Julius-Maximilians-Universität Würzburg
Chair for INFORMATICS I
Efficient Algorithms and Knowledge-Based Systems
Institute for Informatics

The Complexity of Finding Tangles

Oksana Firman, Philipp Kindermann,
Alexander Wolff, **Johannes Zink**

Julius-Maximilians-Universität Würzburg,
Germany

Alexander Ravsky

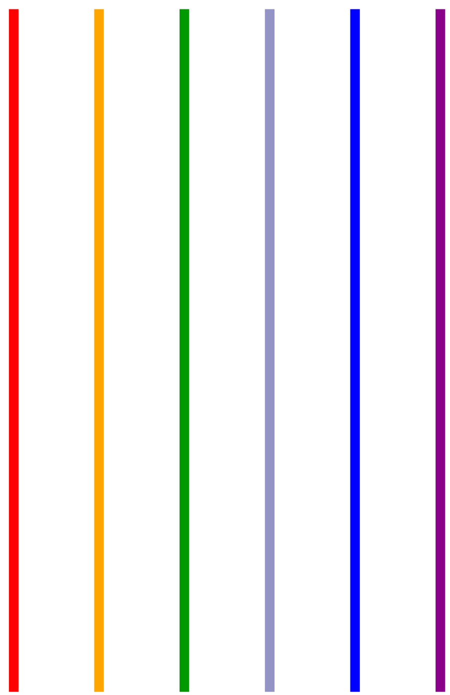
Pidstryhach Institute for Applied Problems
of Mechanics and Mathematics,
National Academy of Sciences of Ukraine,
Lviv, Ukraine

