# Knot Theory

T. V. H. Prathamesh

December 14, 2021

**Abstract**

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

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

## Contents

# 1 Preliminaries: Definitions of tangles and links

**theory** *Preliminaries*
**imports** *Main*
**begin**

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

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

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

block is obtained by putting bricks next to each other

**type-synonym** *block* = *brick list*

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

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

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

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

**lemma** *empty-concatenate*: *xs* ⊗ *Nil* = *xs*
⟨*proof*⟩

Associativity properties of Conscatenation

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

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

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

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

**primrec** *compose* :: *wall* => *wall* => *wall* (**infixr** ∘ *66*) **where**
*compose-Nil*: (*basic x*) ∘ *ys* = *prod x ys* |
*compose-Cons*: ((*prod x xs*)∘*ys*) = *prod x* (*xs*∘*ys*)

Associativity properties of composition

**lemma** *compose-leftassociativity*: (((*x*::*wall*) ∘ *y*) ∘ *z*) = (*x*∘*y* ∘*z*)
⟨*proof*⟩

**lemma** *compose-rightassociativity*: (*x*::*wall*) ∘ (*y* ∘ *z*) = (*x*∘*y* ∘*z*)
⟨*proof*⟩

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

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

**primrec** *domain*::*brick* ⇒ *int*
**where**
*domain vert* = *1*|
*domain cup* = *0*|
*domain cap* = *2*|
*domain over* = *2*|
*domain under* = *2*

**lemma** *domain-non-negative*:∀ *x*.(*domain x*) ≥ *0*

⟨*proof*⟩

**primrec** *codomain*::*brick* ⇒ *int*

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

**primrec** *domain-block*::*block* ⇒ *int*
**where**
*domain-block* [] = *0*
|*domain-block* (*Cons x y*) = (*domain x* + (*domain-block y*))

**lemma** *domain-block-non-negative*:*domain-block xs* ≥ *0*
⟨*proof*⟩

**primrec** *codomain-block*::*block* ⇒ *int*
**where**
*codomain-block* [] = *0*
|*codomain-block* (*Cons x y*) = (*codomain x* + (*codomain-block y*))

**primrec** *domain-wall*:: *wall* ⇒ *int* **where**
*domain-wall* (*basic x*) = *domain-block x*
|*domain-wall* (*x*∗*ys*) = *domain-block x*

**fun** *codomain-wall*:: *wall* ⇒ *int* **where**
*codomain-wall* (*basic x*) = *codomain-block x*
|*codomain-wall* (*x*∗*ys*) = *codomain-wall ys*

**lemma** *domain-wall-compose*: *domain-wall* (*xs*∘*ys*) = *domain-wall xs*
⟨*proof*⟩

**lemma** *codomain-wall-compose*: *codomain-wall* (*xs*∘*ys*) = *codomain-wall ys*
⟨*proof*⟩

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

absolute value

**definition** *abs*::*int* ⇒ *int* **where**

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

theorems about abs

**lemma** *abs-zero*: **assumes** *abs* $x = 0$ **shows** $x = 0$
⟨*proof*⟩

**lemma** *abs-zero-equality*: **assumes** *abs* $(x - y) = 0$ **shows** $x = y$
⟨*proof*⟩

**lemma** *abs-non-negative*: *abs* $x \geq 0$
⟨*proof*⟩

**lemma** *abs-non-negative-sum*: **assumes** *abs* $x$ + *abs* $y = 0$
**shows** *abs* $x = 0$ **and** *abs* $y = 0$
⟨*proof*⟩

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

**lemma** *domain-nonnegative*: $(domain\ x) \geq 0$
⟨*proof*⟩

**lemma** *codomain-nonnegative*: $(codomain\ x) \geq 0$
⟨*proof*⟩

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

**lemma** *domain-block-nonnegative*: *domain-block* $x \geq 0$
⟨*proof*⟩

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

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

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

$\Longrightarrow (domain\text{-}block\ (x\#y) > 0)$
⟨*proof*⟩

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

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

⟨*proof*⟩

**lemma** *domain-zero-sum*: **assumes** (*domain-block x*) + (*domain-block y*) = *0*
**shows** *domain-block x* = *0* **and** *domain-block y* = *0*
⟨*proof*⟩

**lemma** *domain-block-positive*: **fixes** *or* **assumes** *domain-block y>0 or domain-block y>0*
**shows** (*domain-block* (*x*⊗*y*)) > *0*
⟨*proof*⟩

**lemma** *codomain-block-positive*: **fixes** *or* **assumes** *codomain-block y>0 or codomain-block y>0*
**shows** (*codomain-block* (*x*⊗*y*)) > *0*
⟨*proof*⟩

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

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

**primrec** *is-cup*::*block* ⇒ *bool*
**where**
*is-cup* [] = *True*|
*is-cup* (*x*#*y*) = (*if* (*x*= *cup*) *then* (*is-cup y*) *else False*)

**lemma** *brickcount-zero-implies-cup*:(*domain x*= *0*) ⟹ (*x* = *cup*)
⟨*proof*⟩

**lemma** *brickcount-zero-implies-brick-is-cup*:(*domain x*= *0*) ⟹ (*brick-is-cup x*)
⟨*proof*⟩

**lemma** *domain-zero-implies-is-cup*:(*domain-block x*= *0*) ⟹ (*is-cup x*)
⟨*proof*⟩

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

**primrec** *is-tangle-diagram*::*wall* ⇒ *bool*
**where**

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

**definition** *is-link-diagram*::*wall* ⇒ *bool*
**where**
*is-link-diagram x* ≡ (*if* (*is-tangle-diagram x*)
*then* (*abs* (*domain-wall x*) + *abs*(*codomain-wall x*) = *0*)
*else False*)

**end**

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

**theory** *Tangles*
**imports** *Preliminaries*
**begin**

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

**typedef** *Tangle-Diagram* = {(*x*::*wall*). *is-tangle-diagram x*}
⟨*proof*⟩

**typedef** *Link-Diagram* = {(*x*::*wall*). *is-link-diagram x*}
⟨*proof*⟩

The next few lemmas list the properties of well defined diagrams

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

**lemma** *Abs-Rep-well-defined*:
**assumes** *is-tangle-diagram x*
**shows** *Rep-Tangle-Diagram* (*Abs-Tangle-Diagram x*) = *x*
⟨*proof*⟩

