat vert)
\(=2\)
using rat-poly.row-length-def by auto
lemma cup-dim:rat-poly.row-length (brickmat cup) \(=1\) and length (brickmat cup)
\(=4\)
using rat-poly.row-length-def by auto
lemma cap-dim:rat-poly.row-length (brickmat cap) \(=4\) and length (brickmat cap)
\(=1\)
using rat-poly.row-length-def by auto
lemma over-dim:rat-poly.row-length (brickmat over) \(=4\) and length (brickmat over \()=4\)
using rat-poly.row-length-def by auto
lemma under-dim:rat-poly.row-length (brickmat under) \(=4\) and length (brickmat under) \(=4\)
using rat-poly.row-length-def by auto
```

lemma mat-vert:mat 22 (brickmat vert)
unfolding mat-def Ball-def vec-def by auto
lemma mat-cup:mat 14 (brickmat cup)
unfolding mat-def Ball-def vec-def by auto
lemma mat-cap:mat 41 (brickmat cap)
unfolding mat-def Ball-def vec-def by auto
lemma mat-over:mat 44 (brickmat over)
unfolding mat-def Ball-def vec-def by auto
lemma mat-under:mat 44 (brickmat under)
unfolding mat-def Ball-def vec-def by auto
primrec rowlength::nat $\Rightarrow$ nat
where
rowlength $0=1$
$\mid$ rowlength $($ Suc $k)=2 *($ Suc $k)$
lemma $($ rat-poly.row-length $($ brickmat d $))=\left(\right.$ 2^ $^{\text {(nat }}($ domain d $\left.\left.)\right)\right)$
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 $\Rightarrow$ rat-poly mat
where
blockmat [] = [[1]]
$\mid$ blockmat $(l \# l s)=($ brickmat $l) \otimes($ blockmat $l s)$
lemma blockmat $[a]=$ brickmat a
unfolding blockmat.simps rat-poly.Tensor-right-id by auto
lemma nat-sum:
assumes $a \geq 0$ and $b \geq 0$
shows nat $(a+b)=($ nat $a)+($ nat $b)$
using assms by auto
lemma rat-poly.row-length $($ blockmat $l s)=\left(\mathfrak{Z}^{\wedge}(\right.$ nat $(($ domain-block ls $\left.)))\right)$
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) \(=2{ }^{\wedge}\) nat (domain-block \(l s\) )
using Cons by auto
then have rat-poly.row-length (blockmat (l\#ls))
\[
\begin{aligned}
= & (\text { rat-poly.row-length }(\text { brickmat l) }) \\
& *(\text { rat-poly.row-length (blockmat ls }))
\end{aligned}
\]
using blockmat.simps rat-poly.row-length-mat by auto
moreover have \(\ldots=2 *\left(2^{\text {^ nat }}\right.\) (domain-block ls))
using rat-poly.row-length-def Cons vert by auto
moreover have \(\ldots=2^{\wedge}(1+\) nat (domain-block ls \(\left.)\right)\)
using domain-block.simps by auto
moreover have \(\ldots=\mathscr{Z}^{\wedge}(\) nat \((\) domain \(l)+\) nat (domain-block ls) \()\) using domain.simps vert by auto
moreover have \(\ldots=\mathcal{L}^{\wedge}(\) nat \((\) domain \(l+\) domain-block ls))
using Suc-eq-plus1-left Suc-nat-eq-nat-zadd1
calculation(4) domain.simps(1) domain-block-non-negative vert
by (metis)
moreover have \(\ldots=2^{\wedge}(\) nat \((\) domain-block \((l \# l s)))\)
using domain-block.simps by auto
ultimately show ?thesis by metis
next
case over
have rat-poly.row-length (blockmat ls) \(=2^{\wedge}\) nat (domain-block ls) using Cons by auto
then have rat-poly.row-length (blockmat (l\#ls))
\[
\begin{aligned}
= & (\text { rat-poly.row-length (brickmat l) }) \\
& *(\text { rat-poly.row-length (blockmat ls }))
\end{aligned}
\]
using blockmat.simps rat-poly.row-length-mat by auto
also have \(\ldots=4 *\left(\right.\) \(^{\text {^ }}\) nat (domain-block ls) \()\)
using rat-poly.row-length-def Cons over by auto
also have \(\ldots=2\) ^(2 + nat (domain-block ls \()\) )
using domain-block.simps by auto
also have \(\ldots=\mathcal{L}^{\wedge}(\) nat \((\) domain \(l)+\) nat (domain-block ls) \()\)
using domain.simps over by auto
also have \(\ldots=\mathcal{Z}^{\wedge}(\) nat (domain \(l+\) domain-block \(\left.l s)\right)\)
by (simp add: nat-add-distrib domain-block-nonnegative over)
also have \(\ldots=\) 2^ \(^{\text {(nat }}\) (domain-block \(\left.(l \# l s)\right)\) )
by \(\operatorname{simp}\)
finally show ?thesis .
next
case under
have rat-poly.row-length (blockmat ls) = 2 へ nat (domain-block ls) using Cons by auto
then have rat-poly.row-length (blockmat (l\#ls))
\[
\begin{aligned}
= & (\text { rat-poly.row-length }(\text { brickmat l })) \\
& *(\text { rat-poly.row-length }(\text { blockmat ls }))
\end{aligned}
\]
using blockmat.simps rat-poly.row-length-mat by auto
also have \(\ldots=4 *\left(2^{\text {へ }}\right.\) nat (domain-block ls) \()\)
using rat-poly.row-length-def Cons under by auto
also have \(\ldots=2\) ^(2 + nat (domain-block ls))
using domain-block.simps by auto
also have \(\ldots=\) 2^ \(^{\wedge}(\) nat \((\) domain \(l)+\) nat (domain-block ls \()\) )
using domain.simps under by auto
also have \(\ldots=2^{\wedge}(\) nat \((\) domain \(l+\) domain-block ls \())\)
by (simp add: nat-add-distrib domain-block-nonnegative under)
also have \(\ldots=2^{\wedge}(\) nat \((\) domain-block \((l \# l s)))\)
using domain-block.simps by auto
finally show ?thesis.
next
case cup
have rat-poly.row-length (blockmat ls) = 2 \({ }^{\text {へ }}\) nat (domain-block ls) using Cons by auto
then have rat-poly.row-length (blockmat (l\#ls))
\[
\begin{aligned}
= & (\text { rat-poly.row-length (brickmat l) }) \\
& *(\text { rat-poly.row-length (blockmat ls }))
\end{aligned}
\]
using blockmat.simps rat-poly.row-length-mat by auto
moreover have \(\ldots=1 *\left(\mathcal{2}^{\text {~ nat (domain-block ls) })}\right.\)
using rat-poly.row-length-def Cons cup by auto
moreover have \(\ldots=2^{\wedge}(0+\) nat (domain-block ls \(\left.)\right)\) using domain-block.simps by auto
moreover have \(\ldots=\mathscr{Z}^{\wedge}(\) nat \((\) domain \(l)+\) nat \((\) domain-block ls \())\) using domain.simps cup by auto
moreover have \(\ldots=\mathcal{L}^{\wedge}(\) 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 \(\ldots=\mathcal{Z}^{\wedge}(\) nat (domain-block \((l \# l s))\) ) using domain-block.simps by auto
ultimately show ?thesis by metis
next
case cap
have rat-poly.row-length (blockmat ls) \(=2\) へ nat (domain-block ls) using Cons by auto
then have rat-poly.row-length (blockmat ( \(l \# l s\) ))
\[
\begin{aligned}
= & (\text { rat-poly.row-length }(\text { brickmat l) }) \\
& *(\text { rat-poly.row-length }(\text { blockmat ls }))
\end{aligned}
\]
using blockmat.simps rat-poly.row-length-mat by auto
moreover have \(\ldots=4 *\left(\mathcal{2}^{\wedge}\right.\) nat (domain-block ls)) using rat-poly.row-length-def Cons cap by auto
moreover have \(\ldots=2 \uparrow(2+\) nat (domain-block ls \()\) ) using domain-block.simps by auto
moreover have \(\ldots=\mathscr{2}^{\wedge}(\) nat \((\) domain \(l)+\) nat \((\) domain-block ls \())\) using domain.simps cap by auto
moreover have \(\ldots=\mathcal{Z}^{\wedge}(\) nat \((\) domain \(l+\) domain-block ls \())\)
by (simp add: cap domain-block-nonnegative nat-add-distrib)
moreover have \(\ldots=2^{\wedge}(\) nat (domain-block \((l \# l s))\) ) using domain-block.simps by auto
ultimately show ?thesis by metis qed
qed
lemma row-length-domain-block:
rat-poly.row-length (blockmat ls \()=\left(2^{\wedge}(\right.\) nat \(((\) domain-block ls \(\left.)))\right)\)
proof (induct ls)
case Nil
show ?case unfolding blockmat.simps(1) rat-poly.row-length-def by auto
next
case (Cons lls)
show? case
proof (cases l)
case vert
have rat-poly.row-length (blockmat ls) \(=2\) へ nat (domain-block ls)
using Cons by auto
then have rat-poly.row-length (blockmat \((l \# l s))\)
\[
\begin{aligned}
= & (\text { rat-poly.row-length }(\text { brickmat l })) \\
& *(\text { rat-poly.row-length }(\text { blockmat ls }))
\end{aligned}
\]
using blockmat.simps rat-poly.row-length-mat by auto
moreover have \(\ldots=2 *\left(2^{\text {^ nat (domain-block ls) }) ~}\right.\)
using rat-poly.row-length-def Cons vert by auto
moreover have \(\ldots=2 \uparrow(1+\) nat (domain-block ls \()\) )
using domain-block.simps by auto
moreover have \(\ldots=\mathscr{Z}^{\wedge}(\) nat \((\) domain l) + nat (domain-block ls \())\)
using domain.simps vert by auto
moreover have \(\ldots=2^{\wedge}(\) nat \((\) domain \(l+\) domain-block ls \())\)
using Suc-eq-plus1-left Suc-nat-eq-nat-zadd1 calculation(4) domain.simps(1)
domain-block-non-negative vert by metis
moreover have \(\ldots=\mathscr{2}^{\wedge}(\) nat \((\) domain-block \((l \# l s)))\)
using domain-block.simps by auto
ultimately show ?thesis by metis
next
case over
have rat-poly.row-length (blockmat ls) \(=2\) nat (domain-block ls)
using Cons by auto
then have rat-poly.row-length (blockmat (l\#ls))
\(=(\) rat-poly.row-length \((\) brickmat \(l))\)
*(rat-poly.row-length (blockmat ls))
using blockmat.simps rat-poly.row-length-mat by auto
moreover have \(\ldots=4 *\left(2^{\text {^ nat }}\right.\) (domain-block ls))
using rat-poly.row-length-def Cons over by auto
moreover have \(\ldots=2 \uparrow(2+\) nat \((\) domain-block ls \())\)
using domain-block.simps by auto
moreover have \(\ldots=\mathcal{V}^{\wedge}(\) nat \((\) domain l) + nat (domain-block ls \())\)
using domain.simps over by auto
```