Stefan Felsner

TU Berlin,
Germany

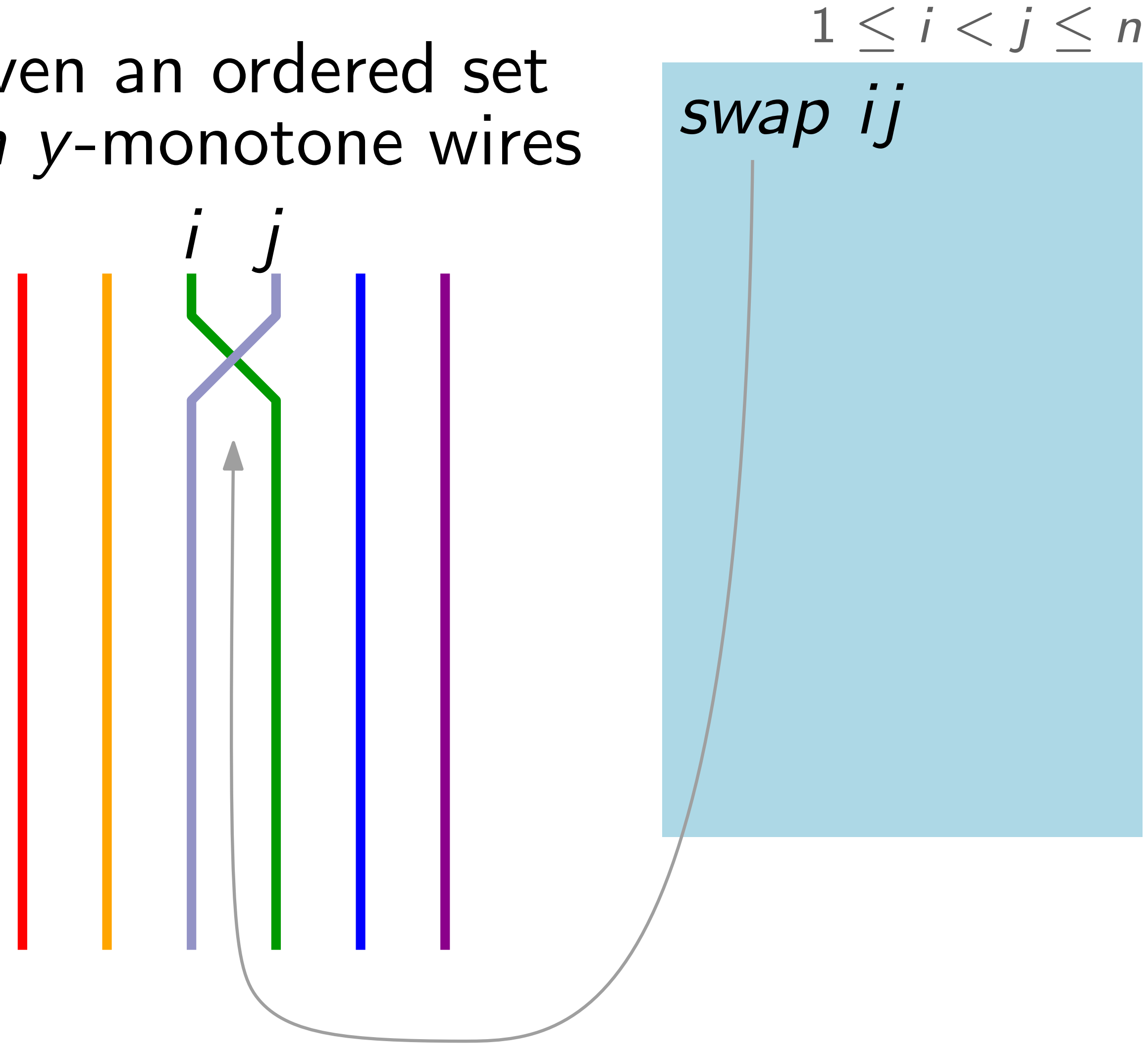
Introduction

Given an ordered set of n y-monotone wires



Introduction

Given an ordered set of n y-monotone wires



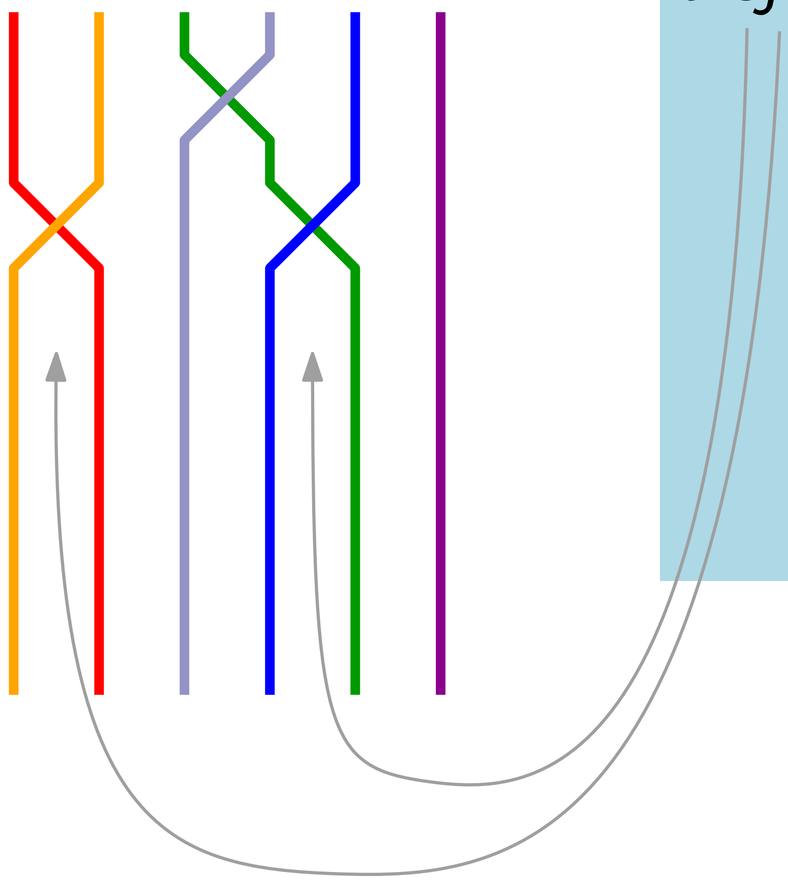
Introduction

Given an ordered set of n y-monotone wires

$1 \leq i < j \leq n$

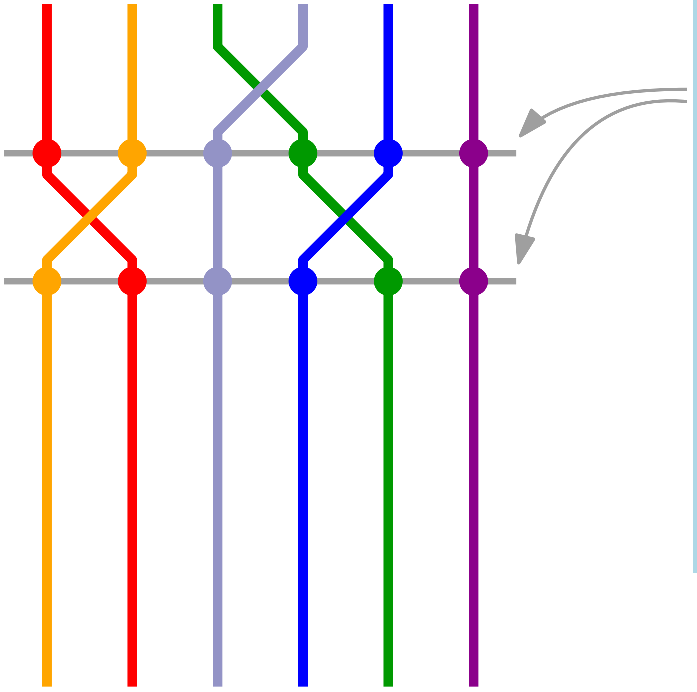
swap ij

disjoint swaps



Introduction

Given an ordered set of n y-monotone wires



$1 \leq i < j \leq n$

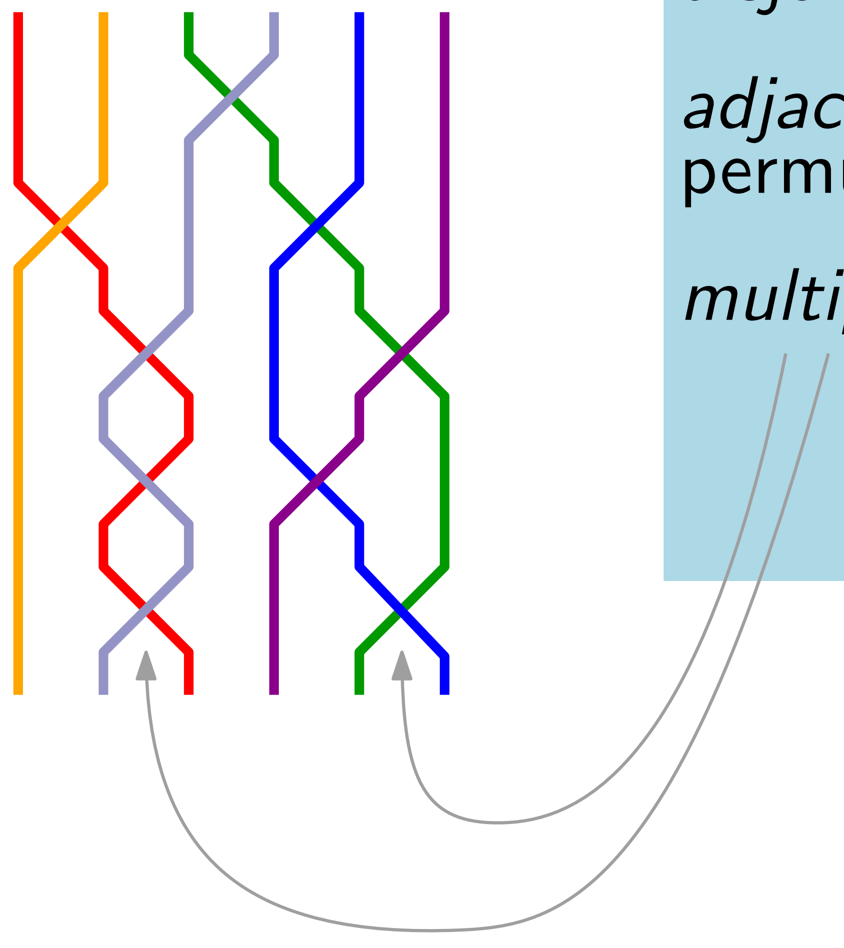
swap ij

disjoint swaps

adjacent permutations

Introduction

Given an ordered set
of n y-monotone wires



$1 \le i < j \le n$

swap ij

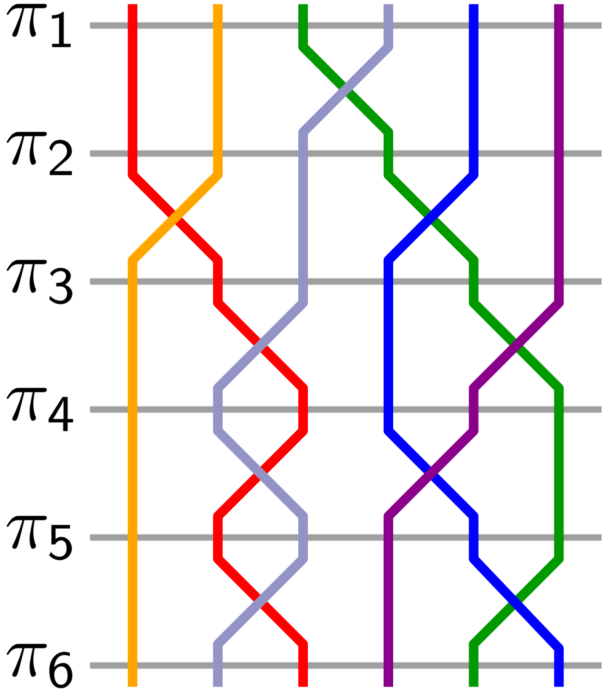
disjoint swaps

adjacent permutations

multiple swaps

Introduction

Given an ordered set of n y-monotone wires



$1 \leq i < j \leq n$

swap ij

disjoint swaps

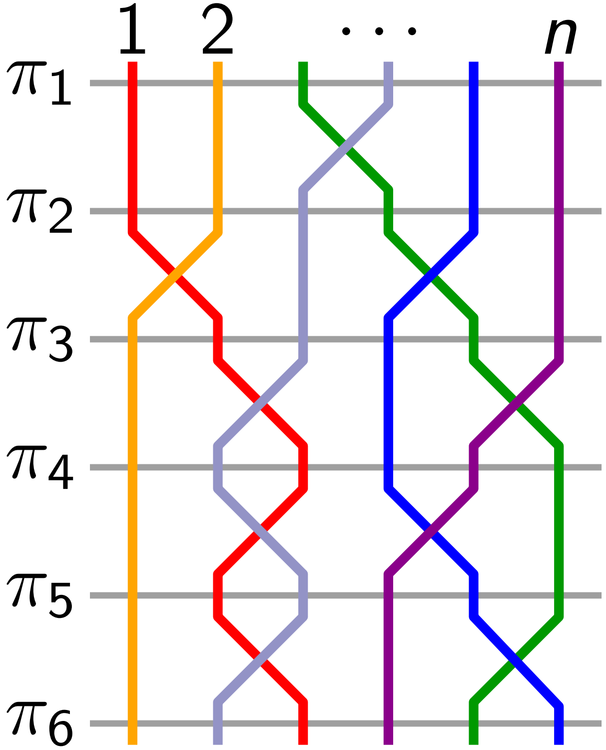
adjacent permutations

multiple swaps

tangle T of *height* $h(T)$

Introduction

Given an ordered set of n y-monotone wires



$1 \leq i < j \leq n$

swap ij

disjoint swaps

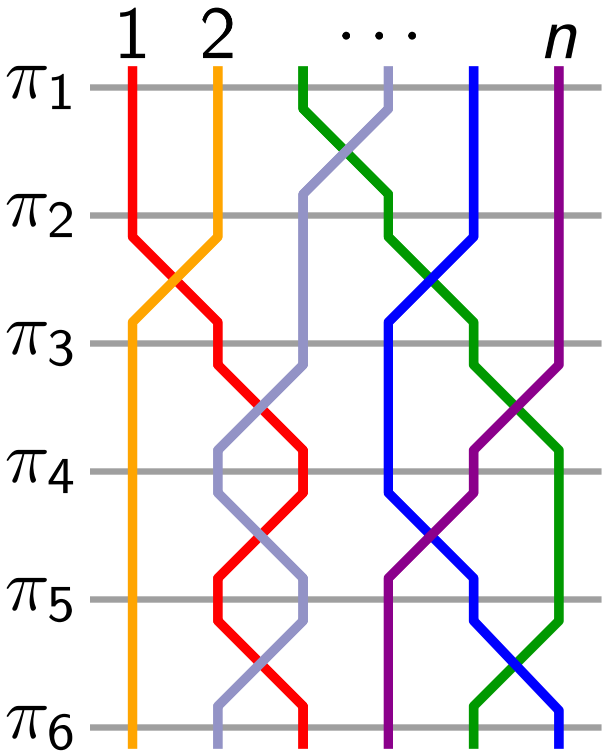
adjacent permutations

multiple swaps

tangle T of *height $h(T)$*

Introduction

Given an ordered set of n y-monotone wires



$1 \leq i < j \leq n$

swap ij

disjoint swaps

adjacent permutations

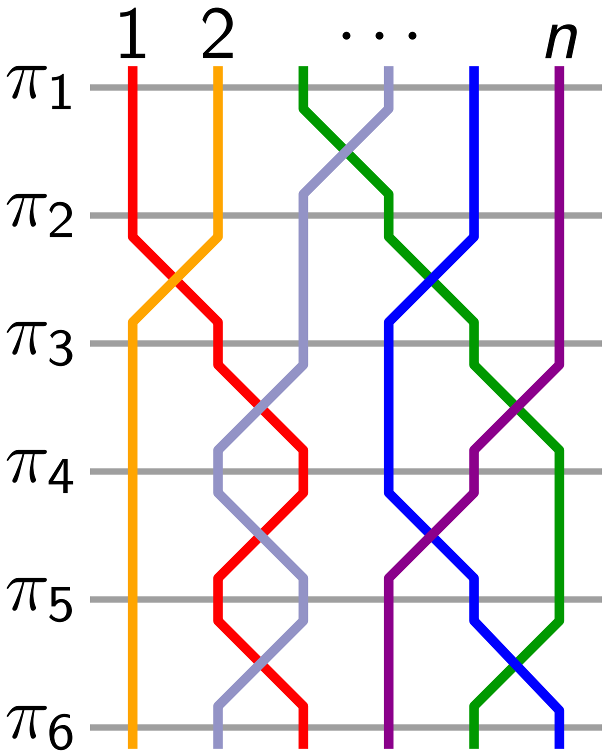
multiple swaps

tangle T of *height h(T)*

... and given a list L of swaps

Introduction

Given an ordered set of n y-monotone wires



$1 \leq i < j \leq n$

swap ij

disjoint swaps

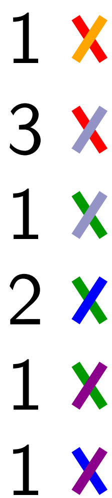
adjacent permutations

multiple swaps

tangle T of *height h(T)*

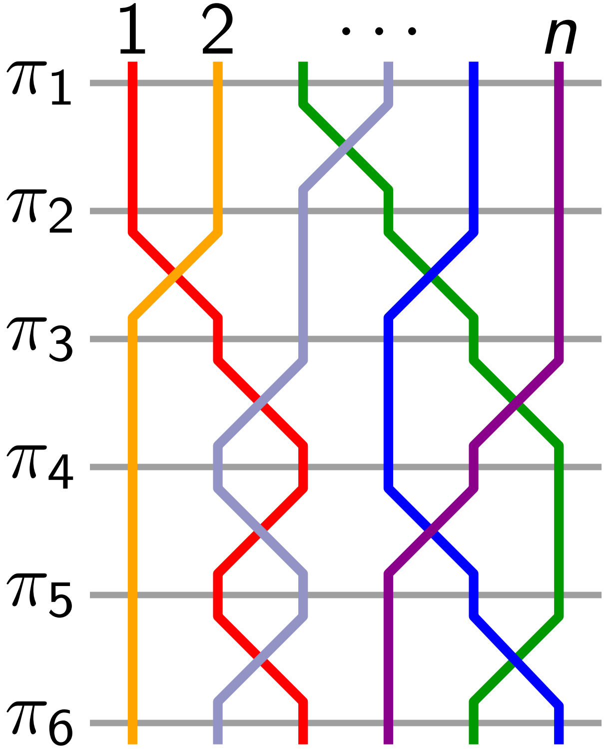
...and given a list L of swaps

as a multiset (ℓ_{ij})



Introduction

Given an ordered set of n y-monotone wires



$1 \leq i < j \leq n$

swap ij

disjoint swaps

adjacent permutations

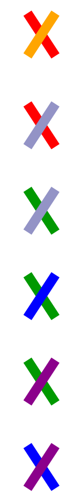
multiple swaps

tangle T of *height* $h(T)$

. . . and given a list L of swaps

as a multiset (ℓ_{ij})

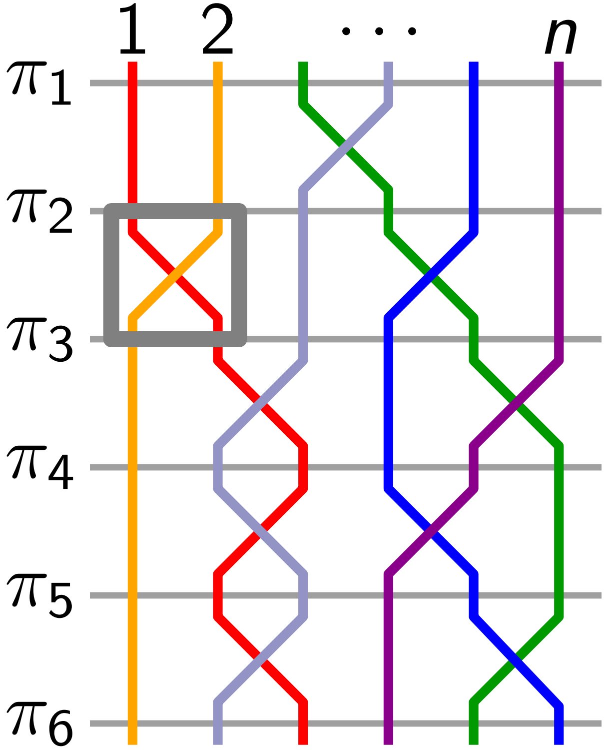
1
3
1
2
1
1



Tangle T *realizes* list L.