The map Abs_diagram is injective

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

⟨*proof*⟩

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*⟩

**lemma** *is-tangle-right-compose*:
*is-tangle-diagram* $(x \circ y) \Longrightarrow$ *is-tangle-diagram* $y$
⟨*proof*⟩

**lemma** *comp-of-tangle-dgms*:
**assumes***is-tangle-diagram* $y$
**shows** ((*is-tangle-diagram* $x$)
$\wedge$(*codomain-wall* $x$ = *domain-wall* $y$))
$\Longrightarrow$ *is-tangle-diagram* $(x \circ y)$
⟨*proof*⟩

**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* $(x \circ y)$
⟨*proof*⟩

**lemma** *converse-composition-of-tangle-diagrams*:
*is-tangle-diagram* $(x \circ y) \Longrightarrow$ (*domain-wall* $y$) = (*codomain-wall* $x$)
⟨*proof*⟩

**definition** *compose-Tangle*::*Tangle-Diagram* ⇒ *Tangle-Diagram* ⇒ *Tangle-Diagram*

(**infixl** ∘ *65*)
**where**
*compose-Tangle* $x$ $y$ = *Abs-Tangle-Diagram*
((*Rep-Tangle-Diagram* $x$) ∘ (*Rep-Tangle-Diagram* $y$))

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

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

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

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

**end**

# 3 Tangle_Algebra: Tensor product of tangles and its properties

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

# 4 Definition of tensor product of walls

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

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

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

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

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

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

# 5 Properties of tensor product of tangles

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

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

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

**definition** *tensor-Tangle* :: *Tangle-Diagram* ⇒ *Tangle-Diagram* ⇒ *Tangle-Diagram*
(**infixl** ⊗ *65*)
**where**
*tensor-Tangle x y* = *Abs-Tangle-Diagram* ((*Rep-Tangle-Diagram x*) ⊗ (*Rep-Tangle-Diagram y*))

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

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

**lemma** *tensor-domain-wall-additivity*:
*domain-wall* $(xs \otimes ys)$ = *domain-wall xs* + *domain-wall ys*
⟨*proof*⟩

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

**lemma** *tensor-codomain-wall-additivity*:
*codomain-wall* $(xs \otimes ys)$ = *codomain-wall xs* + *codomain-wall ys*
⟨*proof*⟩

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

(*is-tangle-diagram xs*)
⟹ *is-tangle-diagram* (*xs* ⊗ (*basic* (*make-vert-block n*)))
⟨*proof*⟩

**theorem** *is-tangle-make-vert-left*:
(*is-tangle-diagram xs*) ⟹ *is-tangle-diagram* ((*basic* (*make-vert-block n*)) ⊗ *xs*)
⟨*proof*⟩

**lemma** *simp1*: (*codomain-block y*) ≠ *0* ⟹
*is-tangle-diagram* (*xs*)
∧ *is-tangle-diagram* ((*basic* (*make-vert-block* (*nat* (*codomain-block y*))))) ⟶
*is-tangle-diagram* (*xs* ⊗ ((*basic* (*make-vert-block* (*nat* (*codomain-block y*)))))) ⟹
(*is-tangle-diagram* (*x* ∗ *xs*) ∧ *is-tangle-diagram* (*basic y*) ⟶ *is-tangle-diagram* (*x* ∗ *xs* ⊗ *basic y*))

⟨*proof*⟩

**lemma** *simp2*:
(*codomain-block x*) ≠ *0*
⟹
*is-tangle-diagram* (*basic* (*make-vert-block* (*nat* (*codomain-block x*))))
∧ *is-tangle-diagram* (*ys*)
⟶
*is-tangle-diagram* ((*basic* (*make-vert-block* (*nat* (*codomain-block x*)))) ⊗ *ys*)
⟹
(*is-tangle-diagram* (*basic x*)
∧ *is-tangle-diagram* (*y*∗*ys*)
⟶ *is-tangle-diagram* ((*basic x*) ⊗ (*y*∗*ys*)))
⟨*proof*⟩

**lemma** *simp3*:
*is-tangle-diagram xs* ∧ *is-tangle-diagram ys* ⟶ *is-tangle-diagram* (*xs* ⊗ *ys*)
⟹
*is-tangle-diagram* (*x* ∗ *xs*) ∧ *is-tangle-diagram* (*y* ∗ *ys*)
⟶ *is-tangle-diagram* (*x* ∗ *xs* ⊗ *y* ∗ *ys*)
⟨*proof*⟩

**theorem** *is-tangle-diagramness*:
**shows**(*is-tangle-diagram x*)∧(*is-tangle-diagram y*) ⟶*is-tangle-diagram* (*tensor x y*)
⟨*proof*⟩

**theorem** *tensor-preserves-is-tangle*:
**assumes** *is-tangle-diagram x*
**and** *is-tangle-diagram y*
**shows** *is-tangle-diagram* $(x \otimes y)$
⟨*proof*⟩

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

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

**theorem** *well-defined-compose*:
**assumes** *is-tangle-diagram x*
**and** *is-tangle-diagram y*
**shows** (*Abs-Tangle-Diagram x*) ⊗ (*Abs-Tangle-Diagram y*) = (*Abs-Tangle-Diagram* (*x* ⊗ *y*))
⟨*proof*⟩

**end**
**theory** *Tangle-Relation*
**imports** *Main*
**begin**

**lemma** *symmetry1*: **assumes** *symp R*
**shows** ∀ *x y*. (*x*, *y*) ∈ {(*x*, *y*). *R x y*}* ⟶ (*y*, *x*) ∈ {(*x*, *y*). *R x y*}*
⟨*proof*⟩

**lemma** *symmetry2*: **assumes** ∀ *x y*. (*x*, *y*) ∈ {(*x*, *y*). *R x y*}* ⟶ (*y*, *x*) ∈ {(*x*, *y*). *R x y*}*
**shows** *symp R*^**
⟨*proof*⟩

**lemma** *symmetry3*: **assumes** *symp R* **shows** *symp R*^** ⟨*proof*⟩

**lemma** *symm-trans*: **assumes** *symp R* **shows** *symp R*^++ ⟨*proof*⟩

**end**

# 6 Tangle_Moves: Defining moves on tangles

**theory** *Tangle-Moves*
**imports** *Tangles Tangle-Algebra Tangle-Relation*

**begin**

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

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

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

**type-synonym** *relation* = *wall* ⇒ *wall* ⇒ *bool*

Link uncross

**abbreviation** *right-over*::*wall*
**where**
*right-over* ≡ ((*basic* [*vert*,*cup*]) ∘ (*basic* [*over*,*vert*])∘(*basic* [*vert*,*cap*]))

**abbreviation** *left-over*::*wall*
**where**
*left-over* ≡ ((*basic* (*cup*#*vert*#[])) ∘ (*basic* (*vert*#*over*#[]))
∘ (*basic* (*cap*#*vert*#[])))