moreover have $\ldots=$ 2 $^{\wedge}$ (nat (domain $l+$ domain-block ls))
by (simp add: over domain-block-nonnegative nat-add-distrib)
moreover have $\ldots=\mathcal{L}^{\wedge}($ nat (domain-block $\left.(l \# l s))\right)$
using domain-block.simps by auto
ultimately show ?thesis by metis
next
case under
have rat-poly.row-length (blockmat ls) $=2$ へ 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 $\ldots=4 *\left(2^{\wedge}\right.$ nat (domain-block ls))
using rat-poly.row-length-def Cons under by auto
moreover have $\ldots=2 \wedge(2+n a t($ domain-block ls $)$ )
using domain-block.simps by auto
moreover have $\ldots=\mathcal{Z}^{\wedge}($ nat $($ domain $l)+$ nat $($ domain-block ls $))$
using domain.simps under by auto
moreover have $\ldots=\mathcal{L}^{\wedge}($ nat $($ domain $l+$ domain-block ls $)$ )
by (simp add: under domain-block-nonnegative nat-add-distrib)
moreover have $\ldots=\mathcal{Z}^{\wedge}($ nat (domain-block $\left.(l \# l s))\right)$
using domain-block.simps by auto
ultimately show ?thesis by metis
next
case cup
have rat-poly.row-length (blockmat ls) = 2 ${ }^{\text {^ nat (domain-block ls) }}$
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 ... $=1 *\left(2^{\text {^ nat }}\right.$ (domain-block ls))
using rat-poly.row-length-def Cons cup by auto
moreover have $\ldots=\mathfrak{Z}^{〔}(0+n a t($ domain-block ls))
using domain-block.simps by auto
moreover have $\ldots=\mathcal{Z}^{\wedge}($ nat $($ domain $l)+$ nat $($ domain-block ls $))$
using domain.simps cup by auto
moreover have $\ldots=2^{\wedge}($ 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 $\ldots=2$ 2 (nat (domain-block $(l \# l s))$ )
using domain-block.simps by auto
ultimately show ?thesis by metis
next
case cap
have rat-poly.row-length (blockmat ls) $=2{ }^{\wedge}$ 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 length-codomain-block:length (blockmat ls)
=(2^
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））
\[
=(\text { length }(\text { brickmat } l)) *(\text { length }(\text { blockmat ls }))
\]
using blockmat．simps rat－poly．length－Tensor by auto
moreover have \(\ldots=4 *\left(2^{\text {へ nat（codomain－block ls）}) ~}\right.\)
using Cons over by auto
moreover have \(\ldots=2 \wedge(2+n a t(\) codomain－block ls \())\) by auto
moreover have \(\ldots=2^{\wedge}(\) nat \((\) codomain \(l)+\) nat \((\) codomain－block ls \())\) using codomain．simps over by auto
moreover have \(\ldots=2^{\wedge}(\) nat \((\) codomain \(l+\) codomain－block ls \())\)
using nat－sum over codomain．simps codomain－block－nonnegative by auto
moreover have \(\ldots=\mathcal{2}^{\wedge}(\) nat（codomain－block \(\left.(l \# l s))\right)\)
by auto
ultimately show ？thesis by metis
next
case under
have length（blockmat ls）\(=2^{\text {へ }}\) nat（codomain－block ls） using Cons by auto
then have length（blockmat（l\＃ls））
\[
=(\text { length }(\text { brickmat } l)) *(\text { length }(\text { blockmat ls }))
\]
using blockmat．simps rat－poly．length－Tensor by auto
moreover have \(\ldots=4 *\left(\mathcal{Z}^{\text {ヘ }}\right.\) nat（codomain－block ls）） using Cons under by auto
moreover have \(\ldots=2 \uparrow(2+\) nat \((\) codomain－block ls \())\) by auto
moreover have \(\ldots=2^{\wedge}(\) nat \((\operatorname{codomain} l)+\) nat \((\) codomain－block ls \())\) using codomain．simps under by auto
moreover have \(\ldots=2^{\wedge}(\) nat \((\) codomain \(l+\) codomain－block ls \())\) using nat－sum under codomain．simps codomain－block－nonnegative by auto
moreover have \(\ldots=\mathfrak{2}^{\wedge}(\) nat \((\) codomain－block \((l \# l s)))\) by auto
ultimately show ？thesis by metis
next
case cup
have length（blockmat ls） 2 \(^{\text {＾nat（codomain－block ls）}}\)
using Cons by auto
then have length（blockmat（l\＃ls））
\[
=(\text { length }(\text { brickmat } l)) *(\text { length }(\text { blockmat ls }))
\]
using blockmat．simps rat－poly．length－Tensor by auto
moreover have \(\ldots=4 *\left(2^{\wedge}\right.\) nat（codomain－block ls）） using Cons cup by auto
moreover have \(\ldots=2^{\wedge}(2+\) nat \((\) codomain－block ls \())\) by auto
moreover have \(\ldots=\mathcal{Z}^{〔}(\) nat \((\) codomain \(l)+\) nat（codomain－block ls \(\left.)\right)\) using codomain．simps cup by auto
moreover have \(\ldots=\mathscr{2}^{\wedge}(\) 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 $l s)$
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 $(a \# l s))=($ brickmat $a) \otimes($ 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 ( \(a \# l s\) ))
\[
\begin{aligned}
= & (\text { length }(\text { brickmat a })) \\
& *(\text { length }(\text { blockmat ls }))
\end{aligned}
\]
using blockmat.simps(2) rat-poly.length-Tensor by (metis)
ultimately have Cons-3:mat
\[
\begin{aligned}
& (\text { rat-poly.row-length (brickmat a) ) } \\
& (\text { length }(\text { brickmat a) }) \\
\Rightarrow & \text { ?case } \quad(\text { brickmat a) }
\end{aligned}
\]
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 \(1 \times 1\) matrix whose entry is the Kauffman bracket.
primrec kauff-mat::wall \(\Rightarrow\) rat-poly mat
where
kauff-mat \((\) basic \(w)=(\) blockmat \(w)\)
\(\mid\) kauff-mat \((w * w s)=\) 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)
\Longrightarrow(rat-poly.row-length (kauff-mat ws)) = 2`(nat (domain-wall ws))     \wedge(length (kauff-mat ws)) = 2`(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))
\(=\mathcal{2}^{\wedge}(\) nat \((\) domain-wall ws) \()\)
using prod.hyps prod-1 by auto
moreover have prod-4:length (kauff-mat ws)
\[
=2 \uparrow(\text { nat }(\text { codomain-wall ws }))
\]
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 \(w\) ) \(=2^{\wedge}\) (nat (domain-block w))
and length \((\) blockmat \(w)=\mathfrak{2}^{\wedge}(\) nat \((\) codomain-block \(w))\)
using row-length-domain-block length-codomain-block
by auto
ultimately have ad1:length (blockmat w) = rat-poly.row-length (kauff-mat ws)
using prod-2 by auto
then have mat
(rat-poly.row-length (blockmat w))
(length (kauff-mat ws))
(rat-poly.matrix-mult (blockmat w) (kauff-mat ws))
using prod-3 prod-5 mat-mult by auto
then have res1:mat
\(\quad\) (rat-poly.row-length (blockmat w))
\((\) length \((\) kauff-mat ws) \()\)
\((\) kauff-mat \((w * w s))\)
using kauff-mat.simps \((\mathcal{2})\) by auto
then have rat-poly.row-length (kauff-mat \((w * w s)\) )
\[
=(\text { rat-poly.row-length }(\text { blockmat } w))
\]
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 \(\ldots=\mathcal{Z}^{\wedge}(\) nat \((\) domain-wall \((w * w s)))\)
using prod-6 domain-wall.simps by auto
ultimately have res2:
\[
\text { rat-poly.row-length (kauff-mat }(w * w s))
\]
\[
=\mathcal{D}^{\wedge}(\text { nat }(\text { domain-wall }(w * w s)))
\]
by auto
have length (kauff-mat \((w * w s))=\) 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 \(\ldots=2^{\wedge}(\) nat \((\) codomain-wall \((w * w s)))\)
using prod-4 codomain-wall.simps(2) by auto
ultimately have res3：length（kauff－mat（w＊ws））
\(=\mathcal{Z}^{〔}(\) nat \((\) codomain－wall \((w * w s)))\)
by auto
with res1 res2 show ？case
using 〈length（kauff－mat ws）\(=2\) へ nat（codomain－wall \((w * w s)\) ）〉
〈rat－poly．row－length（blockmat \(w)=2{ }^{\wedge}\) nat（domain－wall \(\left.\left.(w * w s)\right)\right\rangle\)
by（metis）
qed
theorem effective－matrix－kauff－mat：
assumes is－tangle－diagram ws
shows \((\) rat－poly．row－length \((\) kauff－mat ws \())=\mathcal{Z}^{\wedge}(\) nat \((\) domain－wall ws \())\)
and length（kauff－mat ws）\(=\mathfrak{Z}^{\wedge}\)（nat（codomain－wall ws））
and mat（rat－poly．row－length（kauff－mat ws））（length（kauff－mat ws））
（kauff－mat ws）
apply（auto simp add：matrix－kauff－mat assms ）
using assms matrix－kauff－mat by metis
lemma mat－mult－equiv：
rat－poly．matrix－mult m1 m2＝mat－mult（rat－poly．row－length m1）m1 m2
by auto
theorem associative－rat－poly－mat：
assumes mat（rat－poly．row－length m1）（rat－poly．row－length m2）m1
and mat（rat－poly．row－length m2）（rat－poly．row－length m3）m2
and mat（rat－poly．row－length m3）nc m3
shows rat－poly．matrix－mult m1（rat－poly．matrix－mult m2 m3）
\(=\) rat－poly．matrix－mult（rat－poly．matrix－mult m1 m2）m3
proof－
have（rat－poly．matrix－mult m2 m3）
\(=\) mat－mult（rat－poly．row－length m2）m2 m3
using mat－mult－equiv by auto
then have rat－poly．matrix－mult m1（rat－poly．matrix－mult m2 m3）
＝mat－mult（rat－poly．row－length m1）m1
（ mat－mult（rat－poly．row－length m2）m2 m3）
using mat－mult－equiv by auto
moreover have ．．．＝mat－mult（rat－poly．row－length m1）
（mat－mult（rat－poly．row－length m1）m1 m2）m3
using assms mat－mult－assoc by metis
moreover have \(\ldots=\) rat－poly．matrix－mult（rat－poly．matrix－mult m1 m2）m3
proof－
have mat
（rat－poly．row－length \(m 1\) ）
（rat－poly．row－length m3）
（rat－poly．matrix－mult m1 m2）
using assms（1）assms（2）mat－mult by（metis）
then have rat－poly．row－length（rat－poly．matrix－mult m1 m2）\(=\)
（rat－poly．row－length m1）
using assms(1) assms(2) length-0-conv rat-poly.mat-empty-column-length rat-poly.matrix-row-length rat-poly.row-length-Nil rat-poly.unique-row-col(1) rat-poly.unique-row-col(2)
by (metis)
moreover have rat-poly.matrix-mult (rat-poly.matrix-mult m1 m2) m3
\(=\) mat-mult (rat-poly.row-length
(rat-poly.matrix-mult m1 m2))
(rat-poly.matrix-mult m1 m2) m3
using mat-mult-equiv by auto
then show ?thesis using mat-mult-equiv by (metis calculation)
qed
ultimately show ?thesis by auto
qed
It follows from this result that the Kauffman Matrix of a wall representing a link diagram, is a \(1 \times 1\) matrix. Thus it establishes a correspondence between links and rational functions.
```

theorem link-diagram-matrix:
assumes is-link-diagram ws
shows mat 11 (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) $\wedge$ (is-tangle-diagram ws2)
$\wedge($ domain-wall ws2 $=$ codomain-wall ws1 $)) \Longrightarrow$
kauff-mat $($ ws1 ○ ws2 $)=$ rat-poly.matrix-mult (kauff-mat ws1) (kauff-mat ws2)
proof (induct ws1)
case (basic w1)
have $($ basic w1 $) \circ(w s 2)=(w 1) *(w s 2)$
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.prems by (rule conjE)
then have 2:(is-tangle-diagram ws1)
$\wedge$ (codomain-block w1 = domain-wall ws1)
using is-tangle-diagram.simps(2) by metis
then have
mat ( $\mathfrak{Z}^{\wedge}($ nat $($ domain-wall ws1) $))\left(\mathfrak{Z}^{\wedge}(\right.$ nat $($ codomain-wall ws1) $))($ kauff-mat
ws1)
and mat $\left(\mathbb{R}^{\wedge}\left(\right.\right.$ nat $\left(\right.$ domain-block w1) )) ( $\mathscr{V}^{\wedge}($ nat $($ codomain-block w1) ))
(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))
(2^(nat (domain-wall ws1)))
(blockmat w1)
and mat
(2`(nat (domain-wall ws1)))
(2`(nat (domain-wall ws2)))
(kauff-mat ws1)
and (2^(nat (domain-wall ws1)))
\(=(\) rat-poly.row-length (kauff-mat ws 1\())\)
using effective-matrix-kauff-mat prod.prems 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))
(2ヘ(nat (domain-wall ws2)))
(kauff-mat ws1)
by auto
moreover have mat
(2`(nat (domain-wall ws2)))
(2`(nat (codomain-wall ws2)))
(kauff-mat ws2)
and (2`(nat (domain-wall ws2))) = rat-poly.row-length (kauff-mat ws2)
using prod.prems effective-matrix-kauff-mat
effective-matrix-kauff-mat
by (auto) (metis prod.prems)
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))
(2`(nat (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))
\[
=\text { rat-poly.matrix-mult }
\] (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.prems 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 \(\circ(\) 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 []))
\[
=(\text { kauff-mat ws })
\]
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 []) ows \())\)

\section*{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 []) ows))
= 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
```

            rat-poly.unique-row-col(1)
    ```
        by metis
    then show ?thesis using 1 by auto
qed
lemma left-id-blockmat:blockmat []\(\otimes\) blockmat \(b=\) blockmat \(b\)
unfolding blockmat.simps(1) rat-poly.Tensor-left-id by auto
lemma tens-assoc:
\(\forall a\) xs ys.(brickmat \(a \otimes\) (blockmat xs \(\otimes\) blockmat ys)
\[
=(\text { brickmat } a \otimes \text { blockmat xs }) \otimes \text { blockmat ys })
\]
proof-
have \(\forall a\). (mat
(rat-poly.row-length (brickmat a))
(length (brickmat a))
(brickmat a))
using brickmat.simps
unfolding mat-def rat-poly.row-length-def Ball-def vec-def apply (auto)
by (case-tac a) (auto)
moreover have \(\forall x s\). (mat
(rat-poly.row-length (blockmat xs))
(length (blockmat xs))
(blockmat xs))
using matrix-blockmat by auto
moreover have \(\forall y s\). mat
(rat-poly.row-length (blockmat ys))
(length (blockmat ys))
(blockmat ys)
using matrix-blockmat by auto
ultimately show ?thesis using rat-poly.associativity by auto qed
lemma kauff-mat-tensor-distrib:
\(\forall x s . \forall\) ys.(kauff-mat (basic xs \(\otimes\) basic ys)
\(=\) kauff-mat (basic xs) \(\otimes\) kauff-mat (basic ys))
apply(rule allI)
apply (rule allI)
apply (induct-tac xs)
apply (auto)
apply (metis rat-poly.vec-mat-Tensor-vector-id)
apply (simp add:tens-assoc)
done
lemma blockmat-tensor-distrib:
\((\) blockmat \((a \otimes b))=(\) blockmat \(a) \otimes(\) blockmat \(b)\)
proof -
have blockmat \((a \otimes b)=\) kauff-mat \((b a s i c(a \otimes b))\)
```

    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 \not= [])
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 \not= []
proof-
have (length (kauff-mat ws) = 2` (nat (codomain-wall ws)))
using effective-matrix-kauff-mat assms by auto
then have (length (kauff-mat ws)) \geq1
by auto
then show ?thesis by auto
qed

```
lemma is-tangle-diagram-length-rowlength:
    assumes is-tangle-diagram ( \(w * w s\) )
    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)
                        \(=\mathfrak{2}^{\wedge}(\) nat \((\) domain-wall ws \())\)
        using effective-matrix-kauff-mat by (metis assms is-tangle-diagram.simps(2))
    moreover have length (blockmat w)
                        \(=\) 2^ \(^{\wedge}\) (nat (codomain-block w))
        using matrix-blockmat length-codomain-block by auto
ultimately show ?thesis by auto
qed
```

lemma is-tangle-diagram-matrix-match:
assumes is-tangle-diagram (w1*ws1)
and is-tangle-diagram (w2*ws2)
shows rat-poly.matrix-match (blockmat w1)
(kauff-mat ws1) (blockmat w2) (kauff-mat ws2)
unfolding rat-poly.matrix-match-def
apply(auto)
proof-
show mat (rat-poly.row-length (blockmat w1)) (length (blockmat w1)) (blockmat
w1)
using matrix-blockmat by auto
next
have is-tangle-diagram ws1
using assms(1) is-tangle-diagram.simps(2) by metis
then show mat (rat-poly.row-length (kauff-mat ws1)) (length (kauff-mat ws1))
(kauff-mat ws1)
using matrix-kauff-mat by metis
next
show mat (rat-poly.row-length (blockmat w2)) (length (blockmat w2)) (blockmat
w2)
using matrix-blockmat by auto
next
have is-tangle-diagram ws2
using assms(2) is-tangle-diagram.simps(2) by metis
then show mat (rat-poly.row-length (kauff-mat ws2)) (length (kauff-mat ws2))
(kauff-mat ws2)
using matrix-kauff-mat by metis
next
show length (blockmat w1) = rat-poly.row-length (kauff-mat ws1)
using is-tangle-diagram-length-rowlength assms(1) by auto
next
show length (blockmat w2) = rat-poly.row-length (kauff-mat ws2)
using is-tangle-diagram-length-rowlength assms(2) by auto
next
assume 0:blockmat w1 = []
show False using 0
by (metis blockmat-non-empty)
next
assume 1:kauff-mat ws1=[]
have is-tangle-diagram ws1
using assms(1) is-tangle-diagram.simps(2) by metis
then show False using 1 kauff-mat-non-empty by auto
next
assume 0:blockmat w2 = []
show False using 0
by (metis blockmat-non-empty)
next
assume 1:kauff-mat ws2 = []

```
have is-tangle-diagram ws2
using assms(2) is-tangle-diagram.simps(2) by metis
then show False using 1 kauff-mat-non-empty by auto qed