Introduction

Given an ordered set of n y-monotone wires



$1 \le i < j \le n$

swap ij

disjoint swaps

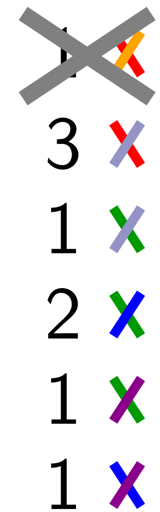
adjacent permutations

multiple swaps

tangle T of *height* $h(T)$

... and given a list L of swaps

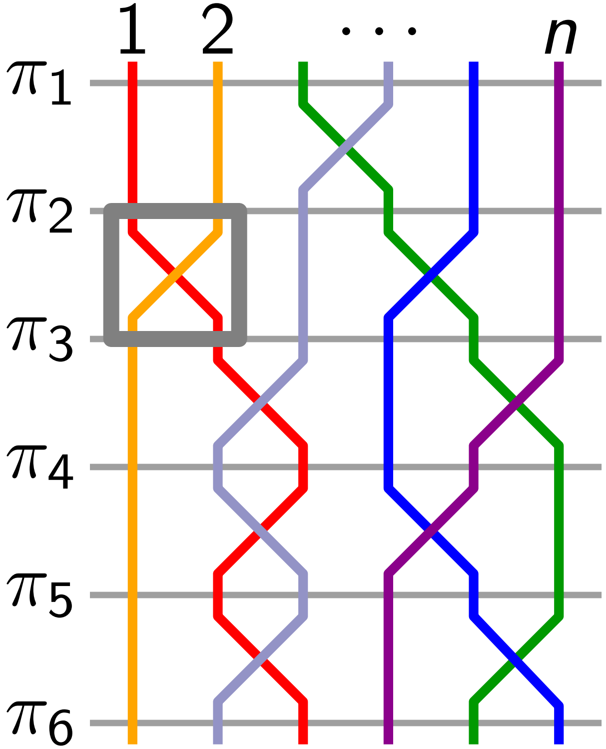
as a multiset (ℓ_{ij})



Tangle T *realizes* list L.

Introduction

Given an ordered set of n y-monotone wires



$1 \leq i < j \leq n$

swap ij

disjoint swaps

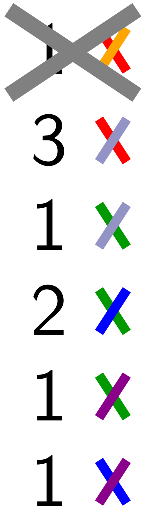
adjacent permutations

multiple swaps

tangle T of *height* $h(T)$

. . . and given a list L of swaps

as a multiset (ℓ_{ij})

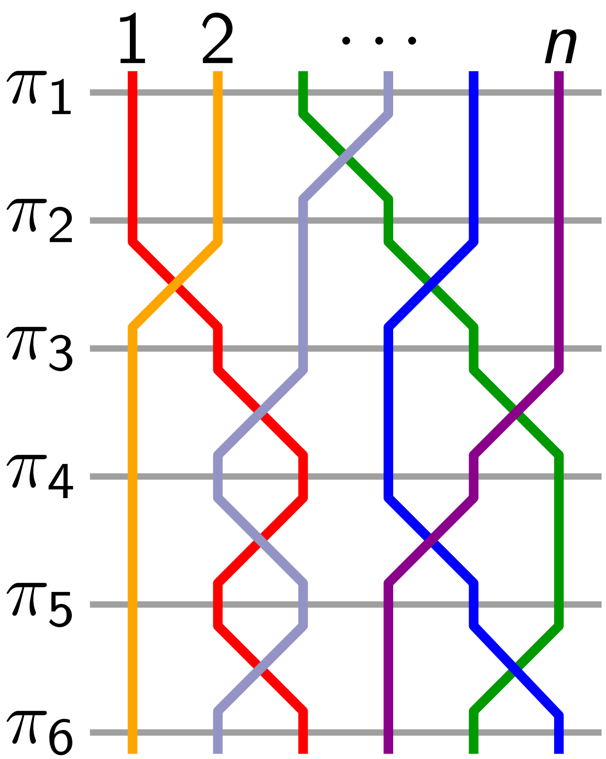


not *feasible*

Tangle T *realizes* list L.

Introduction

Given an ordered set of n y-monotone wires



$1 \leq i < j \leq n$

swap ij

disjoint swaps

adjacent permutations

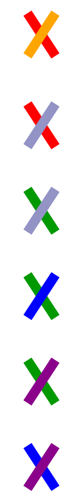
multiple swaps

tangle T of *height h(T)*

... and given a list L of swaps

as a multiset (ℓ_{ij})

1
3
1
2
1
1

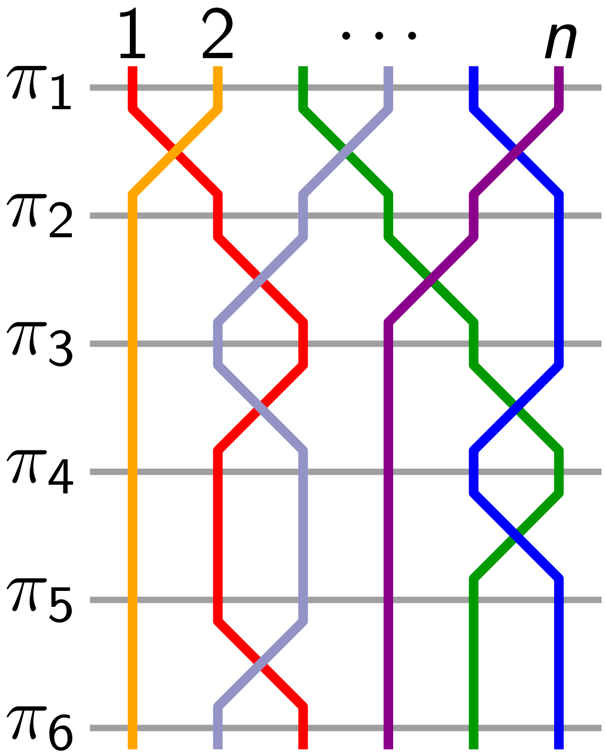


Tangle T *realizes* list L.

A list L of swaps is *feasible* if there exists a tangle that realizes L.
There may be multiple tangles realizing the same list of swaps.

Introduction

Given an ordered set of n y-monotone wires



$1 \le i < j \le n$

swap ij

disjoint swaps

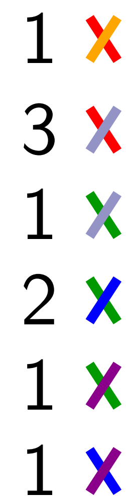
adjacent permutations

multiple swaps

tangle T of *height* $h(T)$

...and given a list L of swaps

as a multiset (ℓ_{ij})

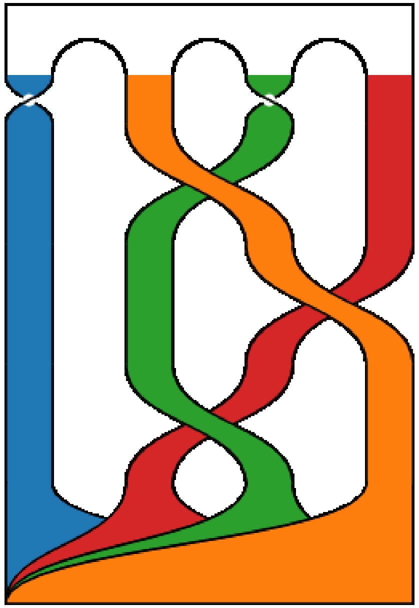


Tangle T *realizes* list L.

A list L of swaps is *feasible* if there exists a tangle that realizes L. There may be multiple tangles realizing the same list of swaps.

Related Work

- *Olszewski et al.*: Visualizing the template of a chaotic attractor. GD 2018

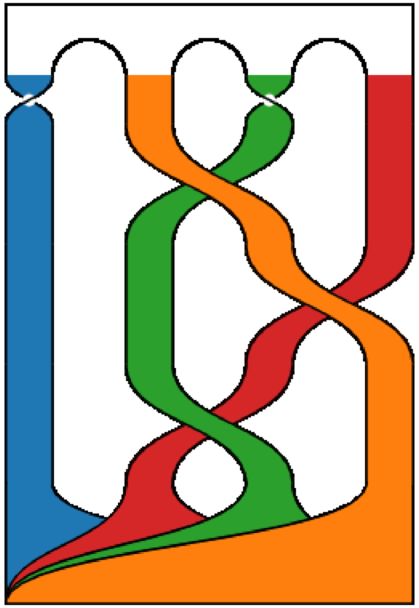


Related Work

list

⇕

- *Olszewski et al.*: Visualizing the template of a chaotic attractor. GD 2018



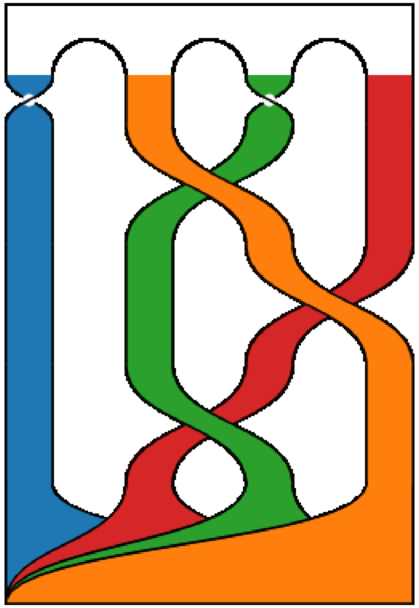
Related Work

list

⇕

- *Olszewski et al.*: Visualizing the template of a chaotic attractor. GD 2018

Exp.-time algorithm for finding optimal-height tangles



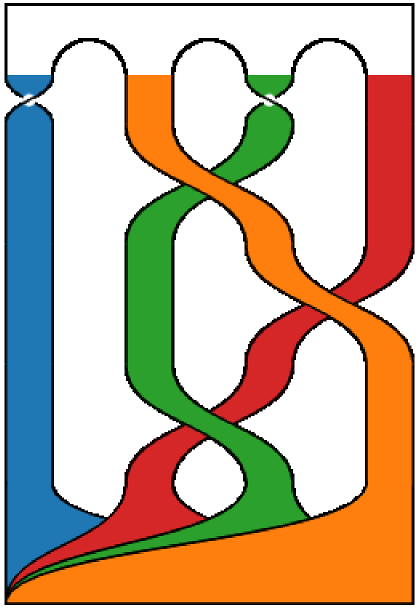
Related Work

list

⇕

- *Olszewski et al.*: Visualizing the template of a chaotic attractor. GD 2018

Exp.-time algorithm for finding optimal-height tangles	*Complexity* ?



Related Work

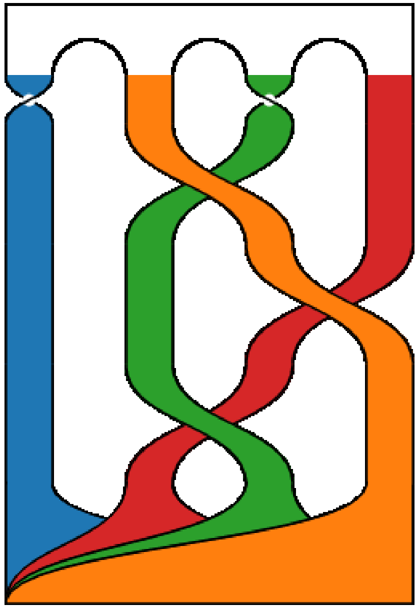
list

⇕

- *Olszewski et al.*: Visualizing the template of a chaotic attractor. GD 2018

Exp.-time algorithm for finding optimal-height tangles

Complexity ?



- *Sado and Igarashi*: A function for evaluating the computing time of a bubbling system. TCS 1987

Given: initial and final permutations

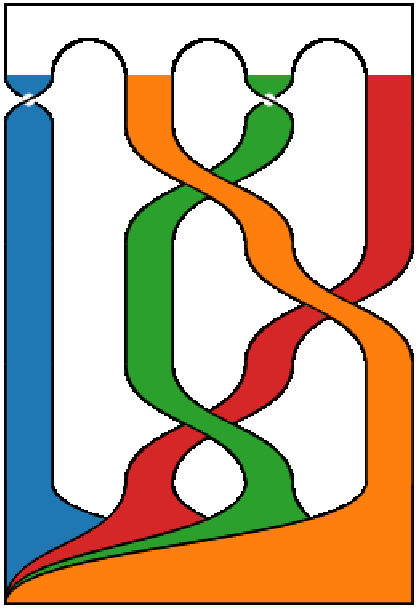
Related Work

list

⇕

- *Olszewski et al.*: Visualizing the template of a chaotic attractor. GD 2018

Exp.-time algorithm for finding optimal-height tangles | *Complexity* ?