**abbreviation** *right-under*::*wall*
**where**
*right-under* ≡ ((*basic* (*vert*#*cup*#[])) ∘ (*basic* (*under*#*vert*#[]))
∘ (*basic* (*vert*#*cap*#[])))

**abbreviation** *left-under*::*wall*
**where**
*left-under* ≡ ((*basic* (*cup*#*vert*#[])) ∘ (*basic* (*vert*#*under*#[]))
∘ (*basic* (*cap*#*vert*#[])))

**abbreviation** *straight-line*::*wall*
**where**
*straight-line* ≡ (*basic* (*vert*#[])) ∘ (*basic* (*vert*#[])) ∘ (*basic* (*vert*#[]))

**definition** *uncross-positive-flip*::*relation*
**where**
*uncross-positive-flip x y* ≡ ((*x* = *right-over*)∧(*y* = *left-over*))

**definition** *uncross-positive-straighten*::*relation*
**where**
*uncross-positive-straighten x y* ≡ ((*x* = *right-over*)∧(*y* = *straight-line*))

**definition** *uncross-negative-flip*::*relation*
**where**
*uncross-negative-flip x y* ≡ ((*x* = *right-under*)∧(*y* = *left-under*))

**definition** *uncross-negative-straighten*::*relation*
**where**
*uncross-negative-straighten x y* ≡ ((*x* = *left-under*)∧(*y* = *straight-line*))

**definition** *uncross*
**where**
*uncross x y* ≡ ((*uncross-positive-straighten x y*)∨(*uncross-positive-flip x y*)
∨(*uncross-negative-straighten x y*)∨ (*uncross-negative-flip x y*))

swing begins

**abbreviation** *r-over-braid*::*wall*
**where**
*r-over-braid* ≡ ((*basic* ((*over*#*vert*#[]))∘(*basic* ((*vert*#*over*#[])))
∘(*basic* (*over*# *vert*#[]))))

**abbreviation** *l-over-braid*::*wall*
**where**
*l-over-braid* ≡ (*basic* (*vert*#*over*#[]))∘(*basic* (*over*#*vert*#[]))
∘(*basic* (*vert*#*over*#[]))

**abbreviation** *r-under-braid*::*wall*
**where**
*r-under-braid* ≡ ((*basic* ((*under*#*vert*#[]))∘(*basic* ((*vert*#*under*#[])))
∘(*basic* (*under*# *vert*#[]))))

**abbreviation** *l-under-braid*::*wall*
**where**
*l-under-braid* ≡ (*basic* (*vert*#*under*#[]))∘(*basic* (*under*#*vert*#[]))
∘(*basic* (*vert*#*under*#[]))

**definition** *swing-pos*::*wall* ⇒ *wall* ⇒ *bool*
**where**

*swing-pos x y* ≡ (*x* = *r-over-braid*)∧(*y* = *l-over-braid*)

**definition** *swing-neg*::*wall* ⇒ *wall* ⇒ *bool*
**where**
*swing-neg x y* ≡(*x* = *r-under-braid*)∧(*y*=*l-under-braid*)

**definition** *swing*::*relation*
**where**
*swing x y* ≡ (*swing-pos x y*)∨(*swing-neg x y*)

pull begins

**definition** *pull-posneg*::*relation*
**where**
*pull-posneg x y* ≡ ((*x* = ((*basic* (*over*#[])∘(*basic* (*under*#[])))
∧(*y* = ((*basic* (*vert*#*vert*#[]))
∘(*basic* ((*vert*#*vert*#[])))))

**definition** *pull-negpos*::*relation*
**where**
*pull-negpos x y* ≡ ((*x* = ((*basic* (*under*#[])∘(*basic* (*over*#[])))
∧(*y* = ((*basic* (*vert*#*vert*#[]))
∘(*basic* ((*vert*#*vert*#[])))))

pull definition

**definition** *pull*::*relation*
**where**
*pull x y* ≡ ((*pull-posneg x y*) ∨ (*pull-negpos x y*))

linkrel-pull ends

linkrel-straighten

**definition** *straighten-topdown*::*relation*
**where**
*straighten-topdown x y* ≡ ((*x* =((*basic* ((*vert*#*cup*#[]))
∘(*basic* ((*cap*#*vert*#[])))))
∧(*y* = ((*basic* (*vert*#[])∘(*basic* (*vert*#[])))))

**definition** *straighten-downtop*::*relation*
**where**
*straighten-downtop x y* ≡ ((*x* =((*basic* ((*cup*# *vert*#[]))
∘(*basic* ((*vert*# *cap*#[])))))
∧(*y* = ((*basic* (*vert*#[])∘(*basic* (*vert*#[])))))

definition straighten

**definition** *straighten*::*relation*
**where**

*straighten x y* ≡ ((*straighten-topdown x y*) ∨ (*straighten-downtop x y*))

straighten ends

rotate moves

**definition** *rotate-toppos*::*relation*
**where**
*rotate-toppos x y* ≡ ((*x* = ((*basic* ((*vert* #*over*#[])))
∘(*basic* ((*cap*# *vert*#[])))))
∧ (*y* = ((*basic* ((*under*#*vert*#[]))
∘(*basic* ((*vert*#*cap*#[])))))))

**definition** *rotate-topneg*::*wall* ⇒ *wall* ⇒ *bool*
**where**
*rotate-topneg x y* ≡ ((*x* = ((*basic* ((*vert* #*under*#[])))
∘(*basic* ((*cap*# *vert*#[])))))
∧ (*y* = ((*basic* ((*over*#*vert*#[]))
∘(*basic* ((*vert*#*cap*#[])))))))

**definition** *rotate-downpos*::*wall* ⇒ *wall* ⇒ *bool*
**where**
*rotate-downpos x y* ≡ ((*x* = ((*basic* (*cup*#*vert*#[]))
∘(*basic* (*vert*#*over*#[]))))
∧ (*y* = ((*basic* ((*vert*#*cup*#[])))
∘(*basic* ((*under*#*vert*#[]))))))