The following function constructs a \(2^{n} \times 2^{n}\) identity matrix for a given \(n\)
primrec make-vert-equiv::nat \(\Rightarrow\) rat-poly mat
where
make-vert-equiv \(0=[[1]]\)
\(\mid\) 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)

\section*{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 22 (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 ( }\foralli<2.\forallj<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)}=(\mathrm{ make-vert-equiv n)
apply(induction n)
apply(simp)
unfolding make-vert-block.simps blockmat.simps make-vert-equiv.simps
using mat1-vert-equiv by auto
lemma prop-make-vert-equiv:
shows rat-poly.row-length (make-vert-equiv n)= 2`n         and length (make-vert-equiv n) =2`n
and mat
(rat-poly.row-length (make-vert-equiv n))
(length (make-vert-equiv n))
(make-vert-equiv n)
proof -
have 1:make-vert-equiv n = (blockmat (make-vert-block n))
using blockmat-make-vert by auto
moreover have 2:domain-block (make-vert-block n)= int n
using domain-make-vert by auto
moreover have 3:codomain-block (make-vert-block n)= int n
using codomain-make-vert by auto
ultimately show rat-poly.row-length (make-vert-equiv n)=2^n
and length (make-vert-equiv n) =2^n
and mat
(rat-poly.row-length (make-vert-equiv n))
(length (make-vert-equiv n))
(make-vert-equiv n)
apply (metis nat-int row-length-domain-block)
using 12 3 apply (metis length-codomain-block nat-int)
using 12 3 by (metis matrix-blockmat)
qed
abbreviation nat-mult::nat }=>\mathrm{ nat }=>\mathrm{ nat (infixl *n 65)
where
nat-mult a b \equiv((a::nat)*b)
lemma equal-div-mod:assumes ((j::nat) div a) = ( }i\mathrm{ div a
and (j mod a) = (i mod a)
shows j=i
proof-
have j=a*(j div a)}+(j\mathrm{ mod a)
by auto

```
```

then have j=a*(i div a) +(i mod a)
using assms by auto
then show ?thesis by auto
qed
lemma equal-div-mod2:(((j::nat) div a) = (i div a)
\wedge((j mod a)=(i\operatorname{mod}a)))=(j=i)
using equal-div-mod by metis
lemma impl-rule:
assumes (\foralli<m.\forallj<n.(Pi)\wedge(Qj))
and }\forallij.(Pi)\wedge(Qj)\longrightarrowRi
shows ( }\foralli<m.\forallj<n.Rij
using assms by metis
lemma implic:
assumes }\forallij.((Pij)\longrightarrow(Qij)
and }\forallij.((Qij)\longrightarrow(Rij)
shows }\forallij.((Pij)\longrightarrow(Rij)
using assms by auto
lemma assumes a< (b*c)
shows ((a::nat) div b)<c
using assms by (metis rat-poly.div-right-ineq)
lemma mult-if-then:((v = (if P then 1 else 0))
\wedge ( w = ( if Q then 1 else 0)) )
\Longrightarrow ( r a t - p o l y - t i m e s ~ v ~ w ~ = ~ ( i f ~ ( P \wedge Q ) ~ t h e n ~ 1 ~ e l s e ~ 0 ) ) ~
by auto
lemma rat-poly-unity:rat-poly-times $11=1$
by auto

```
```

lemma $((P \wedge Q) \longrightarrow R) \Longrightarrow(P \longrightarrow Q \longrightarrow R)$

```
lemma \(((P \wedge Q) \longrightarrow R) \Longrightarrow(P \longrightarrow Q \longrightarrow R)\)
    by auto
    by auto
lemma length (mat1 2) \(=2\)
lemma length (mat1 2) \(=2\)
    apply (simp add:mat1I-def)
    apply (simp add:mat1I-def)
    done
    done
theorem make-vert-equiv-mat:
theorem make-vert-equiv-mat:
make-vert-equiv \(n=(\) mat1 \((2\) 2 \(n))\)
make-vert-equiv \(n=(\) mat1 \((2\) 2 \(n))\)
proof(induction \(n\) )
proof(induction \(n\) )
case 0
case 0
    show ? case using 0 mat1-equiv by auto
    show ? case using 0 mat1-equiv by auto
next
next
case (Suc k)
case (Suc k)
    have 1:make-vert-equiv \(k=\) mat1 ( \(2^{\wedge} k\) )
    have 1:make-vert-equiv \(k=\) mat1 ( \(2^{\wedge} k\) )
        using Suc by auto
        using Suc by auto
    moreover then have make-vert-equiv \((k+1)=(\) mat1 2 \() \otimes\left(\operatorname{mat1}\left(\right.\right.\) 2^k \(\left.\left.^{2}\right)\right)\)
```

    moreover then have make-vert-equiv \((k+1)=(\) mat1 2 \() \otimes\left(\operatorname{mat1}\left(\right.\right.\) 2^k \(\left.\left.^{2}\right)\right)\)
    ```
using make-vert-equiv.simps(2) by auto
then have \(\left(\operatorname{mat1}\right.\) 2) \(\otimes\left(\operatorname{mat1}\left(\right.\right.\) 2^ \(\left.\left.^{\wedge}\right)\right)=\operatorname{mat1}\left(\right.\) 2 \(\left.^{\wedge}(k+1)\right)\) proof-
have 1:mat \(\left(2^{\wedge}(k+1)\right)\left(2^{\wedge}(k+1)\right)\left(\operatorname{mat1}\left(2^{\wedge}(k+1)\right)\right)\)
using mat1 by auto
have \(2:\left(\forall i<\mathcal{Z}^{\wedge}(k+1) . \forall j<\mathfrak{Z}^{\wedge}(k+1)\right.\).
\(\left(\right.\) mat1 \(\left(\mathcal{D}^{\wedge}(k+1)\right)!i!j=(\) if \(i=j\) then 1 else 0\(\left.\left.)\right)\right)\)
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 \(\left(\operatorname{mat1}\left(\mathcal{Z}^{\wedge} k\right)\right)=\mathcal{Z}^{\wedge} k\)
and \(7:\) length \(\left(\left(\operatorname{mat1}\left(2^{\wedge} k\right)\right)\right)=2^{\wedge} k\)
using Suc
by (metis prop-make-vert-equiv(1)) (simp add:mat1I-def)
then have \(8: m a t\)
\[
\begin{aligned}
& \text { (rat-poly.row-length } \left.\left(\text { mat1 }\left(\mathbb{2}^{\wedge}\right)\right)\right) \\
& \left(\text { length }\left(\text { mat1 }\left(\mathcal{2}^{\wedge} k\right)\right)\right) \\
& \quad\left(\text { mat1 }\left(\mathfrak{2}^{\wedge} k\right)\right)
\end{aligned}
\]
using Suc mat1 by (metis)

\section*{then have 9:}
\[
\begin{aligned}
& \left(\forall i<\left(\mathcal{Z}^{\Upsilon}(k+1)\right) . \forall j<\left(\mathcal{Z}^{\wedge}(k+1)\right)\right. \text {. } \\
& \text { ((rat-poly.Tensor (mat1 2) (mat1 (2^k))!j!i) } \\
& =\text { rat-poly-times } \\
& \text { ((mat1 2)! (j div (length (mat1 (2^k)))) } \\
& \text { !(idiv (rat-poly.row-length (mat1 (2~k))))) } \\
& \left(\left(\text { mat1 }\left(\mathfrak{Z}^{\wedge} k\right)\right)!(j \text { mod length }(\text { mat1 (2^k))) }\right. \\
& \left.\left.\left.!\left(i \bmod \left(\text { rat-poly.row-length }\left(\operatorname{mat1}\left(\mathfrak{2}^{\wedge} k\right)\right)\right)\right)\right)\right)\right)
\end{aligned}
\]
proof-
have \((\forall i<((\) rat-poly.row-length (mat1 2))
*n (rat-poly.row-length (mat1 (2^k)))).
\(\forall j<((\) length (mat1 2) ) \()\)
*n (length (mat1 \(\left.\left.\left(\mathcal{2}^{\wedge} k\right)\right)\right)\) ).
((rat-poly.Tensor (mat1 2) (mat1 (2^k))!j!i)
\(=\) rat-poly-times
((mat1 2)! (j div (length (mat1 (2^k))))
! (idiv (rat-poly.row-length (mat1 (2^k)))))
\(((\) mat1 \((2 \wedge k))!(j\) mod length \((\) mat1 (2^k) \())\)
\(!\left(i \bmod \left(\right.\right.\) rat-poly.row-length \(\left.\left.\left.\left.\left.\left(\operatorname{mat1}\left(\mathcal{Z}^{\wedge} k\right)\right)\right)\right)\right)\right)\right)\)
using 58 rat-poly.effective-matrix-Tensor-elements2 by (metis 3467 rat-poly.comm)
moreover have (rat-poly.row-length (mat1 2))
\[
\begin{gathered}
* n\left(\text { rat-poly.row-length }\left(\text { mat1 }\left(2^{\wedge} k\right)\right)\right) \\
=2^{\wedge}(k+1)
\end{gathered}
\]
using 36 by auto
moreover have (length (mat1 2))
\[
\text { *n(length } \left.\left(\text { mat1 }\left(\mathfrak{2}^{\wedge} k\right)\right)\right)
\]
\[
=2 \wedge(k+1)
\]
using 47 by (metis 36 calculation(2))
ultimately show ?thesis by metis
qed
have 10: \(\forall i j\).((idiv (rat-poly.row-length (mat1 (2^k))) < 2)
\(\wedge\left(j\right.\) div length \(\left(\right.\) mat1 \(\left(\right.\) 2^k \(\left.\left.^{\prime}\right)\right)<\) 2)
\(\longrightarrow(((\) mat1 2) \()(j\) div (length \((\) mat1 \((2 \wedge k))))\)
\(!\left(i \operatorname{div}\left(\right.\right.\) rat-poly.row-length \(\left(\right.\) mat1 \(\left.\left.\left.\left.\left(\mathcal{Z}^{\wedge} k\right)\right)\right)\right)\right)\)
\(=(\) if
((j div (length (mat1 (2^k)))) \(=(i \operatorname{div}(\) rat-poly.row-length \((\) mat1 \((2 \wedge k)))))\)
then 1
else 0)))
using mat1-index by (metis 6 7)
have \(11: \forall j .\left(j<\left(2^{\wedge}(k+1)\right) \longrightarrow j\right.\) div \((\) length \((\) mat1 \(\left.(2 \wedge k)))<2\right)\)
proof-
have \(\mathfrak{Z}^{\wedge}(k+1)=\left(2 * n\left(\right.\right.\) 2^ \(\left.\left.^{\wedge}\right)\right)\)
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:
\[
\forall i .(i<(\mathcal{2} \curlyvee(k+1))
\]
\(\longrightarrow(i \operatorname{div}(\) rat-poly.row-length \((\operatorname{mat1}(2 \wedge k))))<2)\)
proof-
have \(\mathscr{L}^{\wedge}(k+1)=\left(2 * n\left(\mathfrak{2}^{\wedge} k\right)\right)\)
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:
\[
\begin{aligned}
& \forall i j .\left(\left(i<\left(\mathcal{D}^{\wedge}(k+1)\right)\right) \wedge j<\left(\text { 2}^{\wedge}(k+1)\right) \longrightarrow\right. \\
& ((i \operatorname{div}(\text { rat-poly.row-length }(\text { mat1 }(2 \wedge k))))<2) \\
& \left.\wedge\left(\left(j \text { div }\left(\text { length }\left(\operatorname{mat1}\left(2^{\wedge} k\right)\right)\right)\right)<2\right)\right)
\end{aligned}
\]
by auto
have \(14: \forall i j .\left(i<\left(\right.\right.\) 2^ \(\left.\left.^{\wedge}(k+1)\right)\right) \wedge\left(j<\left(2^{\wedge}(k+1)\right)\right) \longrightarrow\) (((mat1 2)
! (j div (length (mat1 (2^k)))) !(idiv (rat-poly.row-length (mat1 (2^k))))) \(=(\) if
((j div (length (mat1 (2^k)))) \(=(i \operatorname{div}(\) rat-poly.row-length \((\operatorname{mat1}(2 \wedge k)))))\)
then 1
\(\operatorname{apply}(\) rule allI)
apply(rule alli)
proof
fix \(i j\)
assume \(0:(i:: n a t)<2^{\wedge}(k+1) \wedge(j::\) nat \()<2^{\wedge}(k+1)\)
have \(((i \operatorname{div}(\) rat-poly.row-length \((\) mat1 \((2 \wedge k))))<2)\)
\(\wedge((j\) div \((\) length \((\operatorname{mat1}(2 \sim k))))<2)\)
using 013 by auto
then show (((mat1 2)
\(!(j\) div (length \((\) mat1 \((2 \wedge k))))\)
!(idiv (rat-poly.row-length (mat1 (2^k)))))
\[
=(i f
\]
((jdiv (length (mat1 (2^k))))
\(=\left(i \operatorname{div}\left(\right.\right.\) rat-poly.row-length \(\left(\right.\) mat1 \(\left.\left.\left.\left.\left(2^{\wedge} k\right)\right)\right)\right)\right)\)
then 1
else 0))
using 10 by (metis 6 )
qed
have \(15: \forall i j .\left(\left(i \bmod (\right.\right.\) rat-poly.row-length \((\) mat1 \((2 \wedge k)))<\) 2^k \(\left.^{\prime}\right)\)
\(\wedge(j \bmod\) length \((\operatorname{mat1}(2 \wedge k))<2 \wedge k)\)
\(\longrightarrow\left(\left(\left(\right.\right.\right.\) mat1 \(\quad\) 2 \(\left.\left.^{\wedge} k\right)\right)\)
\(!\left(j \bmod \left(\right.\right.\) length \(\left.\left.\left(\operatorname{mat1}\left(2^{\wedge} k\right)\right)\right)\right)\)
\(!(i \bmod (\) rat-poly.row-length \((\operatorname{mat1}(2 ` k)))))\) \(=(\) if
\(\left(\left(j \bmod \left(\right.\right.\right.\) length \(\left.\left.\left(\operatorname{mat1}\left(2^{\wedge} k\right)\right)\right)\right)\)
\(=\left(i \bmod \left(\right.\right.\) rat-poly.row-length \(\left(\right.\) mat1 \(\left(\right.\) 2^k \(\left.\left.\left.\left.^{2}\right)\right)\right)\right)\)
then 1 else 0)))
using mat1-index by (metis 6 7)
have \(16: \forall j \cdot\left(j<\left(\right.\right.\) 2 \(\left.^{\wedge}(k+1)\right) \longrightarrow j \bmod (\) length \((\operatorname{mat1}(2 \wedge k)))<\) 2^k \(\left.^{2}\right)\)
proof-
have \(\mathfrak{Z}^{\wedge}(k+1)=\left(2 * n\left(\mathfrak{Z}^{\wedge} k\right)\right)\)
by auto
then show ?thesis
using 7 allI mod-less-divisor
nat-zero-less-power-iff zero-less-numeral by (metis)
qed
moreover have 17: \(\forall i .\left(i<\left(\mathcal{D}^{\wedge}(k+1)\right)\right.\)
\(\longrightarrow\left(i \bmod \left(\right.\right.\) rat-poly.row-length \(\left(\right.\) mat1 \(\left(\right.\) 2^ \(\left.\left.\left.\left.\left.^{\wedge}\right)\right)\right)\right)<\mathfrak{Z}^{\wedge} k\right)\)
proof-
have \(\mathcal{Z}^{\wedge}(k+1)=\left(2 * n\left(\mathcal{Z}^{\wedge} k\right)\right)\)
by auto
then show ?thesis using 7 allI by (metis 6 calculation)
qed
ultimately have 18:
\[
\begin{aligned}
& \forall i j .\left(\left(i<\left(\text { 2 }^{\wedge}(k+1)\right)\right) \wedge j<\left(\text { 2 }^{\wedge}(k+1)\right) \longrightarrow\right. \\
& \left(\left(i \bmod \left(\text { rat-poly.row-length }\left(\text { mat1 }\left(\text { 2^k }^{2}\right)\right)\right)\right)<\right.\text { 2^k) } \\
& \wedge((j \bmod (\text { length }(\operatorname{mat1}(2 ` k))))<2 \wedge k))
\end{aligned}
\]
```

    by (metis 7)
    have 19:\forallij.(i< (2`(k+1)))^(j< (2` (k+1)))}
(((mat1 (2`k))             !(j mod (length (mat1 (2`k))))
!(i mod (rat-poly.row-length (mat1 (2^k)))))
= (if
((j mod (length (mat1 (2^k))))
= (i mod (rat-poly.row-length (mat1 (2^k)))))
then 1
else 0))
apply(rule allI)
apply(rule allI)
proof
fix ij
assume 0:(i::nat)<2^(k+1)^(j::nat)<2^(k+1)
have ((i mod (rat-poly.row-length (mat1 (2^k)))) < 2^k)
^((j mod (length (mat1 (2`k))))<2^k)             using 0 18 by auto then show (((mat1 ( }\mp@subsup{\mathfrak{R}}{}{~}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 15 by (metis 6)
qed
have ( }\foralli.\forallj\mathrm{ .
(i<(\mp@subsup{\mathbb{R}}{}{`}(k+1)))})\wedge(j<(\mp@subsup{\mathbb{R}}{}{`}(k+1))
\longrightarrow ~ r a t - p o l y - t i m e s
((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))))                             \wedge((j mod (length (mat1 (2`k))))
= (i mod (rat-poly.row-length (mat1 (2`k)))))
then 1
else 0))
apply(rule allI)
apply(rule allI)
proof
fix ij