- *Sado and Igarashi*: A function for evaluating the computing time of a bubbling system. TCS 1987

Given: initial and final permutations

- *Bereg et al.*: Drawing Permutations with Few Corners. GD 2013

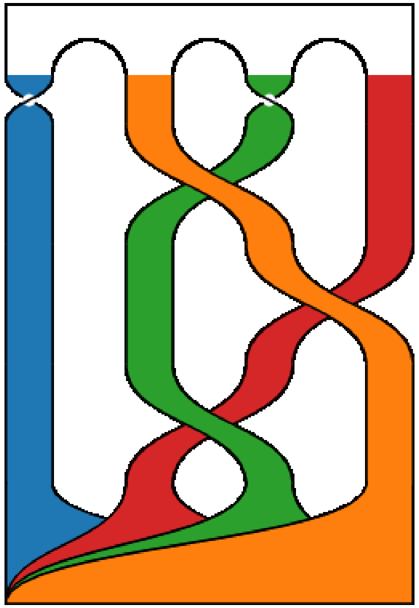
Objective: minimize the number of *bends*

Related Work

list
⇕

- *Olszewski et al.*: Visualizing the template of a chaotic attractor. GD 2018

Exp.-time algorithm for finding optimal-height tangles | *Complexity* **?**



- *Sado and Igarashi*: A function for evaluating the computing time of a bubbling system. TCS 1987

Given: initial and final permutations

- *Bereg et al.*: Drawing Permutations with Few Corners. GD 2013

Objective: minimize the number of *bends*

- *FKRWZ*: Computing optimal-height tangles faster. GD 2019

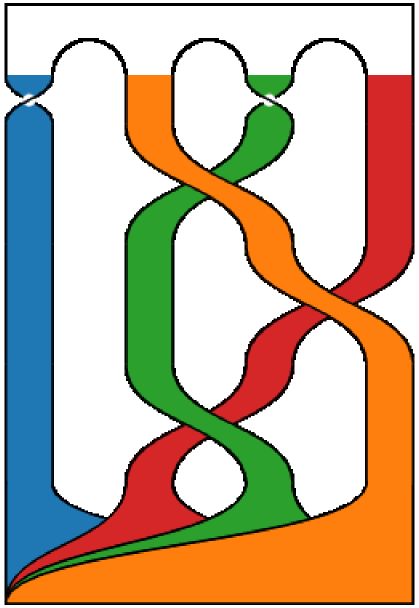
Related Work

- *Olszewski et al.*: Visualizing the template of a chaotic attractor. GD 2018

list ⇕ template

Exp.-time algorithm for finding optimal-height tangles

Complexity ?



- *Sado and Igarashi*: A function for evaluating the computing time of a bubbling system. TCS 1987

Given: initial and final permutations

- *Bereg et al.*: Drawing Permutations with Few Corners. GD 2013

Objective: minimize the number of *bends*

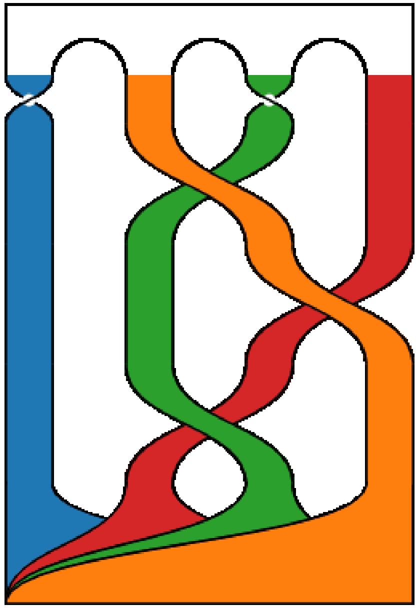
- *FKRWZ*: Computing optimal-height tangles faster. GD 2019

Faster exp.-time algorithm for finding optimal-height tangles

Related Work

list
⇕

- *Olszewski et al.*: Visualizing the template of a chaotic attractor. GD 2018



Exp.-time algorithm for finding optimal-height tangles | *Complexity* **?**

- *Sado and Igarashi*: A function for evaluating the computing time of a bubbling system. TCS 1987

Given: initial and final permutations

- *Bereg et al.*: Drawing Permutations with Few Corners. GD 2013

Objective: minimize the number of *bends*

- *FKRWZ*: Computing optimal-height tangles faster. GD 2019

Faster exp.-time algorithm for finding optimal-height tangles | Finding optimal-height tangles is NP-hard

Contribution

Theorem.
Deciding whether a given list of swaps is feasible is NP-hard.

Contribution

Theorem.
Deciding whether a given list of swaps is feasible is NP-hard.

Proof.
Reduction from POSITIVE NOT-ALL-EQUAL 3-SAT

Contribution

Theorem.
Deciding whether a given list of swaps is feasible is NP-hard.

Proof.
Reduction from POSITIVE NOT-ALL-EQUAL 3-SAT

Contribution

Theorem.
Deciding whether a given list of swaps is feasible is NP-hard.

Proof.
Reduction from Positive Not-All-Equal 3-SAT

$(F \vee F \vee F)$

Contribution

Theorem.
Deciding whether a given list of swaps is feasible is NP-hard.

Proof.
Reduction from POSITIVE NOT-ALL-EQUAL 3-SAT

$(F \vee F \vee F)$

Contribution

Theorem.
Deciding whether a given list of swaps is feasible is NP-hard.

Proof.
Reduction from Positive Not-All-Equal 3-SAT

$(F \vee F \vee F)$

$(T \vee T \vee T)$

Contribution

Theorem.
Deciding whether a given list of swaps is feasible is NP-hard.

Proof.
Reduction from POSITIVE NOT-ALL-EQUAL 3-SAT

$(F \vee F \vee F)$

$(T \vee T \vee T)$

Contribution

Theorem.
Deciding whether a given list of swaps is feasible is NP-hard.

Proof.
Reduction from POSITIVE NOT-ALL-EQUAL 3-SAT

~~$(F \vee F \vee F)$~~

~~$(T \vee T \vee T)$~~

~~negative literals~~

Contribution

Theorem.
Deciding whether a given list of swaps is feasible is NP-hard.

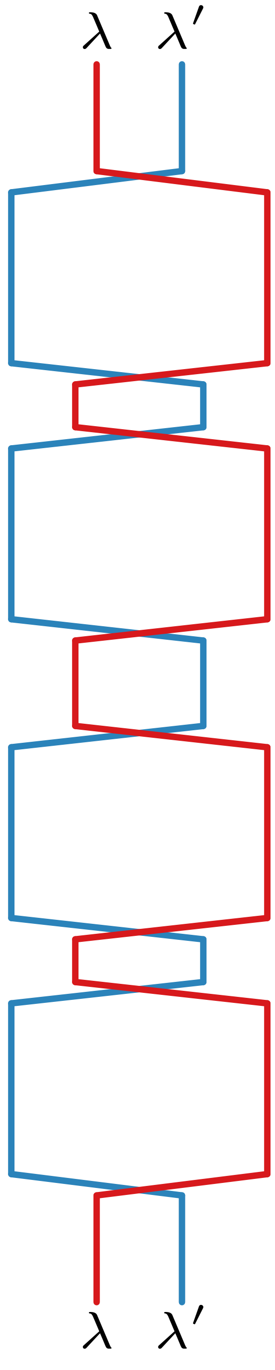
Proof.
Reduction from POSITIVE NOT-ALL-EQUAL 3-SAT