**definition** *rotate-downneg*::*wall* ⇒ *wall* ⇒ *bool*
**where**
*rotate-downneg x y* ≡ ((*x* = ((*basic* (*cup*#*vert*#[]))
∘(*basic* (*vert*#*under*#[]))))
∧ (*y* = ((*basic* ((*vert*#*cup*#[])))
∘(*basic* ((*over*#*vert*#[]))))))

rotate definition

**definition** *rotate*::*wall* ⇒ *wall* ⇒ *bool*
**where**
*rotate x y* ≡ ((*rotate-toppos x y*) ∨ (*rotate-topneg x y*)
∨ (*rotate-downpos x y*) ∨ (*rotate-downneg x y*))

rotate ends

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

**definition** *compress-top1*::*wall* ⇒ *wall* ⇒ *bool*
**where**
*compress-top1 x y* ≡ ∃ *B*.((*x* = (*basic* (*make-vert-block* (*nat* (*domain-wall B*))))∘ *B*)
∧(*y* = *B*)∧(*codomain-wall B* ≠ *0*)
∧(*is-tangle-diagram B*))

**definition** *compress-bottom1*::*wall* ⇒ *wall* ⇒ *bool*
**where**
*compress-bottom1 x y* ≡ ∃ *B*.((*x* = *B* ∘ (*basic* (*make-vert-block* (*nat* (*codomain-wall B*)))))
∧(*y* = *B*))∧(*domain-wall B* ≠ *0*)
∧(*is-tangle-diagram B*)

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

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

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

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

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

linkrel-definition

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

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

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

**lemma** *framed-uncross-implies-uncross*: (*framed-uncross x y*)⟹(*uncross x y*)
⟨*proof*⟩

**end**

# 7 Link_Algebra: Defining equivalence of tangles and links

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

**inductive** *Tangle-Equivalence* :: *wall* ⇒ *wall* ⇒ *bool* (**infixl** ~ *64*)
**where**
*refl* [*intro*!, *Pure.intro*!, *simp*]: *a* ~ *a*
|*equality* [*Pure.intro*]: *linkrel a b* ⟹ *a* ~ *b*
|*domain-compose*:(*domain-wall a* = *0*)∧(*is-tangle-diagram a*) ⟹ *a* ~ ((*basic* [])∘*a*)
|*codomain-compose*:(*codomain-wall a* = *0*) ∧ (*is-tangle-diagram a*) ⟹ *a* ~ (*a* ∘ (*basic* []))
|*compose-eq*:((*B*::*wall*) ~ *D*) ∧ ((*A*::*wall*) ~ *C*)
∧(*is-tangle-diagram A*)∧(*is-tangle-diagram B*)
∧(*is-tangle-diagram C*)∧(*is-tangle-diagram D*)
∧(*domain-wall B*)= (*codomain-wall A*)
∧(*domain-wall D*)= (*codomain-wall C*)
⟹((*A*::*wall*) ∘ *B*) ~ (*C* ∘ *D*)
|*trans*: *A*~*B* ⟹ *B*~*C* ⟹ *A* ~ *C*
|*sym*:*A*~ *B* ⟹ *B* ~*A*
|*tensor-eq*: ((*B*::*wall*) ~ *D*) ∧ ((*A*::*wall*) ~ *C*) ∧(*is-tangle-diagram A*)∧(*is-tangle-diagram B*)
∧(*is-tangle-diagram C*)∧(*is-tangle-diagram D*) ⟹((*A*::*wall*) ⊗ *B*) ~ (*C* ⊗ *D*)

**inductive** *Framed-Tangle-Equivalence* :: *wall* ⇒ *wall* ⇒ *bool* (**infixl** ~*f 64*)
**where**

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

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

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

**quotient-type** *Tangle* = *Tangle-Diagram*/*Tangle-Diagram-Equivalence*
**morphisms** *Rep-Tangles Abs-Tangles*
⟨*proof*⟩

**quotient-type** *Link* = *Link-Diagram*/*Link-Diagram-Equivalence*
**morphisms** *Rep-Links Abs-Links*
⟨*proof*⟩

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

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

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

**quotient-type** *Framed-Tangle* = *Tangle-Diagram*
/*Framed-Tangle-Diagram-Equivalence*
**morphisms** *Rep-Framed-Tangles Abs-Framed-Tangles*
⟨*proof*⟩

**quotient-type** *Framed-Link* = *Link-Diagram*/*Framed-Link-Diagram-Equivalence*
**morphisms** *Rep-Framed-Links Abs-Framed-Links*
⟨*proof*⟩

**end**

# 8 Showing equivalence of links: An example

**theory** *Example*
**imports** *Link-Algebra*
**begin**

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

**lemma** *transitive*: **assumes** *a*~*b* **and** *b*~*c* **shows** *a*~*c*
⟨*proof*⟩

**lemma** *prelim-cup-compress*:
((*basic* (*cup*#[])) ∘ (*basic* (*vert* # *vert* # [])) ~
((*basic* [])∘(*basic* (*cup*#[]))
⟨*proof*⟩

**lemma** *cup-compress*:
(*basic* (*cup*#[])) ∘ (*basic* (*vert* # *vert* # [])) ~ (*basic* (*cup*#[]))
⟨*proof*⟩

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

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

**lemma** *uncross-straighten-left-over*:*left-over* ~ *straight-line*
⟨*proof*⟩

**theorem** *Example*:
$x \sim y$
⟨*proof*⟩

**end**

# 9 Kauffman Matrix and Kauffman Bracket- Definitions and Properties

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

# 10 Rational Functions

intpoly is the type of integer polynomials

**type-synonym** *intpoly* = *int poly*

**lemma** *eval-pCons*: *poly* (*pCons 0 1*) *x* = *x*
⟨*proof*⟩

**lemma** *pCons2*: (*pCons 0 1*) ≠ (*1*::*int poly*)
⟨*proof*⟩

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

**lemma** *non-zero*:*x* ≠ *0*
⟨*proof*⟩

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

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

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

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

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

**lemma assumes** $b \neq 0$ **and** $d \neq 0$
**shows** *Fract a b* = *Fract c d* $\longleftrightarrow$ $a * d = c * b$
⟨*proof*⟩