```
```

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\operatorname{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 ^     \longrightarrow ~ r a t - p o l y - t i m e s ~                 (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 -

```
\(k)\) )

\section*{have}
\(j\) div length (mat1 (2 ^\(k)\) )
\(=i\) div rat-poly.row-length \(\left(\right.\) mat1 \(\left.\left(2^{\wedge} k\right)\right)\)
\(\wedge j \bmod\) length \(\left(\right.\) mat1 \(\left.\left(2^{\wedge} k\right)\right)=i \bmod\) rat-poly.row-length \(\left(\operatorname{mat1}\left(\right.\right.\) 2 \(\left.\left.^{\wedge} k\right)\right)\)
\(\Longrightarrow\) rat-poly-times
(mat1 \(2!\left(j\right.\) div length \(\left(\right.\) mat1 \(\left.\left.\left(2^{\wedge} k\right)\right)\right)!(i\) div rat-poly.row-length (mat1
\(\left.\left.\left(2^{\wedge} k\right)\right)\right)\) )
\[
\begin{aligned}
& \left(\text { mat1 }\left(2^{\wedge} k\right)!\left(j \text { mod length }\left(\text { mat1 }\left(2^{\wedge} k\right)\right)\right)\right. \\
& \left.\quad!\left(i \text { mod rat-poly.row-length }\left(\text { mat1 }\left(2^{\wedge} \wedge\right)\right)\right)\right)=1
\end{aligned}
\]
proof-
assume local-assms:
\(j\) div length \(\left(\operatorname{mat1}\left(\mathcal{D}^{\wedge} k\right)\right)=i\) div rat-poly.row-length (mat1 \(\left.\left(\mathcal{2}^{\wedge} k\right)\right)\)
\(\wedge j \bmod\) length \(\left(\operatorname{mat1}\left(\mathcal{Z}^{\wedge} k\right)\right)=i \bmod\) rat-poly.row-length \(\left(\operatorname{mat1}\left(\mathcal{2}^{\wedge} k\right)\right)\)
have (mat1 2 ! ( \(j\) div length (mat1 (2 ^ \(k)\) )) ! (i div rat-poly.row-length \(\left.\left.\left(\operatorname{mat1}\left(2^{\wedge} k\right)\right)\right)\right)\)
\[
=1
\]
using s1 local-assms by metis
moreover have (mat1 ( \(\left.2^{\wedge} k\right)\)
\(!\left(j \bmod\right.\) length \(\left(\right.\) mat1 \(\left.\left.\left(\mathcal{Z}^{\wedge} k\right)\right)\right)!(i \bmod\) rat-poly.row-length (mat1 (2
\(\left.\left.\left.{ }^{\wedge} k\right)\right)\right)\) ) \(=1\)
using s2 local-assms by metis
ultimately show ?thesis
by (metis 367 Suc.IH local-assms mve1 prop-make-vert-equiv(1) prop-make-vert-equiv(2) rat-poly.right-id)
qed
then show ?thesis by auto
qed
show
( \(j\) div length \(\left(\right.\) mat1 \(\left(\right.\) 2 \(\left.\left.^{\wedge} k\right)\right)=i\) div rat-poly.row-length \(\left(\right.\) mat1 \(\left(\right.\) 2 ^ \(\left.\left.^{\wedge} k\right)\right) \longrightarrow\) \(j\) mod length (mat1 (2 ^ \(k\) )) \(\neq i \bmod\) rat-poly.row-length (mat1 (2 ^
\(k))) \longrightarrow\)
mat1 \(2!\left(j\right.\) div length \(\left(\right.\) mat1 \(\left.\left.\left(2^{\wedge} k\right)\right)\right)!(i\) div rat-poly.row-length (mat1 (2
\(\left.\left.{ }^{\wedge} k\right)\right)=0 \vee\)
mat1 \(\left(\mathcal{2}^{\wedge} k\right)!\left(j\right.\) mod length \(\left(\right.\) mat1 \(\left.\left.\left(2^{\wedge} k\right)\right)\right)!(i \bmod\) rat-poly.row-length \(\left(\operatorname{mat1}\left(\right.\right.\) 2 ^ \(\left.\left.\left.^{\wedge}\right)\right)\right)=0\)
proof-
have \(\left(j\right.\) div length \(\left(\right.\) mat1 \(\left.\left(2^{\wedge} k\right)\right)=i\) div rat-poly.row-length \(\left(\right.\) mat1 \(\left.\left(2^{\wedge} k\right)\right)\)
\(\wedge j\) mod length \(\left(\right.\) mat1 \(\left.\left(\mathcal{Z}^{\wedge} k\right)\right) \neq i \bmod\) rat-poly.row-length (mat1 (2 \({ }^{\wedge}\)
\(k))) \Longrightarrow\)
mat1 2 ! ( \(j\) div length (mat1 (2 ^ \(k)\) )) ! (i div rat-poly.row-length (mat1 \(\left.\left.\left(2^{\wedge} k\right)\right)\right)=0\)
\(\vee \operatorname{mat1}\left(\right.\) 2 \(\left.^{\wedge} k\right)!\left(j\right.\) mod length \(\left(\right.\) mat1 \(\left(\right.\) 2 ^ \(\left.\left.\left.^{\wedge} k\right)\right)\right)!(i \bmod\) rat-poly.row-length \(\left(\operatorname{mat1}\left(\right.\right.\) 2 \(\left.\left.\left.^{\wedge} k\right)\right)\right)=0\)
proof-
assume local-assms:
( \(j\) div length \(\left(\operatorname{mat1}\left(\right.\right.\) 2 \(\left.\left.^{\wedge} k\right)\right)=i\) div rat-poly.row-length \(\left(\right.\) mat1 \(\left.\left(2^{\wedge} k\right)\right)\)
\(\wedge j\) mod length (mat1 \(\left.\left(\mathcal{2}^{\wedge} k\right)\right) \neq i\) mod rat-poly.row-length (mat1 (2
~ \(k\) ) )
have mat1 \(\left(\right.\) 2 \(\left.^{\wedge} k\right)!\left(j\right.\) mod length \(\left(\right.\) mat1 \(\left(\right.\) 2 \(\left.\left.\left.^{\wedge} k\right)\right)\right)!(i\) mod rat-poly.row-length \(\left.\left(\operatorname{mat1}\left(\mathcal{2}^{\wedge} k\right)\right)\right)=0\) using s2 local-assms by metis
then show ?thesis by auto
qed
then have \(l\) :
\(\left(j\right.\) div length \(\left(\right.\) mat1 \(\left.\left(\mathcal{Z}^{\wedge} k\right)\right)=i\) div rat-poly.row-length \(\left(\right.\) mat1 \(\left.\left(\mathcal{2 ~}^{\wedge} k\right)\right)\)
\(\wedge j \bmod\) length \(\left(\operatorname{mat1}\left(2^{\wedge} k\right)\right) \neq i \bmod\) rat-poly.row-length (mat1 (2 ^ \(\left.\left.k\right)\right)\) ) \(\longrightarrow\)
mat1 2 ! ( \(j\) div length (mat1 (2 \(\left.\left.{ }^{\wedge} k\right)\right)\) ! (i div rat-poly.row-length (mat1 (2 - \(k)\) ) \(=0\)
\(\vee\) mat1 (2 ~ \(k)!\left(j\right.\) mod length \(\left(\right.\) mat1 \(\left(\right.\) 2 \(\left.\left.\left.^{\wedge} k\right)\right)\right)!(i \bmod\) rat-poly.row-length \(\left.\left(\operatorname{mat1}\left(2^{\wedge} k\right)\right)\right)=0\)
by auto
show \(\left(j\right.\) div length \(\left(\right.\) mat1 \(\left.\left(2^{\wedge} k\right)\right)=i\) div rat-poly.row-length (mat1 (2 ^k)) \(\longrightarrow\)
\(j \bmod\) length \(\left(\operatorname{mat1}\left(2^{\wedge} k\right)\right) \neq i \bmod\) rat-poly.row-length \(\left.\left(\operatorname{mat1}\left(\mathcal{Z}^{\wedge} k\right)\right)\right) \longrightarrow\) mat1 2 ! ( \(j\) div length \(\left(\right.\) mat1 \(\left.\left(2^{\wedge} k\right)\right)\) ! (i div rat-poly.row-length (mat1 (2 \({ }^{\wedge}\) \(k)))=0 \vee\)
mat1 \(\left(\right.\) 2 \(\left.^{\wedge} k\right)!\left(j\right.\) mod length \(\left(\right.\) mat1 \(\left.\left.\left(2^{\wedge} k\right)\right)\right)!(i \bmod\) rat-poly.row-length (mat1 \(\left.\left.\left(2^{\wedge} k\right)\right)\right)=0\)

\section*{proof-}
have
( \(j\) div length \(\left(\right.\) mat1 \(\left.\left(2^{\wedge} k\right)\right)=i\) div rat-poly.row-length \(\left(\operatorname{mat1}\left(2^{\wedge} k\right)\right) \longrightarrow\)
\(j\) mod length \(\left(\right.\) mat1 \(\left.\left(2^{\wedge} k\right)\right) \neq i \bmod\) rat-poly.row-length (mat1 (2 ^ \(\left.\left.k\right)\right)\) )
mat1 2 ! (j div length \(\left(\right.\) mat1 \(\left.\left.\left(2^{\wedge} k\right)\right)\right)!\left(i\right.\) div rat-poly.row-length (mat1 (2 \({ }^{\wedge}\) \(k)))=0 \vee\)
mat1 (2 \(\left.{ }^{\wedge} k\right)!\left(j\right.\) mod length \(\left(\right.\) mat1 (2 \(\left.\left.\left.{ }^{\wedge} k\right)\right)\right)!(i \bmod\) rat-poly.row-length \(\left(\right.\) mat1 \(\left.\left.\left(2^{\wedge} k\right)\right)\right)=0\)
proof-
assume local-assm1:
\(\left(j\right.\) div length \(\left(\right.\) mat1 \(\left.\left(2^{\wedge} k\right)\right)=i\) div rat-poly.row-length \(\left(\right.\) mat1 \(\left.\left(\mathcal{Z ~}^{\wedge} k\right)\right) \longrightarrow\) \(j\) mod length (mat1 \(\left.\left(\mathcal{Z}^{\wedge} k\right)\right) \neq i\) mod rat-poly.row-length (mat1 (2 \(\left.{ }^{\wedge} k\right)\) )
have \(\left(j\right.\) div length \(\left(\right.\) mat1 \(\left.\left(\mathcal{2}^{\wedge} k\right)\right)=i\) div rat-poly.row-length (mat1 (2 \({ }^{\wedge}\) \(k)\) )
```

mat1 $\left(\mathcal{R}^{\wedge} k\right)!\left(j\right.$ mod length $\left(\right.$ mat1 $\left.\left.\left(\mathcal{Z ~}^{\wedge} k\right)\right)\right)$
! (i mod rat-poly.row-length (mat1 (2 $\left.\left.{ }^{\wedge} k\right)\right)$ )
$=0$
using s2 local-assm1 by (metis 7)