Idea

- Two wires build 4 *loops* that we consider

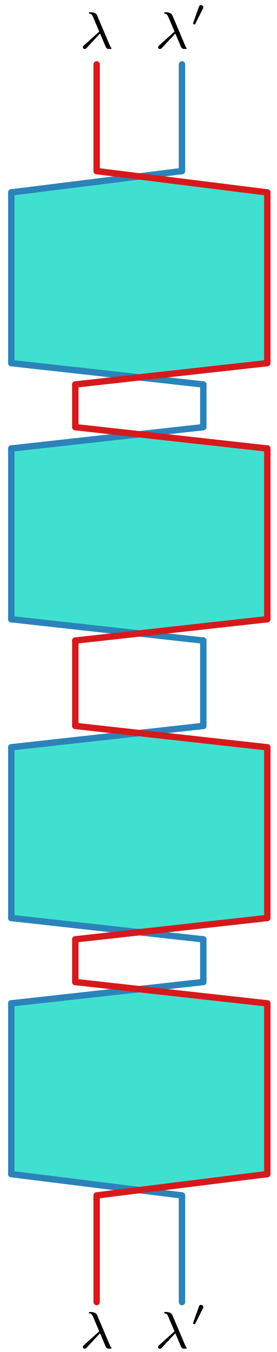
Idea

- Two wires build 4 *loops* that we consider



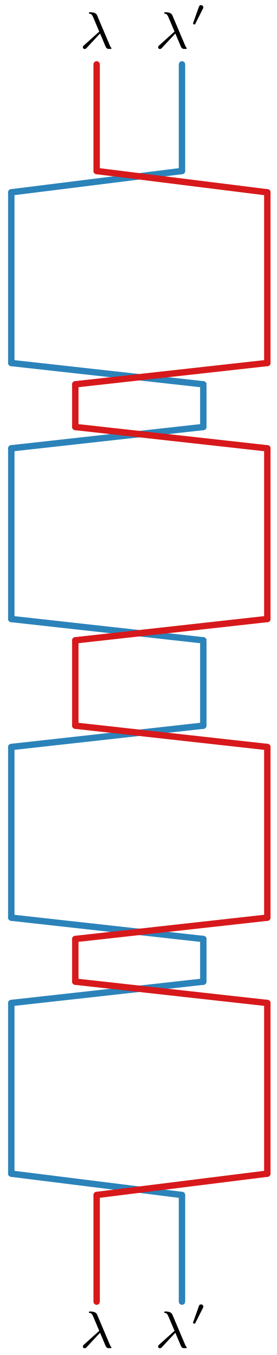
Idea

- Two wires build 4 *loops* that we consider



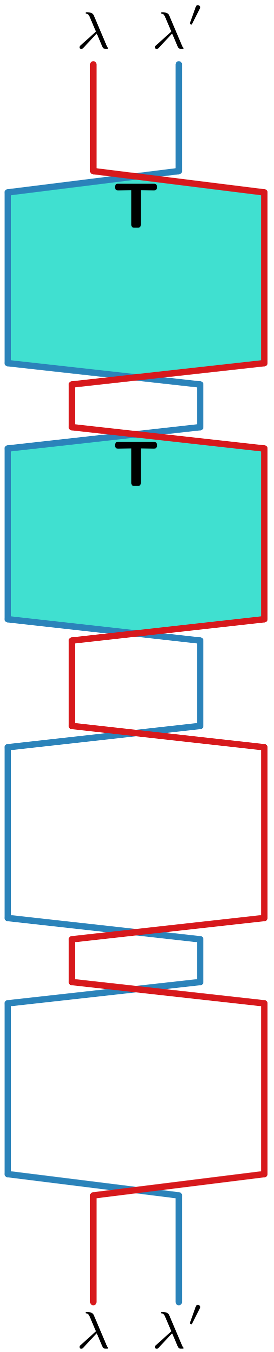
Idea

- Two wires build 4 *loops* that we consider
- Two loops represent *true*,



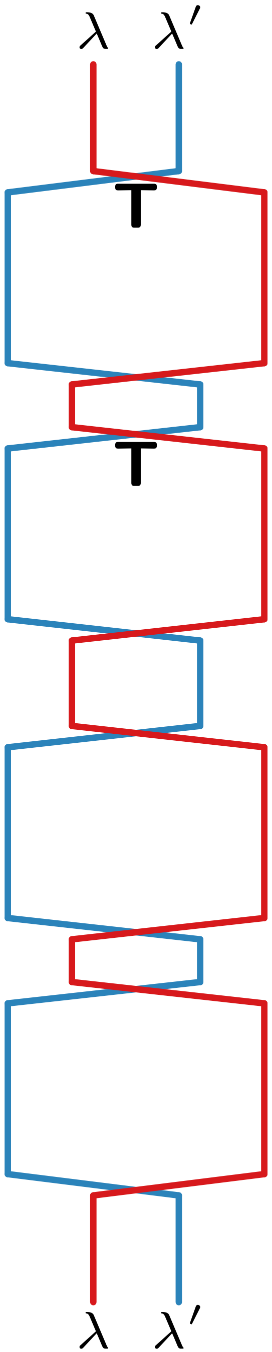
Idea

- Two wires build 4 *loops* that we consider
- Two loops represent *true*,



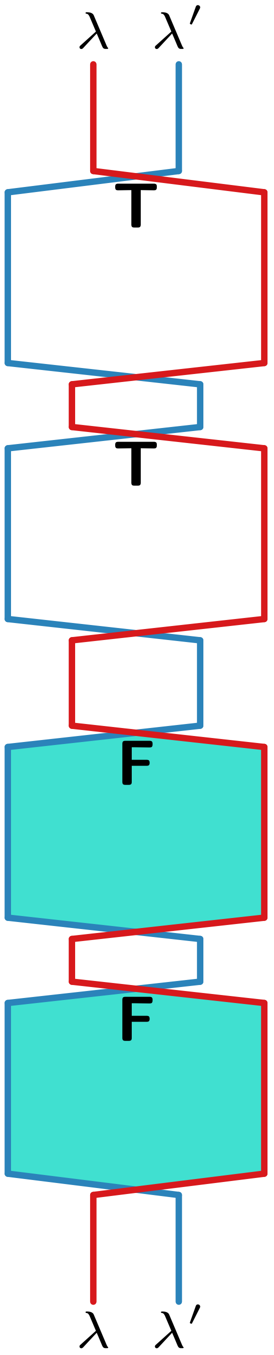
Idea

- Two wires build 4 *loops* that we consider
- Two loops represent *true*,
  the other two *false*



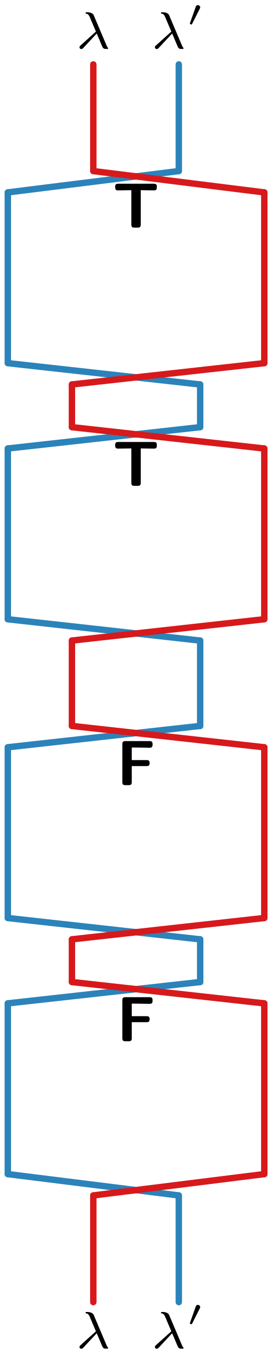
Idea

- Two wires build 4 *loops* that we consider
- Two loops represent *true*, the other two *false*



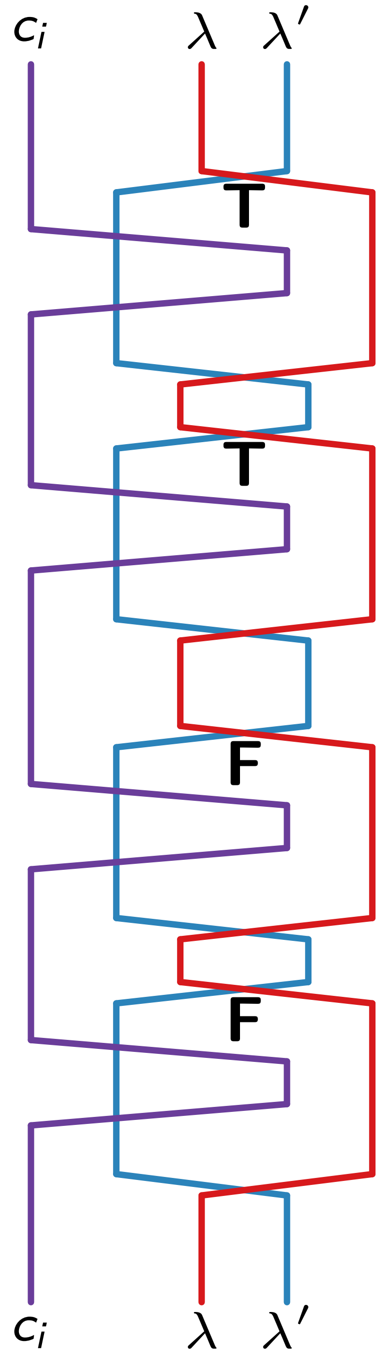
Idea

- Two wires build 4 *loops* that we consider
- Two loops represent *true*, the other two *false*
- For each clause, there is a wire with an *arm* in each of the 4 loops.



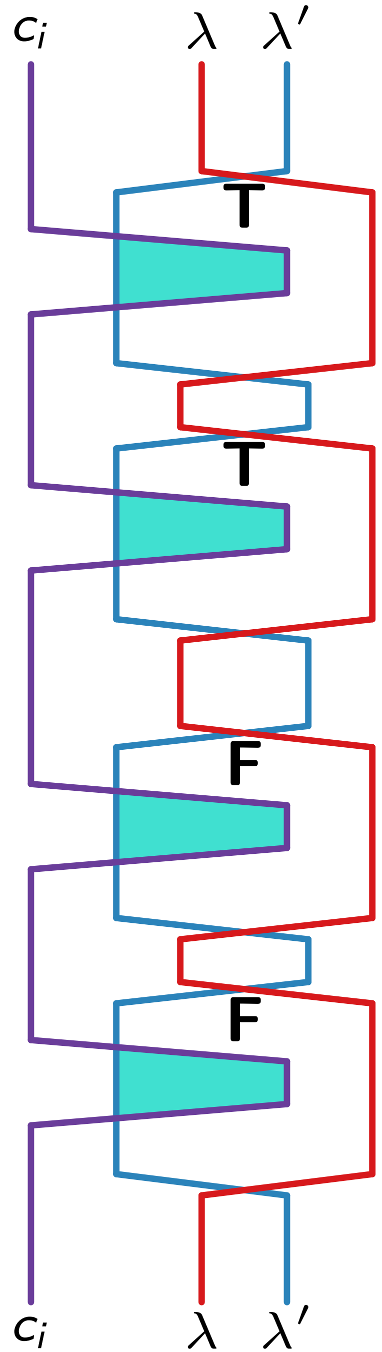
Idea

- Two wires build 4 *loops* that we consider
- Two loops represent *true*, the other two *false*
- For each clause, there is a wire with an *arm* in each of the 4 loops.



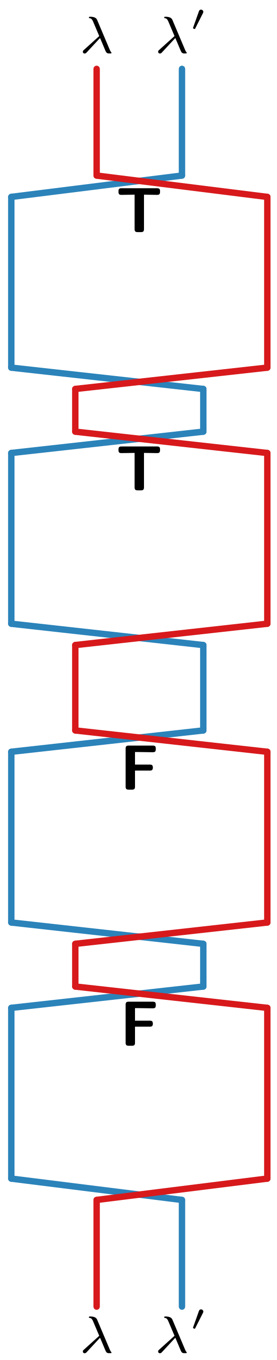
Idea

- Two wires build 4 *loops* that we consider
- Two loops represent *true*, the other two *false*
- For each clause, there is a wire with an *arm* in each of the 4 loops.



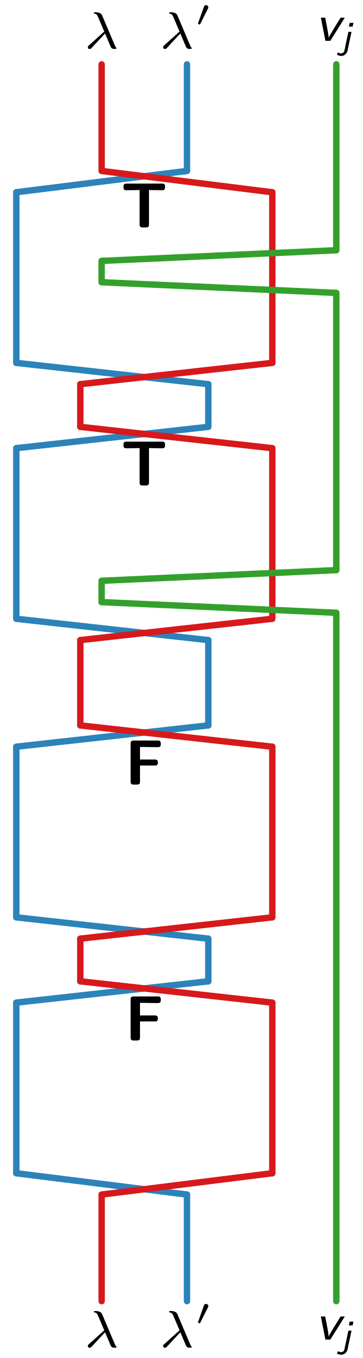
Idea

- Two wires build 4 *loops* that we consider
- Two loops represent *true*, the other two *false*
- For each clause, there is a wire with an *arm* in each of the 4 loops.
- For each variable, there is a wire entering either both *true* or both *false* loops.