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

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

**abbreviation** *rat-poly-times*::*rat-poly* ⇒ *rat-poly* ⇒ *rat-poly*
**where**
*rat-poly-times p q* ≡ $p*q$

**abbreviation** *rat-poly-plus*::*rat-poly* ⇒ *rat-poly* ⇒ *rat-poly*
**where**
*rat-poly-plus p q* ≡ $p+q$

**abbreviation** *rat-poly-inv*::*rat-poly* ⇒ *rat-poly*
**where**
*rat-poly-inv p* ≡ $(-\ p)$

**interpretation** *rat-poly*:*semiring-0 rat-poly-plus 0 rat-poly-times*
⟨*proof*⟩

**interpretation** *rat-poly*:*semiring-1 1 rat-poly-times rat-poly-plus 0*
⟨*proof*⟩

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

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*
⟨*proof*⟩

**lemma** *rat-poly.matrix-mult* $[[A,1],[0,A]]$ $[[A,0],[0,A]]$ = $[[A*A,A],[0,A*A]]$
⟨*proof*⟩

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

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

**lemma** [[*1*]] ⊗ *M* = *M*
⟨*proof*⟩

**lemma** *M* ⊗ [[*1*]] = *M*
⟨*proof*⟩

# 11 Kauffman matrices

We assign every brick to a matrix of rational polynmials

**primrec** *brickmat*::*brick* ⇒ *rat-poly mat*
**where**
*brickmat vert* = [[*1*,*0*],[*0*,*1*]]
|*brickmat cup* = [[*0*],[*A*],[−*B*],[*0*]]
|*brickmat cap* = [[*0*,−*A*,*B*,*0*]]
|*brickmat over* = [[*A*,*0*,*0*,*0*],
[*0*,*0*,*B*,*0*],
[*0*,*B*,*A*−(*B*∗*B*∗*B*),*0*],
[*0*,*0*,*0*,*A*]]
|*brickmat under* = [[*B*,*0*,*0*,*0*],
[*0*,*B*−(*A*∗*A*∗*A*),*A*,*0*],
[*0*,*A*,*0*,*0*],
[*0*,*0*,*0*,*B*]]

**lemma** *inverse1*:*rat-poly-times A B* = *1*
⟨*proof*⟩

**lemma** *inverse2*:*rat-poly-times B A* = *1*
⟨*proof*⟩

**lemma** *B-non-zero*:*B* ≠ *0*
⟨*proof*⟩

**lemma** *rat-poly-times p* (*q* + *r*)
= (*rat-poly-times p q*) + (*rat-poly-times p r*)
⟨*proof*⟩

**lemma** *minus-left-distributivity*:
*rat-poly-times p (q − r)*
*= (rat-poly-times p q) − (rat-poly-times p r)*
⟨*proof*⟩

**lemma** *minus-right-distributivity*:
*rat-poly-times (p − q) r = (rat-poly-times p r) − (rat-poly-times q r)*
⟨*proof*⟩

**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*⟩

**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]]*
⟨*proof*⟩

**lemma** *rat-poly-inv A = −A*
⟨*proof*⟩

**lemma** *vert-dim*:*rat-poly.row-length (brickmat vert) = 2 ∧length (brickmat vert) = 2*
⟨*proof*⟩
**lemma** *cup-dim*:*rat-poly.row-length (brickmat cup) = 1* **and** *length (brickmat cup) = 4*
⟨*proof*⟩
**lemma** *cap-dim*:*rat-poly.row-length (brickmat cap) = 4* **and** *length (brickmat cap) = 1*
⟨*proof*⟩
**lemma** *over-dim*:*rat-poly.row-length (brickmat over) = 4* **and** *length (brickmat over) = 4*
⟨*proof*⟩
**lemma** *under-dim*:*rat-poly.row-length (brickmat under) = 4* **and** *length (brickmat under) = 4*
⟨*proof*⟩
**lemma** *mat-vert*:*mat 2 2 (brickmat vert)*
⟨*proof*⟩
**lemma** *mat-cup*:*mat 1 4 (brickmat cup)*
⟨*proof*⟩
**lemma** *mat-cap*:*mat 4 1 (brickmat cap)*
⟨*proof*⟩
**lemma** *mat-over*:*mat 4 4 (brickmat over)*
⟨*proof*⟩
**lemma** *mat-under*:*mat 4 4 (brickmat under)*

⟨*proof*⟩

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

**lemma** (*rat-poly.row-length* (*brickmat d*)) = (*2*^(*nat* (*domain d*)))
⟨*proof*⟩

**lemma** *rat-poly.row-length* (*brickmat cup*) = *1*
⟨*proof*⟩

**lemma** *two*:(*Suc* (*Suc 0*)) = *2*
⟨*proof*⟩

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

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

**lemma** *blockmat* [*a*] = *brickmat a*
⟨*proof*⟩

**lemma** *nat-sum*:
**assumes** *a* ≥ *0* **and** *b* ≥ *0*
**shows** *nat* (*a*+*b*) = (*nat a*) + (*nat b*)
⟨*proof*⟩

**lemma** *rat-poly.row-length* (*blockmat ls*) = (*2*^(*nat* ((*domain-block ls*))))
⟨*proof*⟩

**lemma** *row-length-domain-block*:
*rat-poly.row-length* (*blockmat ls*) = (*2*^(*nat* ((*domain-block ls*))))
⟨*proof*⟩

**lemma** *length-codomain-block*:*length* (*blockmat ls*)
= (*2*^(*nat* ((*codomain-block ls*))))
⟨*proof*⟩

**lemma** *matrix-blockmat*:
*mat*
(*rat-poly.row-length* (*blockmat ls*))
(*length* (*blockmat ls*))

(*blockmat ls*)
⟨*proof*⟩

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

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

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

**theorem** *matrix-kauff-mat*:
((*is-tangle-diagram ws*)
⟹ (*rat-poly.row-length* (*kauff-mat ws*)) = 2^(*nat* (*domain-wall ws*))
∧ (*length* (*kauff-mat ws*)) = 2^(*nat* (*codomain-wall ws*))
∧ (*mat*
(*rat-poly.row-length* (*kauff-mat ws*))
(*length* (*kauff-mat ws*))
(*kauff-mat ws*)))
⟨*proof*⟩

**theorem** *effective-matrix-kauff-mat*:
**assumes** *is-tangle-diagram ws*
**shows** (*rat-poly.row-length* (*kauff-mat ws*)) = 2^(*nat* (*domain-wall ws*))
**and** *length* (*kauff-mat ws*) = 2^(*nat* (*codomain-wall ws*))
**and** *mat* (*rat-poly.row-length* (*kauff-mat ws*)) (*length* (*kauff-mat ws*))
(*kauff-mat ws*)
⟨*proof*⟩

**lemma** *mat-mult-equiv*:
*rat-poly.matrix-mult m1 m2* = *mat-mult* (*rat-poly.row-length m1*) *m1 m2*
⟨*proof*⟩

**theorem** *associative-rat-poly-mat*:
**assumes** *mat* (*rat-poly.row-length m1*) (*rat-poly.row-length m2*) *m1*
**and** *mat* (*rat-poly.row-length m2*) (*rat-poly.row-length m3*) *m2*
**and** *mat* (*rat-poly.row-length m3*) *nc m3*
**shows** *rat-poly.matrix-mult m1* (*rat-poly.matrix-mult m2 m3*)
= *rat-poly.matrix-mult* (*rat-poly.matrix-mult m1 m2*) *m3*
⟨*proof*⟩