```
then have l1: ( \(j\) div length (mat1 (2 ^ \(k\) ) ) \(=i\) div rat-poly.row-length (mat1 \(\left.\left(2^{\wedge} k\right)\right)\) )
\[
\Longrightarrow \text { ?thesis }
\]
by auto
moreover have \(\neg\left(j\right.\) div length \(\left(\right.\) mat1 \(\left.\left(2^{\wedge} k\right)\right)=i\) div rat-poly.row-length \(\left(\operatorname{mat1}\left(\mathcal{2}^{\text {~ } k)}\right)\right)\)
\(\Longrightarrow\) mat1 2! (j div length (mat1 (2^k)))
! ( \(i\) div rat-poly.row-length (mat1 (2^k))
\[
=0
\]
using \(s 1\) by metis
then have \(\neg\left(j\right.\) div length (mat1 \(\left.\left(2^{\wedge} k\right)\right)=i\) div rat-poly.row-length (mat1 \(\left.\left(2^{\wedge} k\right)\right)\) )
\[
\Longrightarrow \text { ?thesis }
\]
by auto
then show ?thesis using \(l 1\) by auto
qed
then show ?thesis by auto
qed
qed
qed
qed
then have \(\left(\forall i . \forall j .\left(i<\left(\mathcal{D}^{\wedge}(k+1)\right)\right) \wedge\left(j<\left(\mathcal{D}^{\wedge}(k+1)\right)\right) \longrightarrow\right.\)
((rat-poly.Tensor (mat1 2) (mat1 (2^k))! \(j!i)=(\) if
\(\left(\left(\left(j\right.\right.\right.\) div \(\left(\right.\) length \(\left(\right.\) mat1 \(\left.\left.\left.\left(\mathcal{L}^{\wedge} k\right)\right)\right)\right)=(i\) div \((\) rat-poly.row-length \((\) mat1
\(\wedge\left(\left(j \bmod \left(\right.\right.\right.\) length \(\left.\left.\left(\operatorname{mat1}\left(2^{\wedge} k\right)\right)\right)\right)=(i \bmod (\) rat-poly.row-length
\(\left(\operatorname{mat1}\left(\right.\right.\) 2^k \(\left.\left.\left.\left.\left.^{2}\right)\right)\right)\right)\right)\)

> then 1 else 0\()\) )
using 9 by metis
then have \(\left(\forall i . \forall j .\left(i<\left(\mathfrak{D}^{\wedge}(k+1)\right)\right) \wedge\left(j<\left(\mathbb{D}^{\wedge}(k+1)\right)\right) \longrightarrow\right.\)
((rat-poly.Tensor (mat1 2) (mat1 (2^k))!j!i) \(=(\) if
\(\left(((j \operatorname{div}(2 \wedge k)))=\left(i \operatorname{div}\left(\right.\right.\right.\) 2^k \(\left.\left.^{\prime}\right)\right)\)
\(\left.\wedge\left(\left(j \bmod \left(\mathfrak{Z}^{\wedge} k\right)\right)=\left(i \bmod \left(\mathfrak{Z}^{\wedge} k\right)\right)\right)\right)\)
then 1 else 0))
by (metis (opaque-lifting, no-types) 67 )
then have 20: \(\left(\forall i . \forall j .\left(i<\left(2^{\wedge}(k+1)\right)\right) \wedge\left(j<\left(2^{\wedge}(k+1)\right)\right) \longrightarrow\right.\)
((rat-poly.Tensor (mat1 2) (mat1 (2^k))! j! \()=(i f(j=i)\)
then 1 else 0)))
using equal-div-mod2 by auto
with 2 have \(\left(\forall i . \forall j .\left(i<\left(\right.\right.\right.\) 2^ \(\left.\left.^{\wedge}(k+1)\right)\right) \wedge\left(j<\left(\mathcal{R}^{\wedge}(k+1)\right)\right) \longrightarrow\) \(((\) rat-poly.Tensor (mat1 2) (mat1 (2^k))! \(!i)=\) mat1 \(\left.\left.\left.\left(2^{\wedge}(k+1)\right)\right)!j!i\right)\right)\)
by metis
then have \(\left(\forall i<\left(\mathcal{L}^{\wedge}(k+1)\right) . \forall j<\left(\mathbb{R}^{\wedge}(k+1)\right)\right.\).
\(\left((\right.\) rat-poly.Tensor (mat1 2) \((\) mat1 \((2 \wedge k))!j!i)=\left(\right.\) mat1 \(\left(\right.\) 2^ \(\left.\left.\left.\left.^{\wedge}(k+1)\right)\right)!j!i\right)\right)\)
by auto
moreover have mat \(\left(\mathfrak{Z}^{\wedge}(k+1)\right)\left(\mathfrak{Z}^{\wedge}(k+1)\right)\) (rat-poly.Tensor (mat1 2) (mat1 (2^k))
using \(\left\langle\right.\) make-vert-equiv \((k+1)=\) mat1 \(\left.2 \otimes \operatorname{mat1}\left(2^{\wedge} k\right)\right\rangle\)
by (metis prop-make-vert-equiv(1) prop-make-vert-equiv(2)
prop-make-vert-equiv(3))
ultimately have (rat-poly.Tensor (mat1 2) (mat1 \(\left(\right.\) 2 \(\left.\left.\left.^{\wedge} k\right)\right)\right)=\left(\operatorname{mat1}\left(\mathcal{2}^{\wedge}(k+1)\right)\right)\)
using 1 mat-eqI by metis
then show ?thesis by auto
qed
then show ?case using make-vert-equiv.simps
using 〈make-vert-equiv \((k+1)=\) mat1 \(\left.2 \otimes \operatorname{mat1}\left(2^{\wedge} k\right)\right\rangle\)
by (metis Suc-eq-plus1)

\section*{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 \((n c))=m 1\)
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 \(\left(\mathcal{D}^{\wedge}(\right.\) 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)
\(\wedge(A 1 \neq[]) \wedge(A 2 \neq[]) \wedge(B 1 \neq[])\)
\(\wedge\) (length \(A 1=\) rat-poly.row-length A2)
theorem weak-distributivity1:
weak-matrix-match A1 A2 B1
\(\Longrightarrow((\) rat-poly.matrix-mult A1 A2) \(\otimes\) B1 \()\)
\(=(\) rat-poly.matrix-mult \((A 1 \otimes B 1)(A 2))\)
proof-
assume assms:weak-matrix-match A1 A2 B1
have 1 :length \(B 1=1\)
using assms weak-matrix-match-def
by (metis rat-poly.matrix-row-length rat-poly.unique-row-col(2))
have [[1]] \(\neq[]\)
by auto
moreover have mat 1 1 [[1]]
unfolding mat-def Ball-def vec-def by auto
moreover have rat-poly.row-length [[1]] = length B1
unfolding rat-poly.row-length-def 1 by auto
ultimately have rat-poly.matrix-match A1 A2 B1 [[1]]
unfolding rat-poly.matrix-match-def
using assms weak-matrix-match-def 1 blockmat.simps(1) matrix-blockmat by (metis (opaque-lifting, no-types))
then have ((rat-poly.matrix-mult A1 A2) \(\otimes\) (rat-poly.matrix-mult B1 [[1]])) \(=(\) rat-poly.matrix-mult \((A 1 \otimes B 1)(A 2 \otimes[[1]]))\)
using rat-poly.distributivity by auto
moreover have (rat-poly.matrix-mult B1 [[1]]) = B1
using weak-matrix-match-def assms mat1-equiv mat1-mult-right by (metis)
moreover have \((A 2 \otimes[[1]])=A 2\)
using rat-poly.Tensor-right-id by (metis)
ultimately show ?thesis by auto
qed
definition weak-matrix-match2::
rat-poly mat \(\Rightarrow\) rat-poly mat \(\Rightarrow\) rat-poly mat \(\Rightarrow\) bool
where
weak-matrix-match2 A1 B1 B2 \(\equiv(\) mat (rat-poly.row-length A1) 1 A1)
\(\wedge(\) mat (rat-poly.row-length B1) (length B1) B1)
\(\wedge(\) mat (rat-poly.row-length B2) (length B2) B2)
\(\wedge(A 1 \neq[]) \wedge(B 1 \neq[]) \wedge(B 2 \neq[])\)
\(\wedge\) (length B1 = rat-poly.row-length B2)
theorem weak-distributivity2:
weak-matrix-match2 A1 B1 B2
\[
\Longrightarrow(A 1 \otimes(\text { rat-poly.matrix-mult B1 B2 }))
\]
\[
=(\text { rat-poly.matrix-mult }(A 1 \otimes B 1)(\text { B2 }))
\]
proof -
assume assms:weak-matrix-match2 A1 B1 B2
have 1 :length \(A 1=1\)
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]] = 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 ((rat-poly.matrix-mult A1 [[1]])\otimes(rat-poly.matrix-mult B1 B2))
=(rat-poly.matrix-mult (A1 \otimesB1) ([[1]]\otimesB2))
using rat-poly.distributivity by auto
moreover have (rat-poly.matrix-mult A1 [[1]]) = A1
using weak-matrix-match2-def
assms mat1-equiv mat1-mult-right
by (metis)
moreover have ([[1]]\otimesB2)=B2
by (metis rat-poly.Tensor-left-id)
ultimately show ?thesis by auto
qed
lemma is-tangle-diagram-weak-matrix-match:
assumes is-tangle-diagram (w1*ws1)
and codomain-block w2 =0
shows weak-matrix-match (blockmat w1) (kauff-mat ws1) (blockmat w2)
unfolding weak-matrix-match-def
apply(auto)
proof-
show mat
(rat-poly.row-length (blockmat w1))
(length (blockmat w1))
(blockmat w1)
using matrix-blockmat by auto
next
have is-tangle-diagram ws1
using assms(1) is-tangle-diagram.simps(2) by metis
then show mat
(rat-poly.row-length (kauff-mat ws1))
(length (kauff-mat ws1))
(kauff-mat ws1)
using matrix-kauff-mat by metis
next
have mat
(rat-poly.row-length (blockmat w2))
(length (blockmat w2))
(blockmat w2)
using matrix-blockmat by auto

```
then have 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)
\Longrightarrow ( k a u f f - m a t ~ ( w s 1 ~ \otimes ~ w s \mathcal { R } ) = ( kauff-mat ws1) \otimes ( k a u f f - m a t ~ w s 2 ) ) ~ )
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
\Longrightarrow(kauff-mat (as \otimesbs)=kauff-mat as \otimes kauff-mat bs)
assume prems: is-tangle-diagram (a*as) ^ is-tangle-diagram (b*bs)
let ?case = kauff-mat (a*as\otimesb*bs)

```
\[
=k a u f f-m a t(a * a s) \otimes \text { kauff-mat }(b * b s)
\]
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 \(\wedge\) is-tangle-diagram bs
using prems is-tangle-diagram.simps by metis
have kauff-mat \(((a * a s) \otimes(b * b s))\)
\[
=\text { kauff-mat }((a \otimes b) *(a s \otimes b s))
\]
using tensor.simps by auto
moreover have \(\ldots=\) rat-poly.matrix-mult
(blockmat \((a \otimes b)\) )
(kauff-mat \((a s \otimes b s))\)
using kauff-mat.simps(2) by auto
moreover have \(\ldots=\) rat-poly.matrix-mult
\(((\) blockmat a \() \otimes(\) blockmat b) \()\)
\(((\) kauff-mat as \() \otimes(\) kauff-mat bs \())\)
using hyps 1 kauff-mat-tensor-distrib by auto
moreover have \(\ldots=(\) rat-poly.matrix-mult (blockmat a) (kauff-mat as))
\(\otimes\) (rat-poly.matrix-mult (blockmat b) (kauff-mat bs))
using 0 rat-poly.distributivity by auto
moreover have \(\ldots=\) kauff-mat \((a * a s) \otimes\) kauff-mat \((b * b s)\)
by auto
ultimately show ?case by metis
next
fix \(a b\) as \(b s\)
assume hyps:codomain-block \(b \neq 0\)
\(\Longrightarrow\) is-tangle-diagram as
\(\wedge\) is-tangle-diagram
(basic (make-vert-block (nat (codomain-block b))))
\(\Longrightarrow\) kauff-mat
(as \(\otimes\) basic (make-vert-block (nat (codomain-block b))))
\(=\) kauff-mat as
Q kauff-mat (basic (make-vert-block (nat (codomain-block b))))
assume prems:is-tangle-diagram \((a * a s) \wedge i s\)-tangle-diagram (basic b)
let ?case \(=\) kauff-mat \((a * a s \otimes\) basic b) \(=\) kauff-mat \((a * a s) \otimes\) kauff-mat (basic b)
show ? case
proof (cases codomain-block \(b=0\) )
case True
have \(((a * a s) \otimes(b a s i c b))=((a \otimes b) * a s)\)
using tensor.simps True by auto
then have kauff-mat \(((a * a s) \otimes(\) basic \(b))\)
\(=\) kauff-mat \(((a \otimes b) * a s)\)
by auto
moreover have ... =

> rat-poly.matrix-mult
(blockmat \((a \otimes b))\)
(kauff-mat as)
by auto
moreover have ... =
rat-poly.matrix-mult
\(((\) blockmat \(a) \otimes(\) blockmat b) \()\)
(kauff-mat as)
using blockmat-tensor-distrib by (metis)
ultimately have \(T 1\) :
\[
\begin{aligned}
& \text { kauff-mat }((a * \text { as }) \otimes(\text { basic b })) \\
& =\text { rat-poly.matrix-mult } \\
& \quad((\text { blockmat a) } \otimes(\text { blockmat b) }) \\
& \quad(\text { kauff-mat as })
\end{aligned}
\]
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 \() \otimes(\) blockmat \()))\)
\((\) kauff-mat as \()\)
\(\quad=((\) rat-poly.matrix-mult
\(\quad(\) blockmat a \()\)
\(\quad(\) kauff-mat as \())\)
\(\otimes(\) blockmat b) \()\)
using weak-distributivity1 by auto
moreover have \(\ldots=(\) kauff-mat \((a * a s)) \otimes(\) kauff-mat \((\) basic b) \()\)
by auto
ultimately show ?thesis using \(T 1\) by metis
next
case False
let ?bs \(=(\) basic \((\) make-vert-block \((\) nat \((\) codomain-block b) \()))\)
have 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 \(b \neq 0\)
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
using F1 hyps by auto
moreover have \(((a * a s) \otimes(\) basic \(b))=(a \otimes b) *(a s \otimes ? b s)\)
using False tensor.simps by auto
moreover then have kauff-mat \(((a * a s) \otimes(\) basic \(b))\)
\[
=\text { kauff-mat }((a \otimes b) *(a s \otimes ? b s))
\]
by auto
moreover then have \(\ldots=\) rat-poly.matrix-mult
(blockmat \((a \otimes b))\)
(kauff-mat \((a s \otimes ? b s))\)
using kauff-mat.simps by auto moreover then have..\(=\) rat-poly.matrix-mult \(((\) blockmat \(a) \otimes(\) blockmat b) \()\) \(((\) kauff-mat as \() \otimes(\) kauff-mat ?bs \())\)
using F3 blockmat-tensor-distrib by (metis)
moreover then have
\[
\begin{aligned}
= & (\text { rat-poly.matrix-mult (blockmat a) (kauff-mat as) }) \\
& \otimes(\text { rat-poly.matrix-mult (blockmat b) (kauff-mat ?bs) })
\end{aligned}
\]
using rat-poly.distributivity F0 by auto moreover then have ...
\[
\begin{aligned}
&=(\text { rat-poly.matrix-mult } \\
&(\text { blockmat a) } \\
&(\text { kauff-mat as })) \\
& \otimes(\text { blockmat b) }
\end{aligned}
\]
using mat1-vert-block by auto
moreover then have \(\ldots=(\) kauff-mat \((a * a s))\)
\(\otimes(\) kauff-mat (basic b))
using kauff-mat.simps by auto
ultimately show ?thesis by metis
qed
next
fix \(a b\) as \(b s\)
assume hyps:
codomain-block \(b \neq 0\)
\(\Longrightarrow\) is-tangle-diagram
(basic (make-vert-block (nat (codomain-block b)))) \(\wedge\) (is-tangle-diagram as)
\(\Longrightarrow\) kauff-mat (basic (make-vert-block (nat (codomain-block b))) \(\otimes a s)\)
\(=\) kauff-mat (basic (make-vert-block (nat (codomain-block b))))
Q kauff-mat as
assume prems:is-tangle-diagram (basic b) \(\wedge\) is-tangle-diagram \((a * a s)\)
let ?case \(=\) kauff-mat \(((\) basic \(b) \otimes(a * a s))\)
\(=k a u f f-m a t(\) basic \(b) \otimes\) kauff-mat \((a * a s)\)
show ?case
proof (cases codomain-block \(b=0\) )
case True
have \(((\) basic \(b) \otimes(a * a s))=((b \otimes a) * a s)\)
using tensor.simps True by auto
then have kauff-mat \(((\) basic \(b) \otimes(a * a s))\)
\[
=\text { kauff-mat }((b \otimes a) * a s)
\]
by auto
moreover have \(\ldots=\) rat-poly.matrix-mult (blockmat \((b \otimes a))\)
(kauff-mat as)
by auto
moreover have \(\ldots=\) rat-poly.matrix-mult \(((\) blockmat \(b) \otimes(\) blockmat \(a))\)
(kauff-mat as)
using blockmat-tensor-distrib by (metis)
ultimately have T1:kauff-mat \(((\) basic \(b) \otimes(a * a s))\)
\[
\begin{aligned}
= & \text { rat-poly.matrix-mult } \\
& ((\text { blockmat b) } \otimes(\text { blockmat a })) \\
& (\text { kauff-mat as })
\end{aligned}
\]
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 \(\ldots=(\) kauff-mat \((\) basic \(b)) \otimes(\) kauff-mat \((a * a s))\)
by auto
ultimately show ?thesis using \(T 1\) 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 \((? b s \otimes a s)=\) kauff-mat \(? b s \otimes\) kauff-mat as using F1 hyps by auto
```

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