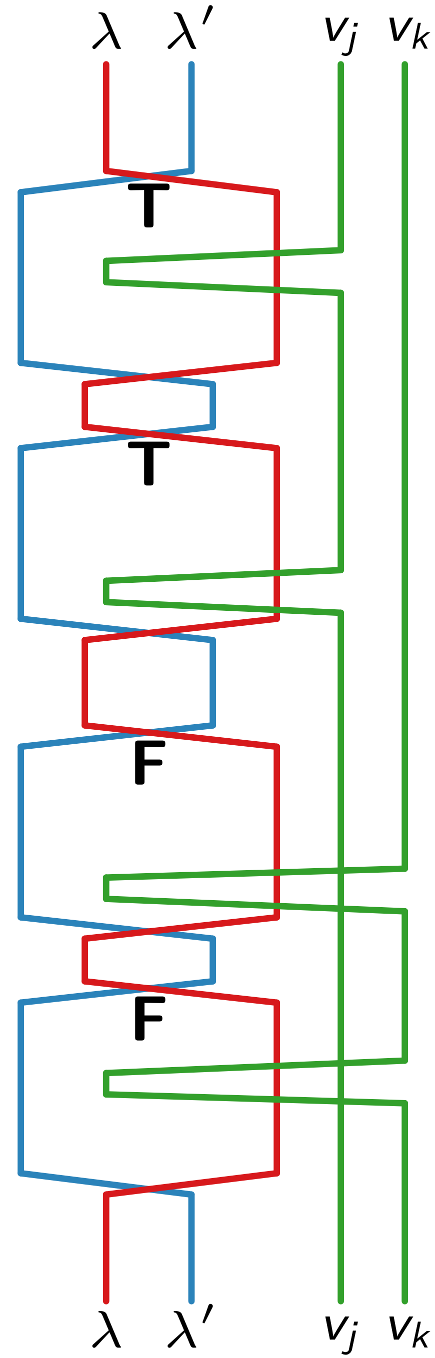
Idea

- Two wires build 4 *loops* that we consider
- Two loops represent *true*, the other two *false*
- For each clause, there is a wire with an *arm* in each of the 4 loops.
- For each variable, there is a wire entering either both *true* or both *false* loops.



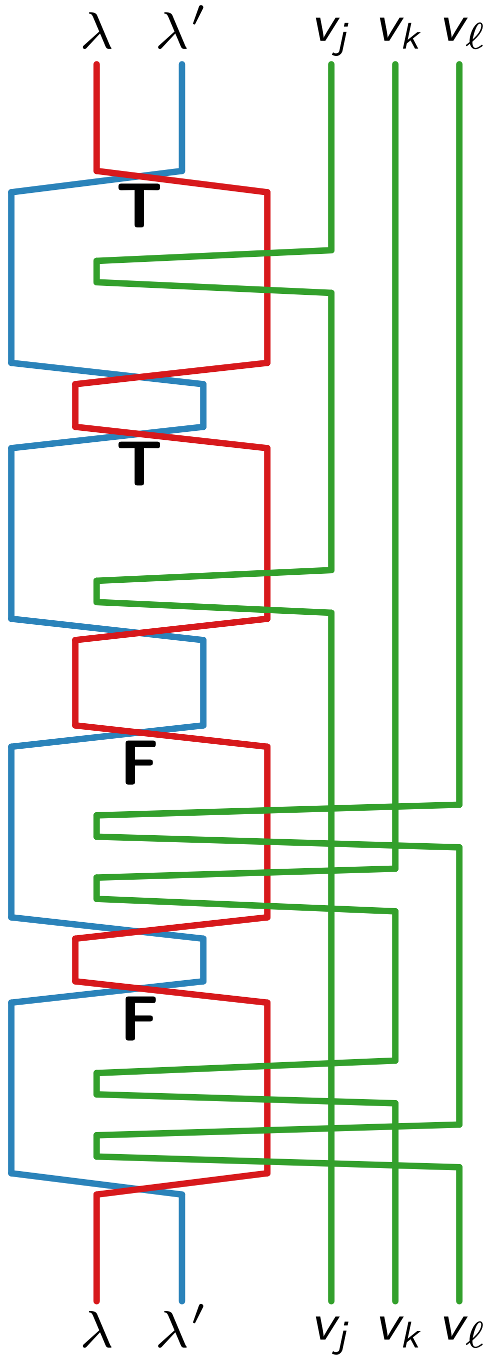
Idea

- Two wires build 4 *loops* that we consider
- Two loops represent *true*, the other two *false*
- For each clause, there is a wire with an *arm* in each of the 4 loops.
- For each variable, there is a wire entering either both *true* or both *false* loops.



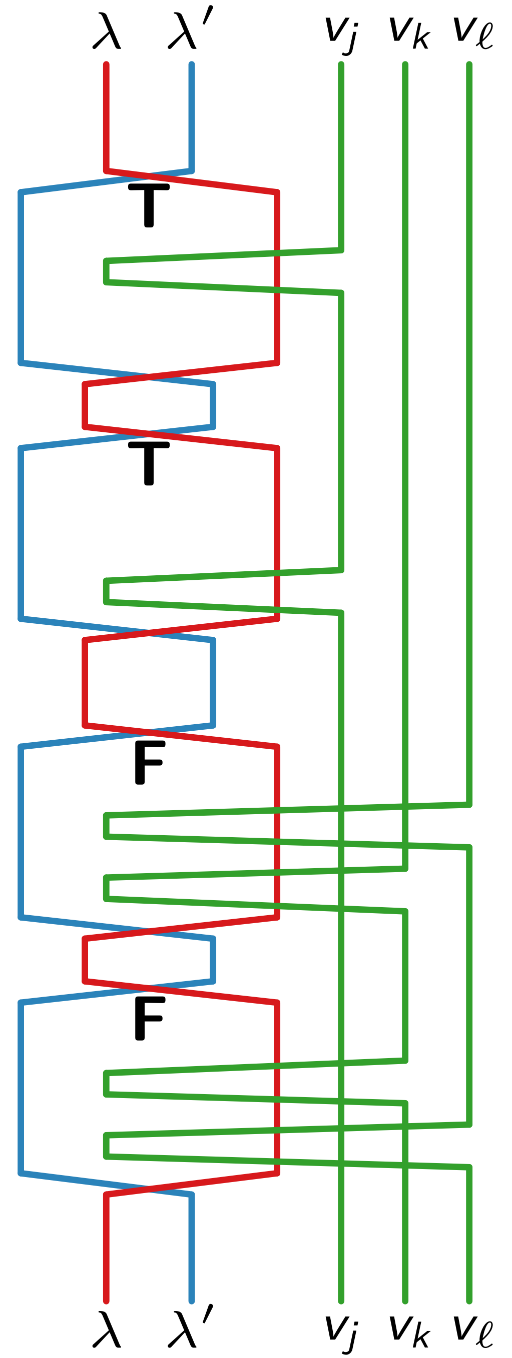
Idea

- Two wires build 4 *loops* that we consider
- Two loops represent *true*,
  the other two *false*
- For each clause, there is a wire with
  an *arm* in each of the 4 loops.
- For each variable, there is a wire entering
  either both *true* or both *false* loops.



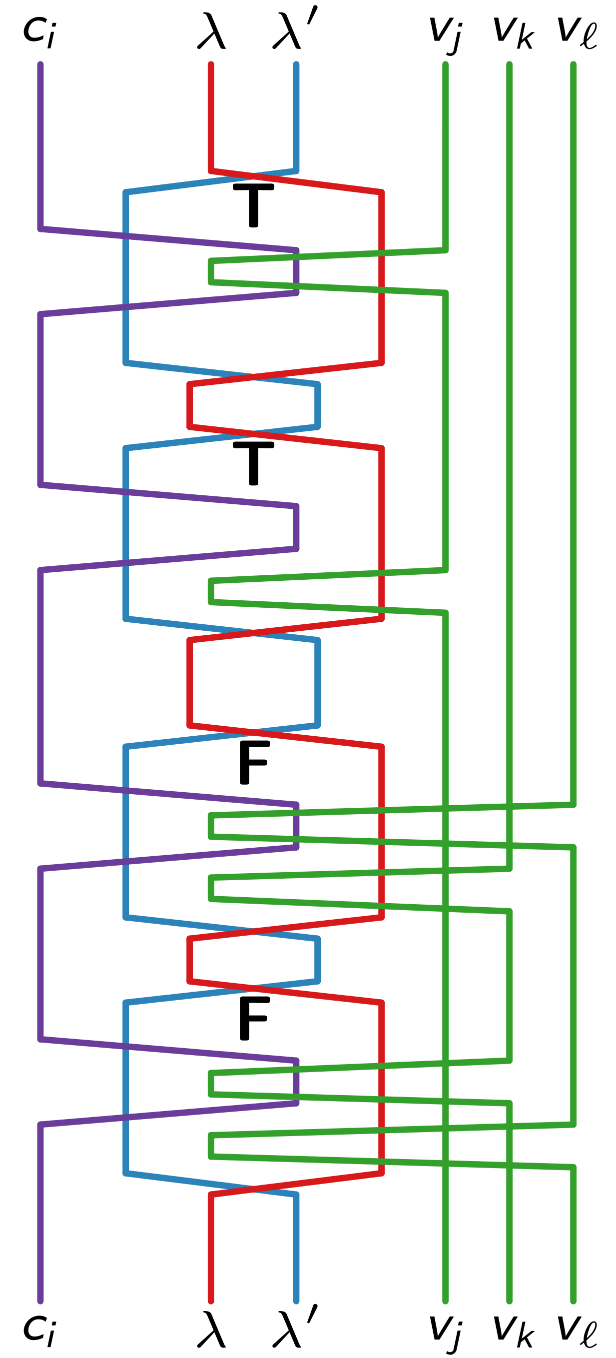
Idea

- Two wires build 4 *loops* that we consider
- Two loops represent *true*, the other two *false*
- For each clause, there is a wire with an *arm* in each of the 4 loops.
- For each variable, there is a wire entering either both *true* or both *false* loops.
- Each clause wire meets precisely its three corresponding variable wires – each one in a different loop.



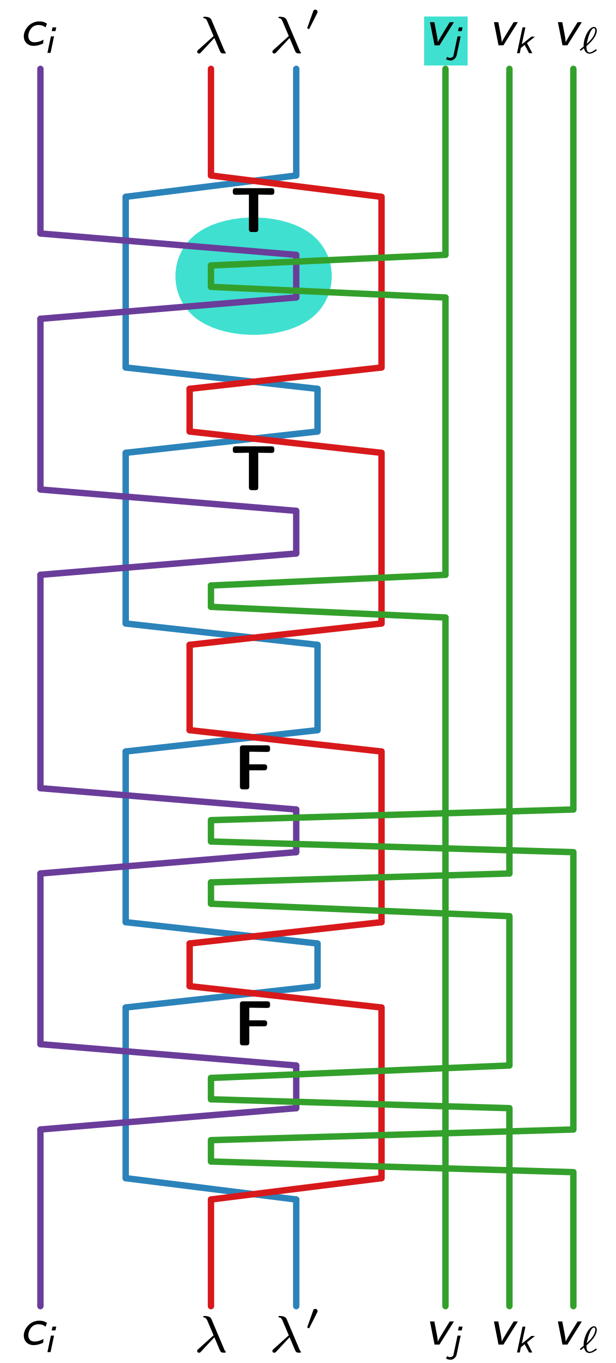
Idea

- Two wires build 4 *loops* that we consider
- Two loops represent *true*, the other two *false*
- For each clause, there is a wire with an *arm* in each of the 4 loops.
- For each variable, there is a wire entering either both *true* or both *false* loops.
- Each clause wire meets precisely its three corresponding variable wires – each one in a different loop.