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

**theorem** *link-diagram-matrix*:
**assumes** *is-link-diagram ws*
**shows** *mat 1 1* (*kauff-mat ws*)
⟨*proof*⟩

**theorem** *tangle-compose-matrix*:
((*is-tangle-diagram ws1*) ∧ (*is-tangle-diagram ws2*)
∧(*domain-wall ws2* = *codomain-wall ws1*)) ⟹
*kauff-mat* (*ws1* ∘ *ws2*) = *rat-poly.matrix-mult* (*kauff-mat ws1*) (*kauff-mat ws2*)
⟨*proof*⟩

**theorem** *left-mat-compose*:
**assumes** *is-tangle-diagram ws*
**and** *codomain-wall ws* = *0*
**shows** *kauff-mat ws* = (*kauff-mat* (*ws* ∘ (*basic* [])))
⟨*proof*⟩

**theorem** *right-mat-compose*:
**assumes** *is-tangle-diagram ws* **and** *domain-wall ws* = *0*
**shows** *kauff-mat ws* = (*kauff-mat* ((*basic* []) ∘*ws*))
⟨*proof*⟩

**lemma** *left-id-blockmat*:*blockmat* [] ⊗ *blockmat b* = *blockmat b*
⟨*proof*⟩

**lemma** *tens-assoc*:
∀ *a xs ys*.(*brickmat a* ⊗ (*blockmat xs* ⊗ *blockmat ys*)
= (*brickmat a* ⊗ *blockmat xs*) ⊗ *blockmat ys*)
⟨*proof*⟩

**lemma** *kauff-mat-tensor-distrib*:
∀ *xs*.∀ *ys*.(*kauff-mat* (*basic xs* ⊗ *basic ys*)
= *kauff-mat* (*basic xs*) ⊗ *kauff-mat* (*basic ys*))
⟨*proof*⟩

**lemma** *blockmat-tensor-distrib*:
(*blockmat* (*a* ⊗ *b*)) = (*blockmat a*) ⊗ (*blockmat b*)
⟨*proof*⟩

**lemma** *blockmat-non-empty*:∀ *bs*.(*blockmat bs* ≠ [])
⟨*proof*⟩

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

**lemma** *kauff-mat-non-empty*:
**fixes** *ws*
**assumes** *is-tangle-diagram ws*
**shows** *kauff-mat ws* ≠ []

⟨*proof*⟩

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

**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*)
⟨*proof*⟩

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

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

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

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

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

**lemma** *blockmat-make-vert*:
*blockmat* (*make-vert-block n*) = (*make-vert-equiv n*)
⟨*proof*⟩

**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*⟩

**abbreviation** *nat-mult*::*nat* ⇒ *nat* ⇒ *nat* (**infixl** ∗*n 65*)
**where**
*nat-mult a b* ≡ ((*a*::*nat*)∗*b*)

**lemma** *equal-div-mod*:**assumes** $((j::nat)\ div\ a) = (i\ div\ a)$
**and** $(j\ mod\ a) = (i\ mod\ a)$
**shows** $j = i$
⟨*proof*⟩

**lemma** *equal-div-mod2*:$(((j::nat)\ div\ a)\ = (i\ div\ a)$
$\wedge\ ((j\ mod\ a) = (i\ mod\ a))) = (j = i)$
⟨*proof*⟩

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

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

**lemma assumes** $a < (b*c)$
**shows** $((a::nat)\ div\ b) < c$
⟨*proof*⟩

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

**lemma** *rat-poly-unity*:*rat-poly-times 1 1 = 1*
⟨*proof*⟩

**lemma** $((P \wedge Q) \longrightarrow R) \Longrightarrow (P \longrightarrow Q \longrightarrow R)$
⟨*proof*⟩
**lemma** *length (mat1 2) = 2*
⟨*proof*⟩

**theorem** *make-vert-equiv-mat*:
*make-vert-equiv n = (mat1 (2^n))*
⟨*proof*⟩

**theorem** *make-vert-block-map-blockmat*:
*blockmat (make-vert-block n) = (mat1 (2^n))*
⟨*proof*⟩

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

**shows** *rat-poly.matrix-mult m1 (mat1 (nc)) = m1*
⟨*proof*⟩

**lemma** *mat1-vert-block*:
*rat-poly.matrix-mult*
*(blockmat b)*
*(blockmat (make-vert-block (nat (codomain-block b))))*
*= (blockmat b)*
⟨*proof*⟩

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

**definition** *weak-matrix-match*::
*rat-poly mat ⇒ rat-poly mat ⇒ rat-poly mat ⇒ bool*
**where**
*weak-matrix-match A1 A2 B1 ≡ (mat (rat-poly.row-length A1) (length A1) A1)*
*∧(mat (rat-poly.row-length A2) (length A2) A2)*
*∧(mat (rat-poly.row-length B1) 1 B1)*
*∧(A1 ≠ [])∧(A2 ≠ [])∧(B1 ≠ [])*
*∧ (length A1 = rat-poly.row-length A2)*

**theorem** *weak-distributivity1*:
*weak-matrix-match A1 A2 B1*
*⟹ ((rat-poly.matrix-mult A1 A2)⊗ B1)*
*= (rat-poly.matrix-mult (A1 ⊗ B1) (A2))*
⟨*proof*⟩

**definition** *weak-matrix-match2*::
*rat-poly mat ⇒ rat-poly mat ⇒ rat-poly mat ⇒ bool*
**where**
*weak-matrix-match2 A1 B1 B2 ≡ (mat (rat-poly.row-length A1) 1 A1)*
*∧(mat (rat-poly.row-length B1) (length B1) B1)*
*∧(mat (rat-poly.row-length B2) (length B2) B2)*
*∧(A1 ≠ [])∧(B1 ≠ [])∧(B2 ≠ [])*
*∧ (length B1 = rat-poly.row-length B2)*

**theorem** *weak-distributivity2*:
*weak-matrix-match2 A1 B1 B2*
*⟹ (A1⊗ (rat-poly.matrix-mult B1 B2))*
*= (rat-poly.matrix-mult (A1 ⊗ B1) (B2))*
⟨*proof*⟩

**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)*
⟨*proof*⟩

**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*)
⟨*proof*⟩

**lemma** *is-tangle-diagram-vert-block*:
*is-tangle-diagram* (*b*∗(*basic* (*make-vert-block* (*nat* (*codomain-block b*)))))
⟨*proof*⟩