```
end

\section*{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))

\section*{(rat-poly-plus}
(rat-poly-times 2 (rat-poly-times \(A(\) rat-poly-times \(A(\) rat-poly-times \(B B)))\)
\((\) rat-poly-times \((\) rat-poly-times \(B B)(\) rat-poly-times \(B B)))=\) \(\left(\left(A_{4}\right)+\left(B_{4}\right)+2\right)\)
proof-
have (rat-poly-times (rat-poly-times \(A A)(\) rat-poly-times \(A A))=A \wedge 4\)
by (simp add: numeral-Bit0)
moreover have (rat-poly-times (rat-poly-times \(B B)(\) rat-poly-times \(B B))\) \(=B^{\wedge} 4\)
by (simp add: numeral-Bit0)
moreover have (rat-poly-times 2 (rat-poly-times \(A\) (rat-poly-times \(A\) (rat-poly-times \(B B())\) )
\[
=2
\]
using inverse1 by (metis mult-2-right one-add-one rat-poly.assoc rat-poly.comm) ultimately show ?thesis by auto
qed
lemma computation-swingpos:
rat-poly-plus (rat-poly-times \(B\) (rat-poly-times \((A-r a t-p o l y-t i m e s\) (rat-poly-times B B) B) B) )

(rat-poly-times \(A(A-\) rat-poly-times \((\) rat-poly-times \(B B) B)))=\)
rat-poly-times \(A\) (rat-poly-times \((A\) rat-poly-times (rat-poly-times B B) B) A)
(is ?l \(=? r\) )
proof-
have 1:(A-rat-poly-times (rat-poly-times B B) B)
\(=A-\left(B^{\wedge} 3\right)\)
by (metis power3-eq-cube)
then have 2:(rat-poly-times \((A-\) rat-poly-times (rat-poly-times \(B B) B) B)\) \(=A * B-\left(B^{\wedge} 3\right) * B\)
by (metis minus-right-distributivity)
then have \(\ldots=1-\left(B^{\wedge} 4\right)\)
by (simp add: inverse1 numeral-Bit0 power3-eq-cube)
then have (rat-poly-times \(B\) (rat-poly-times \((A-\) rat-poly-times (rat-poly-times \(B\) B) B) B) )
\[
=B-\left(B^{\wedge} 4\right) * B
\]
using \({ }^{2}\)
by (metis minus-right-distributivity mult.commute mult.right-neutral)
then have \(3:\) (rat-poly-times \(B\) (rat-poly-times ( \(A\) - rat-poly-times (rat-poly-times \(B B) B) B)\)
\[
=B-\left(B^{\wedge} 5\right)
\]
by (metis (no-types, lifting) inverse1 minus-right-distributivity
mult.left-commute mult.right-neutral power2-eq-square power-numeral-odd)
have (rat-poly-times ( \(A\) - rat-poly-times (rat-poly-times \(B B\) ) B)
(rat-poly-times \(A(A-\) rat-poly-times \((\) rat-poly-times \(B B) B))\)
\(=\left(A-\left(B^{\wedge} 3\right)\right) *\left(A *\left(A-\left(B^{\wedge} 3\right)\right)\right)\)
using 1 by auto
moreover then have \(\ldots=\left(A-\left(B^{\wedge} 3\right)\right) *\left(A * A-\left(A *\left(B^{\wedge} 3\right)\right)\right)\)
by (metis minus-left-distributivity)
```

    moreover then have ... = (A - (B`3))*(A*A - (B`2))
        using inverse1
    by (simp add: power2-eq-square power3-eq-cube)
    moreover then have ... = A*(A*A - (B`2)) - (B`3)*(A*A - (B`2))
        by (metis minus-right-distributivity)
    moreover then have ... = ((A^3)-B)-B+(B`5)
    proof-
    have }A*(A*A-(B`2))=(A*A*A-A*(\mp@subsup{B}{}{`}2)
        by (simp add: right-diff-distrib)
    moreover have ... =( A*A*A - A*(B*B))
        by (metis power2-eq-square)
    moreover have ... = ((A^3) - ((A::rat-poly)*B)*B)
        by (simp add: power3-eq-cube)
    moreover have ... = ((A^3) - ((1::rat-poly)*B))
        by (metis inverse1)
    moreover have ... = (A`3) - B
        by auto
    ultimately have s1:(A::rat-poly)*(A*A - (B^2)) = (A`3) - (B::rat-poly)
        by metis
    have s2:((B::rat-poly)`3)*(A*A - (B`2)) = (B`3)*(A*A) - (B`(3::nat))*(B`2)
        by (metis minus-left-distributivity power3-eq-cube)
    moreover then have ... = (((B::rat-poly)^3)*(A*A) - (B`5))
        using power-add
        proof-
        have (B`(3::nat))*(B`2) = (B`5)
                            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))-(B`5))         by (metis power3-eq-cube) moreover then have ... = ((((B::rat-poly)*(B*(B*A)*A))) - (B`5))
by auto
moreover then have ... = ((((B::rat-poly)*(B*(1)*A))) - (B`5))         using inverse2 by auto     moreover then have ... = ((((B::rat-poly)*(B*A))) - (B^5))         by auto     moreover then have ... = ((((B::rat-poly))) - (B`5))
using inverse2
by simp
ultimately have ((B::rat-poly)^3)*(A*A-(B^2)) = ((B::rat-poly ) - (B^5))
by auto
then have }A*(A*A-(\mp@subsup{B}{}{`}2))-(B`3)*(A*A - (B`2)         =(A^3) - (B::rat-poly) - ((B::rat-poly ) - (B`5))
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)

```
```

(rat-poly-times A (A - rat-poly-times (rat-poly-times B B) B)))
=((A^3) - B ) - B + (B`5)

```
            by auto
    then have \(? l=B-\left(B^{\wedge} 5\right)+\left(\left(A^{\wedge} 3\right)-B\right)-B+\left(B^{\wedge} 5\right)\)
        using 3 by auto
    then have \(4: ? l=(A \sim 3)-B\)
        by auto
    have ?r \(=A *((A-\) rat-poly-times \((\) rat-poly-times \(B B) B) * A)\)
        by auto
    moreover then have \(\ldots=A *\left(A-\left(B^{\wedge} 3\right)\right) * A\)
        using 1 by auto
    moreover have \(\ldots=A *\left(A * A-\left(B^{\wedge} 3\right) * A\right)\)
        by (simp add: minus-left-distributivity mult.commute)
    moreover have \(\ldots=A *(A * A-(B * B * B) * A)\)
        by (metis power3-eq-cube)
    moreover have \(\ldots=A *(A * A-(B * B *(B * A)))\)
        by auto
    moreover have \(\ldots=A *(A * A-B * B)\)
        using inverse2 minus-left-distributivity by auto
    moreover have \(\ldots=A * A * A-A *(B * B)\)
        by (metis minus-left-distributivity rat-poly.comm)
    moreover have \(\ldots=A^{\wedge} 3-(A * B) * B\)
        by (metis ab-semigroup-mult-class.mult-ac(1) power3-eq-cube)
    moreover have \(\ldots=A^{\wedge} 3-B\)
        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:
    rat-poly-plus (rat-poly-times \(A\) (rat-poly-times ( \(B\) - rat-poly-times (rat-poly-times
A A) A) A))
    (rat-poly-times \((B-\) rat-poly-times (rat-poly-times \(A A) A)\)
            \((\) rat-poly-times \(B(B-\) rat-poly-times \((\) rat-poly-times \(A A) A))=\)
    rat-poly-times \(B\) (rat-poly-times \((B-\) rat-poly-times (rat-poly-times \(A A) A) B)\)
(is \(? l=? r\) )
proof-
    have \(1:(B-\) rat-poly-times (rat-poly-times \(A A) A)\)
        \(=B-\left(A^{\wedge} 3\right)\)
        by (metis power3-eq-cube)

        \(=B * A-\left(A^{\wedge} 3\right) * A\)
    by (metis minus-right-distributivity)
    then have \(\ldots=1-\left(A^{\wedge}\right)\)
    using inverse2
    by (metis mult.commute one-plus-numeral power-add power-one-right
semiring-norm(2)
    semiring-norm(4))
then have (rat-poly-times \(A\) (rat-poly-times ( \(B\) - rat-poly-times (rat-poly-times A A) A) A) )
\[
=A-(A \wedge 4) * A
\]
using 2
by (simp add: minus-left-distributivity)
then have 3:(rat-poly-times \(A\) (rat-poly-times ( \(B\) - rat-poly-times (rat-poly-times A A) A) A) )
\[
=A-(A \bumpeq 5)
\]
by (simp add: numeral-Bit0 numeral-Bit1)
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 \(A A) A))\)
\[
=(B-(A \sim 3)) *(B *(B-(A \sim 3)))
\]
using 1 by auto
moreover then have \(\ldots=\left(B-\left(A^{\wedge} 3\right)\right) *\left(B * B-\left(B *\left(A^{\wedge} 3\right)\right)\right)\) by (metis minus-left-distributivity)
moreover then have \(\ldots=\left(B-\left(A^{\wedge} 3\right)\right) *\left(B * B-\left(A^{\wedge} 2\right)\right)\)
using inverse2
by (simp add: power2-eq-square power3-eq-cube)
moreover then have \(\ldots=B *(B * B-(A \wedge 2))-(A \wedge 3) *(B * B-(A \wedge 2))\)
by (metis minus-right-distributivity)
moreover then have \(\ldots=\left(\left(B^{\wedge} 3\right)-A\right)-A+\left(A^{\wedge} 5\right)\)
proof -
have \(B *\left(B * B-\left(A^{\wedge} 2\right)\right)=(B * B * B-B *(A \subset 2))\)
by (simp add: right-diff-distrib)
moreover have \(\ldots=(B * B * B-B *(A * A))\) by (metis power2-eq-square)
moreover have \(\ldots=\left(\left(B^{\wedge} 3\right)-((B::\right.\) rat-poly \(\left.) * A) * A\right)\) by (simp add: power3-eq-cube)
moreover have \(\ldots=\left(\left(B^{\wedge} 3\right)-((1::\right.\) rat-poly \(\left.) * A)\right)\) by (metis inverse2)
moreover have \(\ldots=\left(B^{\wedge} 3\right)-A\) by auto
ultimately have s1:(B::rat-poly \() *\left(B * B-\left(A^{\wedge} 2\right)\right)=\left(B^{\wedge} 3\right)-(A::\) rat-poly \()\) by metis
have s2: \(\left((A::\right.\) rat-poly \(\left.) \wedge_{3} 3\right) *\left(B * B-\left(A^{\wedge} 2\right)\right)=\left(A^{\wedge} 3\right) *(B * B)-\left(A^{\wedge}(3::\right.\) nat \(\left.)\right) *\left(A^{\wedge} 2\right)\) by (metis minus-left-distributivity power3-eq-cube)
moreover then have \(\ldots=\left(\left((A:: \text { rat-poly })^{\wedge} 3\right) *(B * B)-(A \wedge 5)\right)\)
using power-add
proof -
have \((A \wedge(3:: n a t)) *\left(A^{\wedge} 2\right)=A^{\wedge} 5\)
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 \(\ldots=\left((((A::\right.\) rat-poly \(\left.) * A * A) *(B * B))-\left(A^{\wedge} 5\right)\right)\) by (metis power3-eq-cube)
moreover then have..\(=\left((((A::\right.\) rat-poly \(\left.) *(A *(A * B) * B)))-\left(A^{\wedge} 5\right)\right)\) by auto
moreover then have \(\ldots=\left((((A::\right.\) rat-poly \(\left.) *(A *(1) * B)))-\left(A^{\wedge} 5\right)\right)\)

> using inverse1 by auto
moreover then have \(\ldots=((((A::\) rat-poly \() *(A * B)))-(A \uparrow 5))\)
by auto
moreover then have \(\ldots=\left((((\right.\) A::rat-poly \(\left.)))-\left(A^{\wedge} 5\right)\right)\)
using inverse1 by auto
ultimately have \(\left((A:: \text { rat-poly })^{\wedge} 3\right) *(B * B-(A \wedge 2))=\left((A::\right.\) rat-poly \(\left.)-\left(A^{\wedge} 5\right)\right)\)
by auto
then have \(B *\left(B * B-\left(A^{\wedge} 2\right)\right)-(A \subset 3) *(B * B-(A \subset 2))\)
\[
=(B \wedge 3)-(A:: \text { rat-poly })-((A:: \text { rat-poly })-(A \wedge 5))
\]