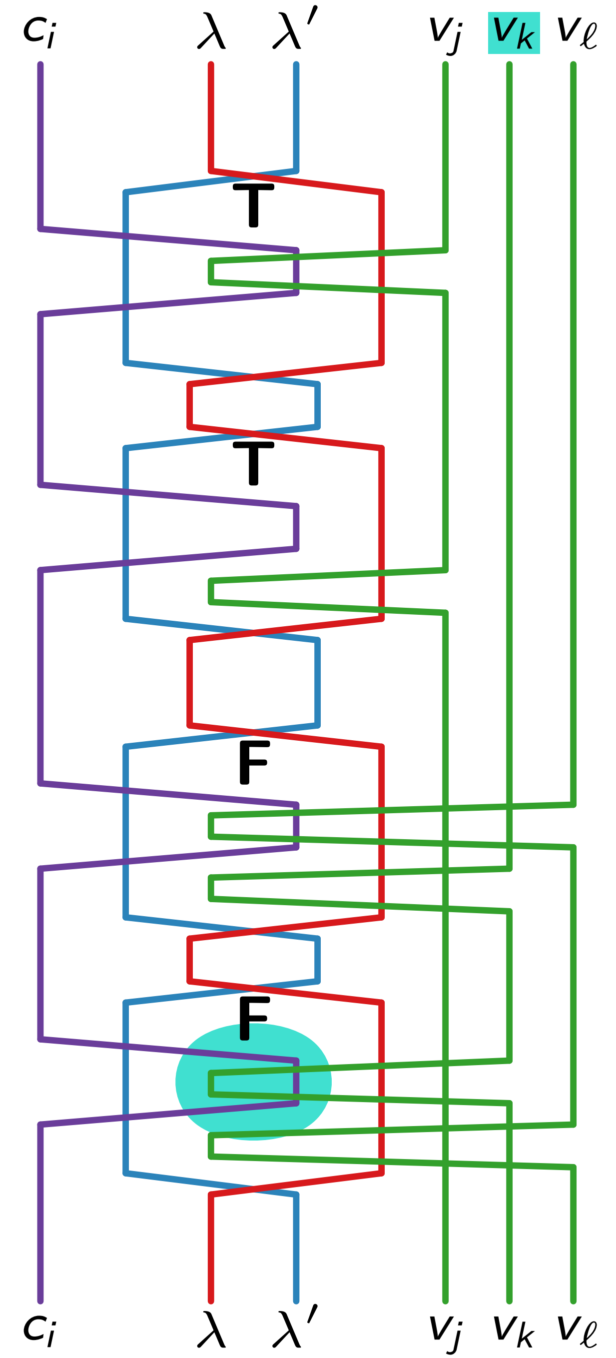
Idea

- Two wires build 4 *loops* that we consider
- Two loops represent *true*, the other two *false*
- For each clause, there is a wire with an *arm* in each of the 4 loops.
- For each variable, there is a wire entering either both *true* or both *false* loops.
- Each clause wire meets precisely its three corresponding variable wires – each one in a different loop.



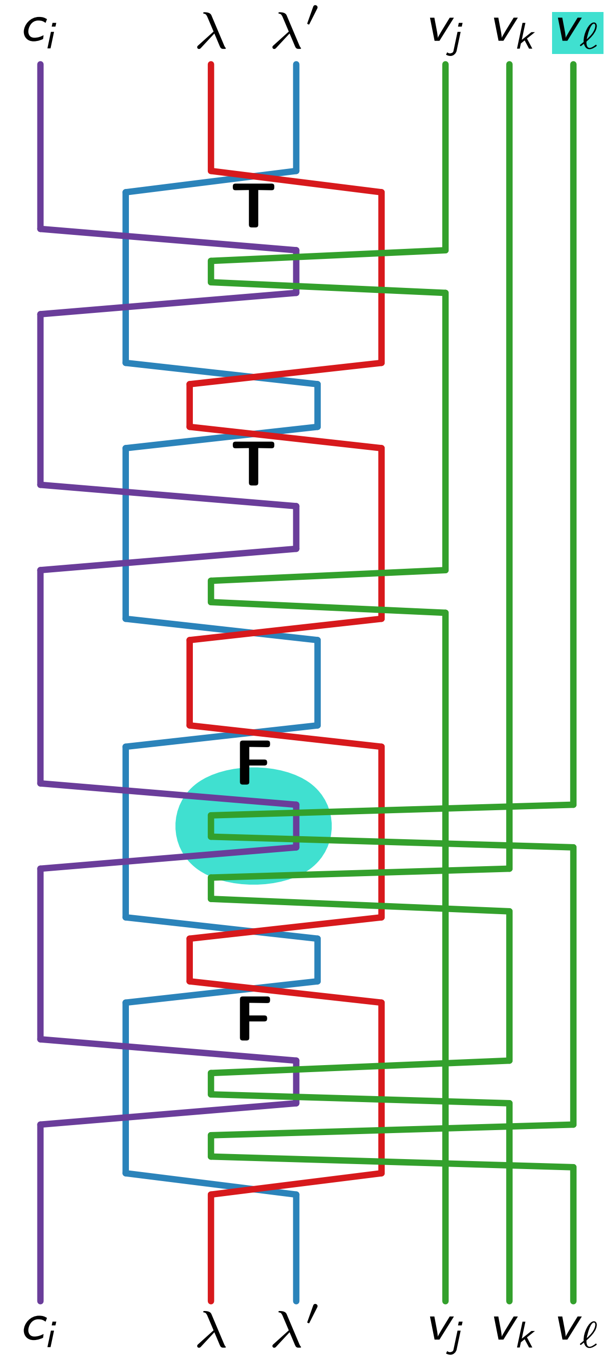
Idea

- Two wires build 4 *loops* that we consider
- Two loops represent *true*,
  the other two *false*
- For each clause, there is a wire with an *arm* in each of the 4 loops.
- For each variable, there is a wire entering either both *true* or both *false* loops.
- Each clause wire meets precisely its three corresponding variable wires – each one in a different loop.



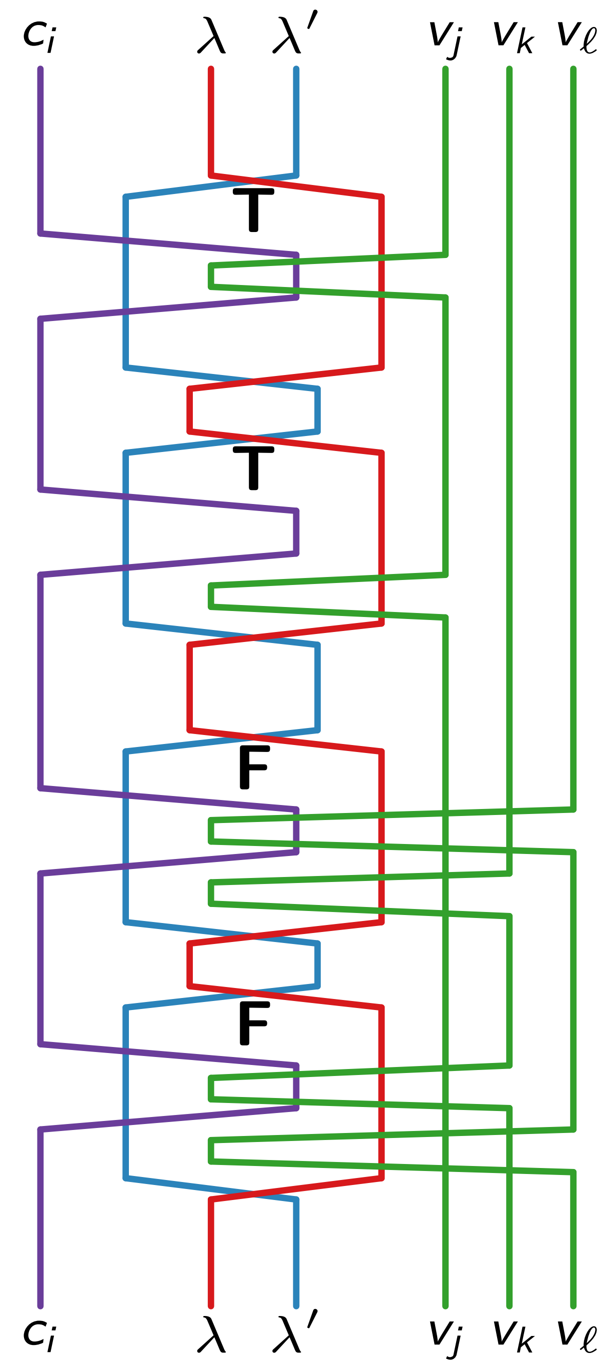
Idea

- Two wires build 4 *loops* that we consider
- Two loops represent *true*, the other two *false*
- For each clause, there is a wire with an *arm* in each of the 4 loops.
- For each variable, there is a wire entering either both *true* or both *false* loops.
- Each clause wire meets precisely its three corresponding variable wires – each one in a different loop.

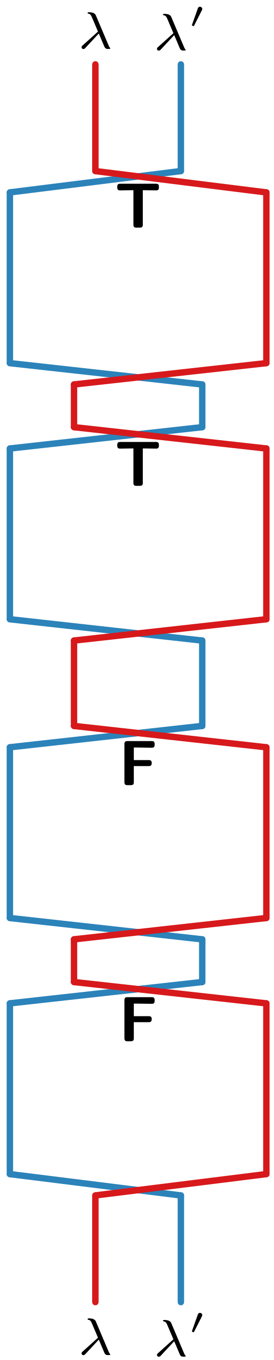


Idea

- Two wires build 4 *loops* that we consider
- Two loops represent *true*, the other two *false*
- For each clause, there is a wire with an *arm* in each of the 4 loops.
- For each variable, there is a wire entering either both *true* or both *false* loops.
- Each clause wire meets precisely its three corresponding variable wires – each one in a different loop.
- Only 2 *true* loops and 2 *false* loops $\Rightarrow$ clause wires meet all their variable wires iff $\textsc{Positive Not-All-Equal}$ 3-SAT formula satisfiable