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

**theorem** *Tensor-Invariance*:
(*is-tangle-diagram ws1*) ∧ (*is-tangle-diagram ws2*)
⟹ (*kauff-mat* (*ws1* ⊗ *ws2*) = (*kauff-mat ws1*) ⊗ (*kauff-mat ws2*))
⟨*proof*⟩

**end**

# 12 Computations: This section can be skipped

**theory** *Computations*
**imports** *Kauffman-Matrix*
**begin**

**lemma** *unlink-computation*:
*rat-poly-plus* (*rat-poly-times* (*rat-poly-times A A*) (*rat-poly-times A A*))
(*rat-poly-plus*
(*rat-poly-times 2* (*rat-poly-times A* (*rat-poly-times A* (*rat-poly-times B B*))))
(*rat-poly-times* (*rat-poly-times B B*) (*rat-poly-times B B*))) =
((*A*^*4*)+(*B*^*4*)+*2*)
⟨*proof*⟩

**lemma** *computation-swingpos*:
*rat-poly-plus* (*rat-poly-times B* (*rat-poly-times* (*A* − *rat-poly-times* (*rat-poly-times B B*) *B*) *B*))
(*rat-poly-times* (*A* − *rat-poly-times* (*rat-poly-times B B*) *B*)
(*rat-poly-times A* (*A* − *rat-poly-times* (*rat-poly-times B B*) *B*))) =
*rat-poly-times A* (*rat-poly-times* (*A* − *rat-poly-times* (*rat-poly-times B B*) *B*) *A*)
(**is** *?l* = *?r*)
⟨*proof*⟩

**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*⟩

**lemma** *computation-swingneg*:*rat-poly-times B* (*rat-poly-times* (*B* − *rat-poly-times*
(*rat-poly-times A A*) *A*) *B*) =
*rat-poly-plus*
(*rat-poly-times* (*B* − *rat-poly-times* (*rat-poly-times A A*) *A*)
(*rat-poly-times B* (*B* − *rat-poly-times* (*rat-poly-times A A*) *A*)))
(*rat-poly-times A* (*rat-poly-times* (*B* − *rat-poly-times* (*rat-poly-times A A*) *A*)
*A*))
⟨*proof*⟩

**lemma** *computation-toppos*:*rat-poly-inv* (*rat-poly-times* (*A* − *rat-poly-times* (*rat-poly-times*
*B B*) *B*) *A*) =
*rat-poly-times* (*B* − *rat-poly-times* (*rat-poly-times A A*) *A*) *B*(**is** *?l* = *?r*)
⟨*proof*⟩

**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* − *rat-poly-times* (*rat-poly-times B B*) *B*) *A*(**is** *?l* = *?r*)
⟨*proof*⟩

**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*))
⟨*proof*⟩

**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*⟩

**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*⟩

**lemma** *computation-pull-pos-neg*:
*rat-poly-plus* (*rat-poly-times B* (*B* − *rat-poly-times* (*rat-poly-times A A*) *A*))
(*rat-poly-times* (*A* − *rat-poly-times* (*rat-poly-times B B*) *B*) *A*) = *0*
⟨*proof*⟩

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

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

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

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

**lemma** *cube-minus2*:
**fixes** *p q*
**shows** (((*p*::*rat-poly*) − (*q*::*rat-poly*))^*3*)
= (*p*^*3*) − *3*∗(*p*^*2*)∗(*q*) + *3*∗(*p*)∗(*q*^*2*) − (*q*^*3*)
⟨*proof*⟩

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

**lemma** *sub1*:
**assumes** *p*∗*q* = *1*
**shows** *r*∗(*p*∗*q*) = *r*∗*1*
⟨*proof*⟩

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

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

**lemma** *power-prod*:(*A*^(*n*::*nat*))∗(*B*^*n*) = (*1*::*rat-poly*)
⟨*proof*⟩
**lemma** (*pCons 0 1*) ≠ *0*
⟨*proof*⟩

**end**

# 13 Tangle moves and Kauffman bracket

**theory** *Linkrel-Kauffman*
**imports** *Computations*
**begin**

**lemma** *mat1-vert-wall-left*:
**assumes** *is-tangle-diagram b*
**shows**
*rat-poly.matrix-mult* (*blockmat* (*make-vert-block* (*nat* (*domain-wall b*)))) (*kauff-mat b*)
= (*kauff-mat b*)
⟨*proof*⟩

**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*⟩

**lemma** *compress-top-inv*:(*compress-top w1 w2*) ⟹ *kauff-mat w1* = *kauff-mat w2*
⟨*proof*⟩

**lemma** *domain-make-vert-int*:(*n* ≥ *0*) ⟹ (*domain-block* (*make-vert-block* (*nat n*)))
= *n*
⟨*proof*⟩

**lemma** *compress-bottom-inv*:(*compress-bottom w1 w2*) ⟹ *kauff-mat w1* = *kauff-mat w2*
⟨*proof*⟩

**theorem** *compress-inv*:*compress w1 w2* ⟹ (*kauff-mat w1* = *kauff-mat w2*)
⟨*proof*⟩

**lemma** *striaghten-topdown-computation*:*kauff-mat* ((*basic* ([*vert*,*cup*]))∘(*basic* ([*cap*,*vert*])))
= *kauff-mat* ((*basic* ([*vert*]))∘(*basic* ([*vert*])))
⟨*proof*⟩

**theorem** *straighten-topdown-inv*:*straighten-topdown w1 w2* ⟹ (*kauff-mat w1*) = (*kauff-mat w2*)
⟨*proof*⟩

**lemma** *striaghten-downtop-computation*:*kauff-mat* ((*basic* ([*cup*,*vert*]))∘(*basic* ([*vert*,*cap*])))
= *kauff-mat* ((*basic* ([*vert*]))∘(*basic* ([*vert*])))
⟨*proof*⟩

**theorem** *straighten-downtop-inv*:*straighten-downtop w1 w2* ⟹ (*kauff-mat w1*) = (*kauff-mat w2*)
⟨*proof*⟩

**theorem** *straighten-inv*:*straighten w1 w2* ⟹ (*kauff-mat w1*) = (*kauff-mat w2*)
⟨*proof*⟩

**lemma** *kauff-mat-swingpos*:
*kauff-mat* (*r-over-braid*) = *kauff-mat* (*l-over-braid*)
⟨*proof*⟩

**lemma** *swing-pos-inv*:*swing-pos w1 w2* ⟹ (*kauff-mat w1*) = (*kauff-mat w2*)
⟨*proof*⟩

**lemma** *kauff-mat-swingneg*:
*kauff-mat* (*r-under-braid*) = *kauff-mat* (*l-under-braid*)
⟨*proof*⟩

**lemma** *swing-neg-inv*:*swing-neg w1 w2* ⟹ (*kauff-mat w1*) = (*kauff-mat w2*)
⟨*proof*⟩

**theorem** *swing-inv*:
*swing w1 w2* ⟹ (*kauff-mat w1*) = (*kauff-mat w2*)
⟨*proof*⟩

**lemma** *rotate-toppos-kauff-mat*:*kauff-mat* ((*basic* [*vert*,*over*])∘(*basic* [*cap*, *vert*]))
= *kauff-mat* ((*basic* [*under*,*vert*])∘(*basic* [*vert*,*cap*]))
⟨*proof*⟩