> 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 \(A A) A))\)
\[
=\left(\left(B^{\wedge} 3\right)-A\right)-A+(A \wedge 5)
\]
by auto
then have ?l \(=A-\left(A^{\wedge} 5\right)+\left(\left(B^{\wedge} 3\right)-A\right)-A+\left(A^{\wedge} 5\right)\)
using 3 by auto
then have \(4: ? l=\left(B^{\wedge} 3\right)-A\)
by auto
have \(? r=B *((B-\) rat-poly-times \((\) rat-poly-times \(A A) A) * B)\) by auto
moreover then have \(\ldots=B *\left(B-\left(A^{\wedge} 3\right)\right) * B\)
using 1 by auto
moreover have \(\ldots=B *\left(B * B-\left(A^{\wedge} 3\right) * B\right)\)
using minus-left-distributivity by (simp add: minus-left-distributivity
mult.commute)
moreover have \(\ldots=B *(B * B-(A * A * A) * B)\)
by (metis power3-eq-cube)
moreover have \(\ldots=B *(B * B-(A * A *(A * B)))\) by auto
moreover have \(\ldots=B *(B * B-A * A)\) using inverse1 by auto
moreover have \(\ldots=B * B * B-B *(A * A)\) by (metis minus-left-distributivity rat-poly.comm)
moreover have \(\ldots=B^{\wedge} 3-(B * A) * A\) by (metis ab-semigroup-mult-class.mult-ac(1) power3-eq-cube)
moreover have \(\ldots=B^{\wedge} 3-A\)
using inverse2 by (metis monoid-mult-class.mult.left-neutral)
ultimately have \(? r=B^{\wedge} 3-A\)
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 \(B(B-r a t-p o l y-t i m e s(r a t-p o l y-t i m e s ~ A ~ A) ~ A)))\)

A))
using computation2 by auto
lemma computation-toppos:rat-poly-inv (rat-poly-times ( \(A\) - rat-poly-times (rat-poly-times \(B B) B) A)=\) rat-poly-times \((B-r a t-p o l y-t i m e s(r a t-p o l y-t i m e s ~ A A) A) B(\mathbf{i s} ? l=? r)\) proof-
have (rat-poly-times \((A\) - rat-poly-times (rat-poly-times \(B B) B) A\) )
\[
=((A-((B * B) * B)) * A)
\]
by auto
moreover then have \(\ldots=(A * A)-((B * B) * B) * A\) by (metis minus-left-distributivity rat-poly.comm)
moreover then have \(\ldots=(A * A)-(B * B) *(B * A)\) by auto
moreover then have \(\ldots=(A * A)-(B * B)\) using inverse2 by auto
ultimately have ?l= 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 \(\ldots=B * B-((A * A) * A) * B\)
by (metis minus-left-distributivity rat-poly.comm)
moreover have \(\ldots=(B * B)-((A * A) *(A * B))\)
by auto
moreover have \(\ldots=((B::\) rat-poly \() * B)-(A * A)\)
using inverse 1 by auto
ultimately have ? \(r=(B * B)-(A * A)\)
by auto
then show ?thesis using 1 by auto
qed
lemma computation-downpos-prelim:
rat-poly-inv (rat-poly-times ( \(B\) - rat-poly-times (rat-poly-times \(A\) A) A) B) \(=\) rat-poly-times \((A-r a t-p o l y-t i m e s ~(r a t-p o l y-t i m e s ~ B ~ B) B) A(i s ? l=? r)\)
proof-
have (rat-poly-times \((B\) - rat-poly-times (rat-poly-times \(A A) A) B)\)
\[
=((B-((A * A) * A)) * B)
\]
by auto
moreover then have \(\ldots=(B * B)-((A * A) * A) * B\)
by (metis minus-left-distributivity rat-poly.comm)
moreover then have \(\ldots=(B * B)-(A * A) *(A * B)\)
by auto
moreover then have \(\ldots=(B * B)-(A * A)\)
using inverse1 by auto
ultimately have ?l = 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 \(\ldots=A * A-((B * B) * B) * A\)
by (metis minus-left-distributivity rat-poly.comm)
moreover have \(\ldots=(A * A)-((B * B) *(B * A))\)
by auto
moreover have \(\ldots=((A::\) 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 \(A(A-\) rat-poly-times (rat-poly-times B B) \(B)=\)
rat-poly-inv (rat-poly-times \(B(B-\) rat-poly-times (rat-poly-times \(A A) A))\) using computation-downpos-prelim by (metis rat-poly.comm)
lemma computatition-positive-flip:rat-poly-plus
(rat-poly-inv (rat-poly-times \(A\) (rat-poly-times ( \(A\) - rat-poly-times (rat-poly-times B B) B) A)))
(rat-poly-inv \((\) rat-poly-times \(B(\) rat-poly-times \(A B)))=\)
rat-poly-inv (rat-poly-times \(A(\) rat-poly-times \(A A))(\) is \(? l=? r)\)
proof-
have (rat-poly-inv (rat-poly-times \(B\) (rat-poly-times \(A B)\) ) \(=(\) rat-poly-inv \((\) rat-poly-times B 1) \()\)
using inverse 1 by auto
moreover have \(\ldots=-B\)
by auto
ultimately have \(1:(\) rat-poly-inv \((\) rat-poly-times \(B(\) rat-poly-times \(A B)))=-B\) by auto
have (rat-poly-times \(A\) (rat-poly-times ( \(A\) - rat-poly-times (rat-poly-times B B) B) A))
\(=A *((A-((B * B) * B)) * A)\)
by auto
moreover then have \(\ldots=A *((A * A)-((B * B) * B * A))\)
by (metis minus-left-distributivity rat-poly.comm)
moreover then have \(\ldots=A *((A * A)-((B * B) * 1))\)
using inverse2 by auto
moreover then have \(\ldots=A *((A * A)-(B * B))\)
by auto
moreover then have \(\ldots=A *(A * A)-(A *(B * B))\)
by (metis minus-left-distributivity)
moreover then have \(\ldots=(A *(A * A))-(1 * B)\)
using inverse1 by auto
moreover then have \(\ldots=(A *(A * A))-B\)
by auto
ultimately have (rat-poly-times \(A\) (rat-poly-times ( \(A\) - rat-poly-times (rat-poly-times \(B B) B) A\) )
\[
=(A *(A * A))-B
\]
by auto
then have rat-poly-inv (rat-poly-times \(A\) (rat-poly-times ( \(A\) - rat-poly-times (rat-poly-times \(B\) B) \(B) A\) )
\[
=B-(A * A * A)
\]
by auto
then have \(3: ? l=-(A * A * A)\)
using 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 ( \(B\) - 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 \(1:(\) rat-poly-inv \((\) rat-poly-times \(A(\) rat-poly-times \(B A)))=-A\)
by auto
have (rat-poly-times \(B\) (rat-poly-times ( \(B\) - rat-poly-times (rat-poly-times \(A\) )
A) \(B\) )
\(=B *((B-((A * A) * A)) * B)\)
by auto
moreover then have \(\ldots=B *((B * B)-((A * A) * A * B))\)
by (metis minus-left-distributivity rat-poly.comm)
moreover then have \(\ldots=B *((B * B)-((A * A) * 1))\)
using inverse 1 by auto
moreover then have \(\ldots=B *((B * B)-(A * A))\)
by auto
moreover then have \(\ldots=B *(B * B)-(B *(A * A))\)
by (metis minus-left-distributivity)
moreover then have \(\ldots=(B *(B * B))-(1 * A)\)
using inverse2 by auto
moreover then have \(\ldots=(B *(B * B))-A\)
by auto
ultimately have (rat-poly-times \(B\) (rat-poly-times ( \(B\) - rat-poly-times (rat-poly-times

A A) A) B)
\[
=(B *(B * B))-A
\]
by auto
then have rat-poly-inv (rat-poly-times \(B\) (rat-poly-times ( \(B\) - rat-poly-times (rat-poly-times A A) A) B) )
\[
=A-(B * B * B)
\]
by auto
then have \(3: ? l=-(B * B * B)\)
using 1 by auto
moreover have ? \(r=-(B * B * B)\)
by auto
ultimately show ?thesis by auto
qed
lemma computation-pull-pos-neg:
rat-poly-plus (rat-poly-times \(B(B-\) rat-poly-times (rat-poly-times \(A A) A))\)
(rat-poly-times \((A-\) rat-poly-times \((\) rat-poly-times \(B B) B) A)=0\) proof-
have rat-poly-times (rat-poly-times A A) A
\[
=((A * A) * A)
\]
by auto
then have rat-poly-times \(B(B-\) rat-poly-times (rat-poly-times \(A A) A)\)
\[
=B * B-B *((A * A) * A)
\]
using minus-left-distributivity by auto
moreover have \(\ldots=B * B-(B *(A *(A * A)))\)
by auto
moreover have \(\ldots=B * B-((B * A) *(A * A))\)
by auto
moreover have \(\ldots=B * B-A * A\)
using inverse2 by auto
ultimately have 1:rat-poly-times \(B\) ( \(B\) - rat-poly-times (rat-poly-times \(A A\) )
A)
\[
=B * B-A * A
\]
by auto
have rat-poly-times (rat-poly-times \(B\) ) \(B=(B * B) * B\)
by auto
then have (rat-poly-times \((A\) - rat-poly-times (rat-poly-times B B) B) A)
\[
=(A * A)-((B * B) * B) * A
\]
using minus-right-distributivity by auto
moreover have \(\ldots=(A * A)-((B * B) *(B * A))\)
by auto
moreover have \(\ldots=(A * A)-(B * B)\)
using inverse2 by auto
ultimately have 2:(rat-poly-times \((A-\) rat-poly-times (rat-poly-times \(B B) B\) )
A)
\[
\begin{aligned}
& =(A * A)-(B * B) \\
& \text { by auto }
\end{aligned}
\]
have \(B * B-A * A+(A * A)-(B * B)=0\)

\section*{by auto}
with 12 show ?thesis by auto
qed
lemma aux1:( \(A\) - rat-poly-times (rat-poly-times B B) B)
\[
=A-\left(B^{\wedge} 3\right)
\]
using power3-eq-cube by (metis)
lemma square-subtract:(( \(p::\) rat-poly \()-(q::\) rat-poly \(\left.))^{\wedge} 2\right)\)
\[
=(p \text { ^2 })-(2 * p * q)+(q \text { ค2 })
\]
proof -
have \(1:(((p::\) rat-poly \()-(q:\) rat-poly \())\) ~2 \()\)
\(=(p-q) *(p-q)\)
by (metis power2-eq-square)
then have \((p-q) *(p-q)=(p-q) * p-(p-q) * q\)
by (metis minus-right-distributivity rat-poly.comm)
moreover have \((p-q) * p=p * p-q * p\)
by (metis minus-left-distributivity rat-poly.comm)
moreover have \((p-q) * q=p * q-q * q\)
by (metis minus-left-distributivity rat-poly.comm)
ultimately have \((p-q) *(p-q)=p * p-q * p-(p * q-q * q)\) by auto
moreover have \(\ldots=(p * p)-q * p-p * q+q * q\)
by auto
moreover have \(\ldots=\left(p^{\wedge} 2\right)-p * q-p * q+\left(q^{\wedge} 2\right)\)
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::rat-poly) - (q::rat-poly)) ^3)
\[
\left.=\left(p^{\wedge} 3\right)-3 *\left(p^{\wedge} 2\right) *(q)+3 *(p) *\left(q^{\wedge} 2\right)-\left(q^{\wedge} 3\right)\right)
\]
apply(rule allI)
apply(rule allI)
proof-
fix \(p q\)
have \(1:\left(((p:: \text { rat-poly })-(q:: \text { rat-poly }))^{\wedge} 3\right)=(p-q) *(p-q)^{\wedge} 2\)
by (metis One-nat-def Suc-1 numeral-3-eq-3 power-Suc)
then have \((p-q)^{\wedge} 2=\left(p^{\wedge} 2\right)-(2 * p * q)+\left(q^{\wedge 2}\right)\)
using square-subtract by auto
then have 2: \((p-q) *(p-q)^{\wedge} 2=(p-q) *\left(\left(p^{\wedge} 2\right)-(2 * p * q)+\left(q^{\wedge} 2\right)\right)\)
by auto
moreover have 3: \((p-q) *\left(\left(p^{\wedge}\right.\right.\) 2 \()-(2 * p * q)+\left(q^{\text {^2 } 2)}\right)\)
\[
=p *((p \text { 2) }-(2 * p * q)+(q \text { ค } 2))
\]
\(-\left(q *\left(\left(p^{\text {へ2 }}\right)-(2 * p * q)+\left(q^{\text {へ2 }}\right)\right)\right)\)
by (metis minus-right-distributivity)
moreover have \(p *\left(\left(p^{\wedge} 2\right)-(2 * p * q)+\left(q^{\wedge} 2\right)\right)\)
\[
=p *(p \wedge 2)-p *(2 * p * q)+\left(p *\left(q^{\wedge} 2\right)\right)
\]
using minus-left-distributivity by (simp add: distrib-left)
moreover have \(p *\left(p^{\wedge} 2\right)=p^{\wedge} 3\)
by (metis One-nat-def Suc-1 numeral-3-eq-3 power-Suc)
moreover have \(p *(2 * p * q)=2 *\left(p^{\wedge} 2\right) * q\)
by (metis (no-types, lifting) distrib-left mult-2 power2-eq-square
semigroup-mult-class.mult.assoc)
ultimately have \(4: p *\left(\left(p^{\wedge}\right.\right.\) 2 \(\left.)-(2 * p * q)+\left(q^{\wedge} 2\right)\right)\)
\[
=(p 3)-(2 *(p-2) * q)+(p *(q-2))
\]
by auto
have \(q *\left(\left(p^{\text {^2 }}\right)-(2 * p * q)+\left(q^{\text {^2 }}\right)\right)\)
\[
=q *(p \wedge 2)-q *(2 * p * q)+(q *(q \wedge 2))
\]
by (simp add: distrib-left minus-left-distributivity)
moreover have \(q *\left(p^{\wedge} 2\right)=\left(p^{\wedge} 2\right) * q\)
by \(\operatorname{simp}\)
moreover have \(q *(2 * p * q)=2 * p *\left(q^{\wedge} 2\right)\)
by (simp add: power2-eq-square)
ultimately have 5:q*((p^2)-(2*p*q)+(q^2))
\[
=(p \text { 2 }) * q-2 * p *(q 2)+(q 3)
\]
by (metis One-nat-def Suc-1 numeral-3-eq-3 power-Suc)
with 1234 have \((p-q)^{\wedge} 3\)
\[
\begin{aligned}
= & \left(p^{\wedge} 3\right)-(2 *(p \text { ^2 }) * q)+(p *(q \text { ^2 })) \\
& -\left(\left(p^{\wedge} 2\right) * q-2 * p *(q \text { 2 })+\left(q^{\wedge} 3\right)\right)
\end{aligned}
\]
by auto
moreover have \(\ldots=\left(p^{\wedge} 3\right)-\left(2 *\left(p^{\wedge} 2\right) * q\right)+\left(p *\left(q^{\wedge} 2\right)\right)\) \(-\left(p^{\wedge} 2\right) * q+2 * p *\left(q^{\wedge} 2\right)-\left(q^{\wedge} 3\right)\)
by auto
moreover have \(\ldots=\left(p^{\wedge} 3\right)-3 *\left(p^{\wedge} 2\right) *(q)+3 *(p) *\left(q^{\wedge} 2\right)-\left(q^{\wedge} 3\right)\)
by auto
ultimately show
\((p-q){ }^{\wedge} 3\)
\(=\) rat-poly-plus \(\left(p^{\wedge} 3-\right.\)
rat-poly-times
(rat-poly-times \(\left.3\left(p^{2}\right)\right)\) )
(rat-poly-times (rat-poly-times \(3 p\) ) \(\left(q^{2}\right)\) )
\[
-q^{\wedge} 3
\]
by auto
qed
lemma power-mult: \(((p::\) :rat-poly \() \uparrow m) \wedge n=(p) \bigwedge(m *(n:: n a t))\)
by (metis power-mult)
lemma cube-minus2:
fixes \(p q\)
shows \((((p::\) rat-poly \()-(q::\) rat-poly \()) \wedge 3)\)
\[
=\left(p^{\wedge} 3\right)-3 *(p \wedge 2) *(q)+3 *(p) *\left(q^{\wedge} 2\right)-\left(q^{\wedge} 3\right)
\]
using cube-minus by auto
lemma subst-poly:assumes \(a=b\) shows ( \(p:\) :rat-poly) \(* a=p * b\)
using assms by auto
```

lemma sub1:
assumes }p*q=
shows r*(p*q) =r*1
using assms by metis
lemma n-distrib:(A^(n::nat))*(B`n)=(A*B)^n by (induct n)(auto) lemma rat-poly-id-pow:(1::rat-poly)`n}=
by (induct n)(auto)
lemma power-prod:(A`(n::nat))*(B`n) = (1::rat-poly)
apply(simp add:n-distrib)
apply(simp add:inverse1)
done
lemma (pCons 0 1)}\not=
by (metis non-zero var-def)
end