**lemma** *rotate-toppos-inv*:*rotate-toppos w1 w2* ⟹ (*kauff-mat w1*) = (*kauff-mat w2*)
⟨*proof*⟩

**lemma** *rotate-topneg-kauff-mat*:*kauff-mat* ((*basic* [*vert*,*under*])∘(*basic* [*cap*, *vert*]))

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

**lemma** *rotate-topneg-inv*:*rotate-topneg w1 w2* ⟹ (*kauff-mat w1*) = (*kauff-mat w2*)
⟨*proof*⟩

**lemma** *rotate-downpos-kauff-mat*:
*kauff-mat* ((*basic* [*cup*,*vert*])∘(*basic* [*vert*,*over*]))= *kauff-mat* ((*basic* [*vert*,*cup*])∘(*basic* [*under*,*vert*]))
⟨*proof*⟩

**lemma** *rotate-downpos-inv*:*rotate-downpos w1 w2* ⟹ (*kauff-mat w1*) = (*kauff-mat w2*)
⟨*proof*⟩

**lemma** *rotate-downneg-kauff-mat*:
*kauff-mat* ((*basic* [*cup*,*vert*])∘(*basic* [*vert*,*under*]))= *kauff-mat* ((*basic* [*vert*,*cup*])∘(*basic* [*over*,*vert*]))
⟨*proof*⟩

**lemma** *rotate-downneg-inv*:*rotate-downneg w1 w2* ⟹ (*kauff-mat w1*) = (*kauff-mat w2*)
⟨*proof*⟩

**theorem** *rotate-inv*:*rotate w1 w2* ⟹ (*kauff-mat w1*) = (*kauff-mat w2*)
⟨*proof*⟩

**lemma** *positive-flip-kauff-mat*:
*kauff-mat* (*left-over*) = *kauff-mat* (*right-over*)
⟨*proof*⟩

**lemma** *uncross-positive-flip-inv*: *uncross-positive-flip w1 w2* ⟹ (*kauff-mat w1*) = (*kauff-mat w2*)
⟨*proof*⟩

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

**lemma** *uncross-negative-flip-inv*: *uncross-negative-flip w1 w2* ⟹ (*kauff-mat w1*) = (*kauff-mat w2*)
⟨*proof*⟩

**theorem** *framed-uncross-inv*:(*framed-uncross w1 w2*) ⟹ (*kauff-mat w1*) = (*kauff-mat w2*)
⟨*proof*⟩

**lemma** *pos-neg-kauff-mat*:
*kauff-mat* ((*basic* [*over*]) ∘ (*basic* [*under*]))
= *kauff-mat* ((*basic* [*vert*,*vert*]) ∘ (*basic* [*vert*,*vert*]))
⟨*proof*⟩

**lemma** *pull-posneg-inv*: *pull-posneg w1 w2* ⟹ (*kauff-mat w1*) = (*kauff-mat w2*)
⟨*proof*⟩

**lemma** *neg-pos-kauff-mat*:*kauff-mat* ((*basic* [*under*]) ∘ (*basic* [*over*]))
= *kauff-mat* ((*basic* [*vert*,*vert*]) ∘ (*basic* [*vert*,*vert*]))
⟨*proof*⟩

**lemma** *pull-negpos-inv*:*pull-negpos w1 w2* ⟹ (*kauff-mat w1*) = (*kauff-mat w2*)
⟨*proof*⟩

**theorem** *pull-inv*:*pull w1 w2* ⟹ (*kauff-mat w1*) = (*kauff-mat w2*)
⟨*proof*⟩

**theorem** *slide-inv*:*slide w1 w2* ⟹ (*kauff-mat w1* = *kauff-mat w2*)
⟨*proof*⟩

**theorem** *framed-linkrel-inv*:*framed-linkrel w1 w2* ⟹ (*kauff-mat w1*)= (*kauff-mat w2*)
⟨*proof*⟩

**end**

# 14 Kauffman_Invariance: Proving the invariance of Kauffman Bracket

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

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

**theorem** *kauffman-invariance*:(*w1*::*wall*) ~*f w2* ⟹ *kauff-mat w1* = *kauff-mat w2*
⟨*proof*⟩

**lemma** *rat-poly-times A B* = *1*
⟨*proof*⟩

we calculate kauffman bracket of a few links

kauffman bracket of an unknot with zero crossings

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

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

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

**definition** *trefoil-polynomial*::*rat-poly*
**where**
*trefoil-polynomial* ≡
*rat-poly-plus*
(*rat-poly-times* (*rat-poly-times A A*)
(*rat-poly-plus*
(*rat-poly-times B*
(*rat-poly-times B*
(*rat-poly-times* (*A* − *rat-poly-times* (*rat-poly-times B B*) *B*)
(*rat-poly-times A A*))))
(*rat-poly-times* (*A* − *rat-poly-times* (*rat-poly-times B B*) *B*)
(*rat-poly-plus* (*rat-poly-times B* (*rat-poly-times B* (*rat-poly-times A A*)))
(*rat-poly-times* (*A* − *rat-poly-times* (*rat-poly-times B B*) *B*)
(*rat-poly-times* (*A* − *rat-poly-times* (*rat-poly-times B B*) *B*)
(*rat-poly-times A A*)))))))
(*rat-poly-plus*
(*rat-poly-times 2*
(*rat-poly-times A*
(*rat-poly-times A*
(*rat-poly-times A*
(*rat-poly-times A* (*rat-poly-times A* (*rat-poly-times B B*)))))))
(*rat-poly-times* (*rat-poly-times B B*)
(*rat-poly-times B*
(*rat-poly-times* (*A* − *rat-poly-times* (*rat-poly-times B B*) *B*)
(*rat-poly-times B* (*rat-poly-times B B*))))))

kauffman bracket of trefoil

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

**end**
**theory** *Knot-Theory*
**imports** *Kauffman-Invariance Example*
**begin**

**end**