```

\section*{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^(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))))
\[
=\text { mat1 }\left(\mathcal{Q}^{\wedge}(\text { nat }(\text { domain-wall } b))\right)
\]
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))))
```

    =(kauff-mat b)
    proof-
have mat (rat-poly.row-length (kauff-mat b)) (\mp@subsup{\mathbb{N`}}{}{`}(\mathrm{ nat (codomain-wall b))) (kauff-mat}

```
    b)
    by (metis assms matrix-kauff-mat)
moreover have (blockmat (make-vert-block (nat (codomain-wall b))))
                    \(=\) mat1 \(\left(\mathcal{Z}^{\wedge}(\right.\) nat \((\) codomain-wall b) \())\)
    using make-vert-block-map-blockmat by auto
ultimately show ?thesis by (metis mat1-rt-mult)
qed
lemma compress-top-inv:(compress-top w1 w2 ) \(\Longrightarrow\) kauff-mat w1 \(=\) kauff-mat w2 proof-
assume assm:compress-top w1 w2
have \(\exists B .((w 1=(\) basic \((\) make-vert-block \((\) nat \((\) domain-wall B) \())) \circ B)\)
\[
\wedge(w 2=(B \circ(\text { basic }([])))) \wedge(\text { codomain-wall } B=0)
\]
\(\wedge(\) is-tangle-diagram B) \()\)
using compress-top-def assm by auto
then obtain \(B\) where \((w 1=(\) basic \((\) make-vert-block \((\) nat \((\) domain-wall \(B)))) \circ\) B)
\[
\wedge(w 2=(B \circ(\text { basic }([])))) \wedge(\text { codomain-wall } B=
\]
0) \(\wedge(\) is-tangle-diagram B)
by auto
then have \(1:(w 1=(\) basic \((\) make-vert-block \((\) nat \((\) domain-wall \(B)))) \circ B)\) \(\wedge(\) w2 \(=(B \circ(\) basic \(([])))) \wedge(\) codomain-wall \(B=\)
0) \(\wedge(\) is-tangle-diagram B)
by auto
then have kauff-mat \((w 1)=(\) kauff-mat \(((\) basic (make-vert-block (nat (domain-wall \(B))\) ) \(\circ B\) )
by auto
moreover then have \(\ldots=\) kauff-mat \(((\) make-vert-block \((\) nat \((\) domain-wall \(B))) * B)\) by auto
moreover then have \(\ldots=\) rat-poly.matrix-mult (blockmat (make-vert-block (nat
(domain-wall \(B)\) )))
(kauff-mat B)
by auto
moreover then have \(\ldots=(\) kauff-mat \(B)\)
using 1 mat1-vert-wall-left by (metis)
ultimately have kauff-mat (w1) = kauff-mat B
by auto
moreover have kauff-mat w2 \(=\) kauff-mat \(B\)
using 1 by (metis left-mat-compose)
ultimately show ?thesis by auto
qed
lemma domain-make-vert-int: \((n \geq 0) \Longrightarrow\) (domain-block (make-vert-block (nat \(n)\) ))
\[
=n
\]
using domain-make-vert by auto
lemma compress-bottom-inv:(compress-bottom w1 w2) \(\Longrightarrow\) kauff-mat w1 \(=\) kauff-mat w2
proof-
assume assm:compress-bottom w1 w2
have \(\exists B .((w 1=B \circ(\) basic \((\) make-vert-block \((\) nat \((\operatorname{codomain-wall~B)}))))\)
\(\wedge(\) w2 \(=((\) basic \(([]) \circ B))) \wedge(\) domain-wall \(B=0)\)
\(\wedge(\) is-tangle-diagram \(B))\)
using compress-bottom-def assm by auto
then obtain \(B\) where ( \((w 1=B \circ\) (basic (make-vert-block (nat (codomain-wall B)) )) )
```

$\wedge(w 2=(($ basic $([]) \circ B))) \wedge($ domain-wall $B=0)$

```
\(\wedge(\) is-tangle-diagram \(B))\)
by auto
then have \(1:((w 1=B \circ(\) basic \((\) make-vert-block \((\) nat \((\operatorname{codomain-wall~B)}))))\) \(\wedge(w 2=((\) basic \(([]) \circ B))) \wedge(\) domain-wall \(B=0)\)
\(\wedge(\) is-tangle-diagram B) \()\)
by auto
then have kauff-mat \((w 1)=(\) kauff-mat \((B \circ\) (basic (make-vert-block (nat (codomain-wall \(B)\) )) ))
by auto
moreover then have \(\ldots=\) rat-poly.matrix-mult (kauff-mat B)
(kauff-mat (basic (make-vert-block (nat (codomain-wall
\(B)\) ) ) )
proof-
have is-tangle-diagram \(B\)
using 1 by auto
moreover have is-tangle-diagram (basic (make-vert-block (nat (codomain-wall B))))
using is-tangle-diagram.simps by auto
moreover have codomain-wall \(B=\) domain-wall (basic (make-vert-block (nat
(codomain-wall B))))
proof-
have codomain-wall \(B \geq 0\)
apply (induct \(B\) )
by (auto) (metis codomain-block-nonnegative)
then have domain-block (make-vert-block (nat (codomain-wall B)))
= codomain-wall B
using domain-make-vert-int by auto
then show ?thesis unfolding domain-wall.simps(1) by auto
qed
ultimately show ?thesis using tangle-compose-matrix by auto
qed
moreover then have...\(=\) rat-poly.matrix-mult (kauff-mat B)
(blockmat (make-vert-block (nat (codomain-wall B))))
using kauff-mat.simps(1) tangle-compose-matrix by auto
moreover then have \(\ldots=(\) kauff-mat \(B)\)
using 1 mat1-vert-wall-right by auto
ultimately have kauff-mat \((w 1)=\) kauff-mat \(B\)
by auto
moreover have kauff-mat w2 \(=\) kauff-mat \(B\)
using 1 by (metis right-mat-compose)
ultimately show ?thesis by auto
qed
```

theorem compress-inv:compress w1 w2 \Longrightarrow(kauff-mat w1 = kauff-mat w2)
unfolding compress-def using compress-bottom-inv compress-top-inv
by auto
lemma striaghten-topdown-computation:kauff-mat ((basic ([vert,cup]))\circ(basic ([cap,vert])))
= kauff-mat ((basic ([vert]))\circ(basic ([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 \Longrightarrow(kauff-mat w1)=
(kauff-mat w2)
unfolding straighten-topdown-def using striaghten-topdown-computation by auto
lemma striaghten-downtop-computation:kauff-mat ((basic ([cup,vert]))\circ(basic ([vert,cap])))
= kauff-mat ((basic ([vert]))\circ(basic ([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 \Longrightarrow(kauff-mat w1)=
(kauff-mat w2)
unfolding straighten-downtop-def using striaghten-downtop-computation by auto
theorem straighten-inv:straighten w1 w2 \Longrightarrow(kauff-mat w1) = (kauff-mat w2)
unfolding straighten-def using straighten-topdown-inv straighten-downtop-inv by
auto

```
lemma kauff-mat-swingpos:
    kauff-mat (r-over-braid) \(=\) kauff-mat (l-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 \(\Longrightarrow\) (kauff-mat w1) \(=(\) kauff-mat w2 \()\)
unfolding swing-pos-def using kauff-mat-swingpos by auto
lemma kauff-mat-swingneg:
kauff-mat \((r\)-under-braid \()=\) kauff-mat \((l\)-under-braid \()\)
apply ( \(\operatorname{simp}\) )
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 \(\Longrightarrow\) (kauff-mat w1) \(=(\) kauff-mat w2 \()\)
unfolding swing-neg-def using kauff-mat-swingneg by auto
theorem swing-inv:
swing w1 w2 \(\Longrightarrow(\) kauff-mat w1 \()=(\) kauff-mat w2 \()\)
unfolding swing-def using swing-pos-inv swing-neg-inv by auto
lemma rotate-toppos-kauff-mat:kauff-mat ((basic [vert,over])○(basic [cap, vert]))
    \(=\) kauff-mat \(((\) basic \([\) under, vert \(]) \circ(\) basic \([\) vert, cap \(]))\)
apply ( \(\operatorname{simp\text {)}}\)
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 \(\Longrightarrow\) (kauff-mat w1) \(=(\) kauff-mat w2)
unfolding rotate-toppos-def using rotate-toppos-kauff-mat by auto
lemma rotate-topneg-kauff-mat:kauff-mat ((basic [vert,under])॰(basic [cap, vert]))
\(=\) kauff-mat \(((\) basic \([\) over,vert \(]) \circ(\) basic \([\) vert, cap \(]))\)
apply (simp add:mat-multI-def)
apply (simp add:matT-vec-multI-def)
apply(auto simp add:replicate-def rat-poly.row-length-def)
apply(auto simp add:scalar-prod)
```

apply(simp add:computation-toppos)
done
lemma rotate-topneg-inv:rotate-topneg w1 w2 \Longrightarrow(kauff-mat w1) = (kauff-mat
w2)
unfolding rotate-topneg-def using rotate-topneg-kauff-mat by auto
lemma rotate-downpos-kauff-mat:
kauff-mat ((basic [cup,vert])\circ(basic [vert,over]))= kauff-mat ((basic [vert,cup])\circ(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 \(\Longrightarrow(\) kauff-mat w1 \()=(\) kauff-mat w2)
unfolding rotate-downpos-def using rotate-downpos-kauff-mat by auto
lemma rotate-downneg-kauff-mat:
kauff-mat \(((\) basic \([\) cup,vert \(]) \circ(\) basic \([\) vert,under \(]))=\) kauff-mat \(((\) basic \([\) vert, cup \(]) \circ(\) 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 \(\Longrightarrow(\) kauff-mat w1 \()=(\) kauff-mat w2)
unfolding rotate-downneg-def using rotate-downneg-kauff-mat by auto
```

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

```
lemma positive-fip-kauff-mat:
kauff-mat (left-over) \(=\) kauff-mat (right-over \()\)
```

apply(simp add:mat-multI-def)
apply (simp add:matT-vec-multI-def)
apply(auto simp add:replicate-def rat-poly.row-length-def)
apply(auto simp add:scalar-prod)
using computatition-positive-flip apply auto[1]
using computatition-positive-flip by auto
lemma uncross-positive-flip-inv: uncross-positive-flip w1 w2 $\Longrightarrow$ (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 $\Longrightarrow$ (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 $) \Longrightarrow($ 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] $))$

$$
=\text { kauff-mat }((\text { basic }[\text { vert }, \text { vert }]) \circ(\text { basic }[\text { vert,vert }]))
$$

$\operatorname{apply}$ (simp add:mat-multI-def)
apply (simp add:matT-vec-multI-def)
$\operatorname{apply}($ 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 \(\Longrightarrow\) (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 \(\Longrightarrow\) (kauff-mat w1) \(=(\) kauff-mat w2 \()\) unfolding pull-negpos-def using neg-pos-kauff-mat by auto
theorem pull-inv:pull w1 w2 \(\Longrightarrow\) (kauff-mat w1) \(=(\) kauff-mat w2 \()\)
unfolding pull-def using pull-posneg-inv pull-negpos-inv by auto
theorem slide-inv:slide w1 w2 \(\Longrightarrow\) (kauff-mat w1 = kauff-mat w2)
proof-
assume assm:slide w1 w2
have \(\exists B .((w 1=((\) basic \((\) make-vert-block \((\) nat \((\) domain-block B) \())) \circ(\) basic B) \())\) \(\wedge(w 2=((\) 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 \(\quad((w 1=\) ( 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) = 0)) by auto

```
then have \(1:((w 1=((\) 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^(nat (domain-block B))) (length (blockmat B)) (blockmat B)
by (metis matrix-blockmat row-length-domain-block)
have \(w 1=((\) basic \((\) make-vert-block \((\) nat \((\) domain-wall \((\) basic B) \())) \circ(\) basic B) \())\)
using 1 domain-wall.simps by auto
then have kauff-mat \(w 1=\) rat-poly.matrix-mult
(kauff-mat (basic (make-vert-block (nat (domain-wall
\((\) basic B))))))
(kauff-mat (basic B))
using tangle-compose-matrix is-tangle-diagram.simps
by (metis compose-Nil kauff-mat.simps(1) kauff-mat.simps(2))
moreover then have \(\ldots=\) rat-poly.matrix-mult (mat1 (2^(nat (domain-block
B)))) (blockmat B)
using kauff-mat.simps(1) domain-wall.simps(1) by (metis make-vert-block-map-blockmat)
moreover have \(\ldots=(\) 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) \circ(basic (make-vert-block (nat (codomain-wall (basic
B))))),
using }1\mathrm{ 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 \Longrightarrow(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

\section*{14 Kauffman_Invariance: Proving the invariance of Kauffman Bracket}

\author{
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 \(\Longrightarrow\) 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\mathrm{ - 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```