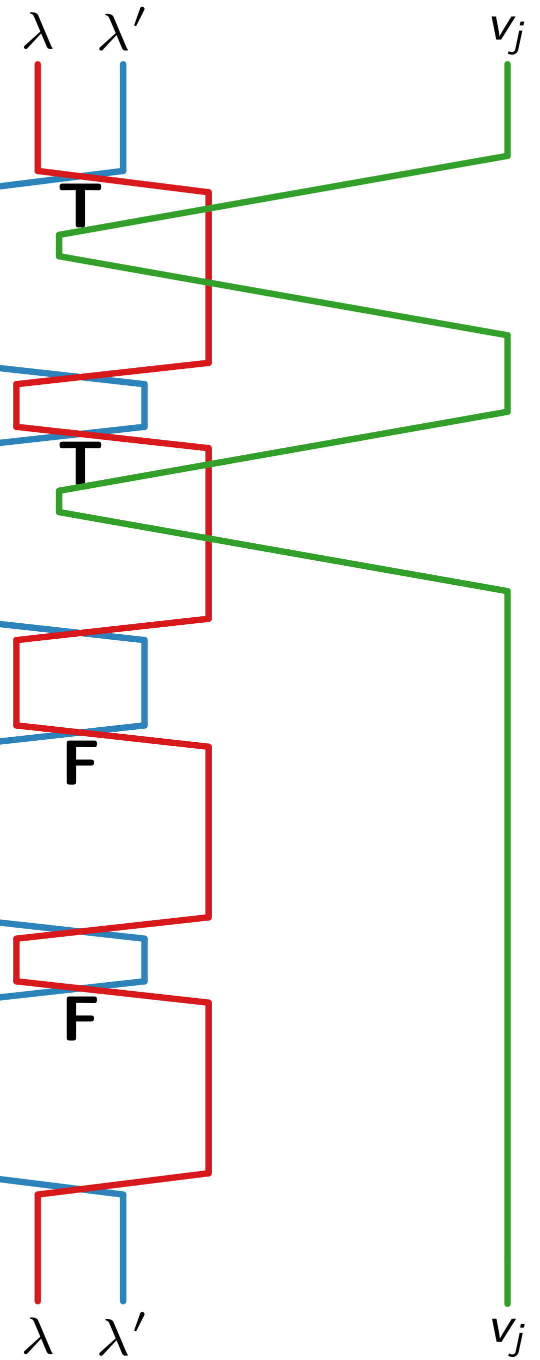


Variable Gadget



λ, λ': central loop structure

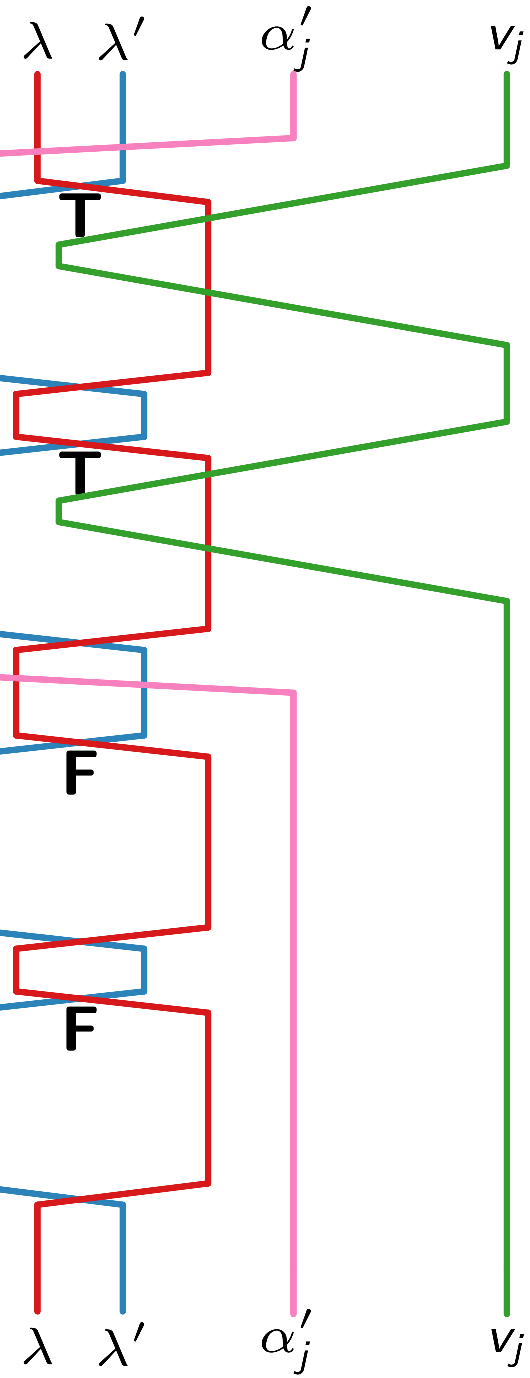
Variable Gadget



λ, λ': central loop structure

v_j: variable wire of j-th variable

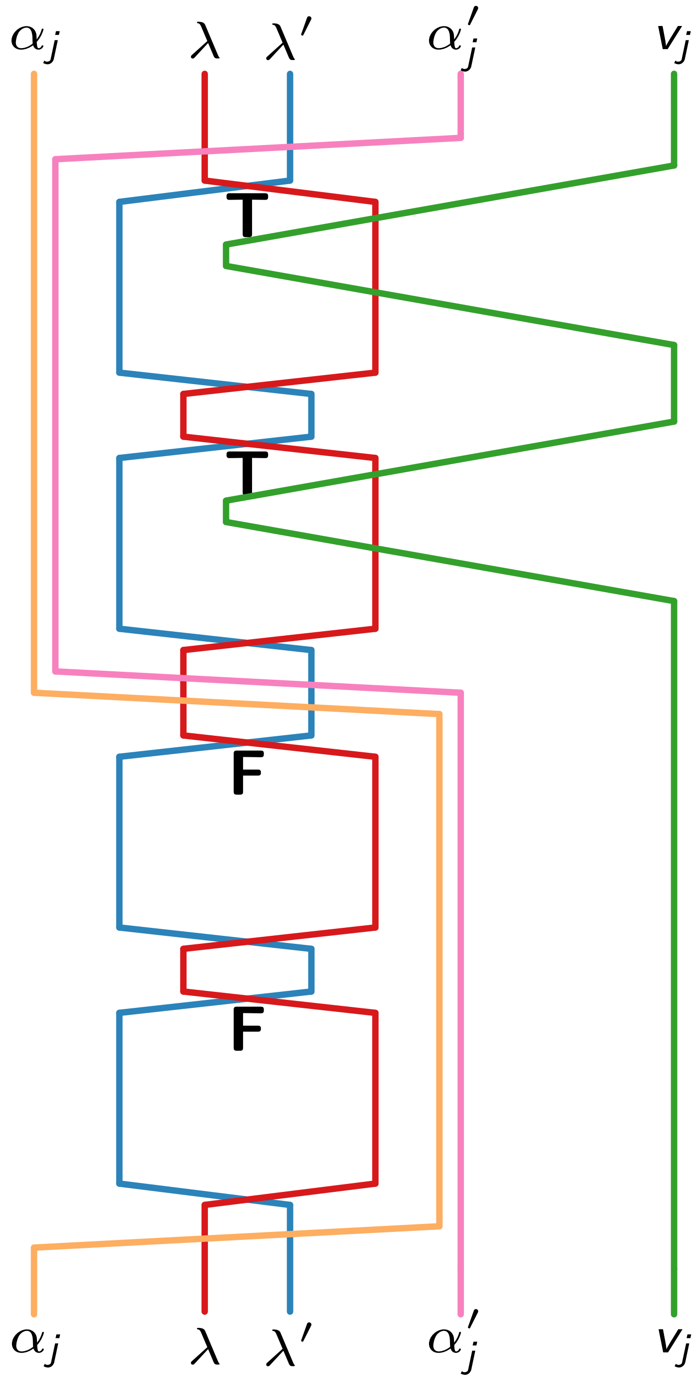
Variable Gadget



λ, λ': central loop structure

v_j: variable wire of j-th variable

Variable Gadget

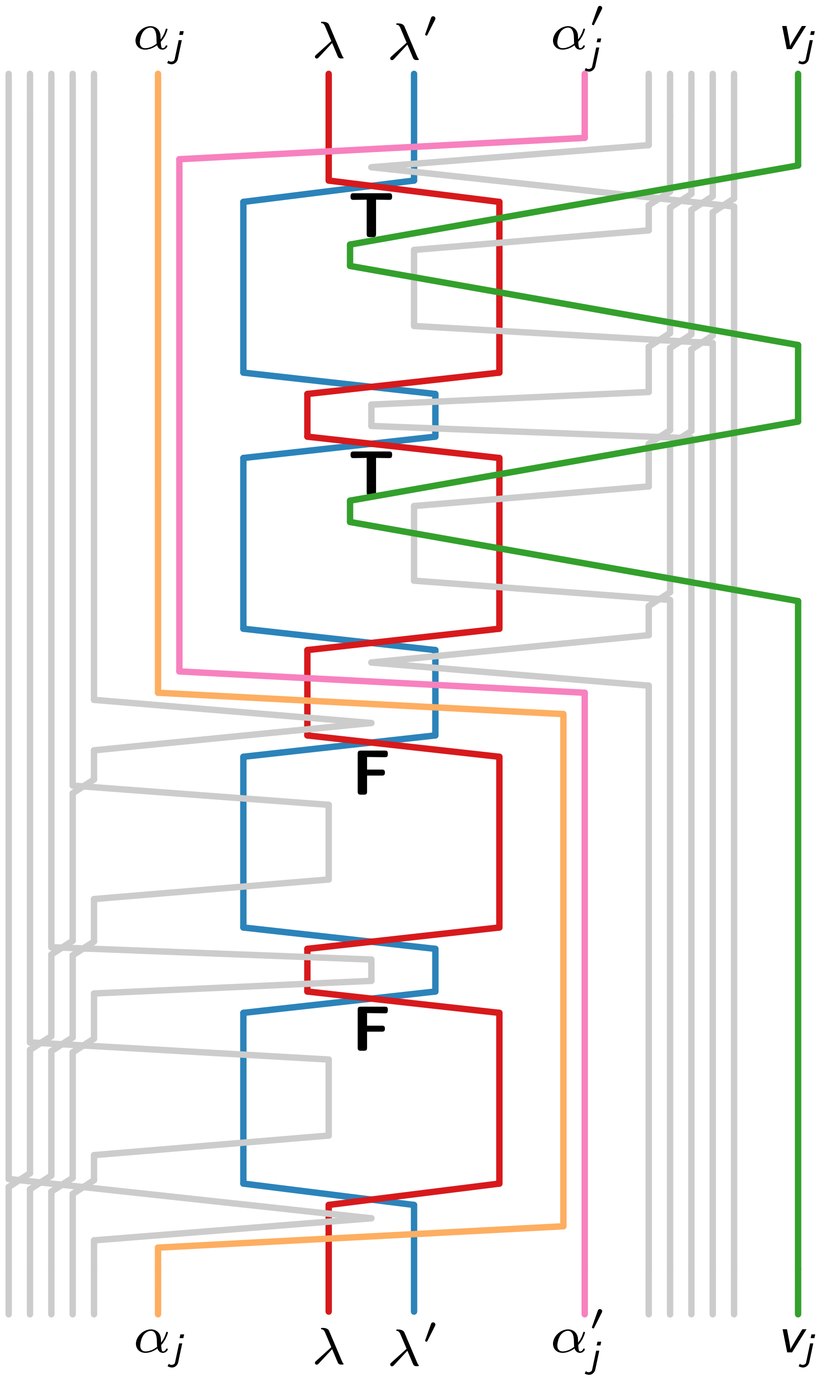


λ, λ': central loop structure

v_j: variable wire of j-th variable

α_j, α'_j: make v_j appear only in *true* or in *false* loops

Variable Gadget

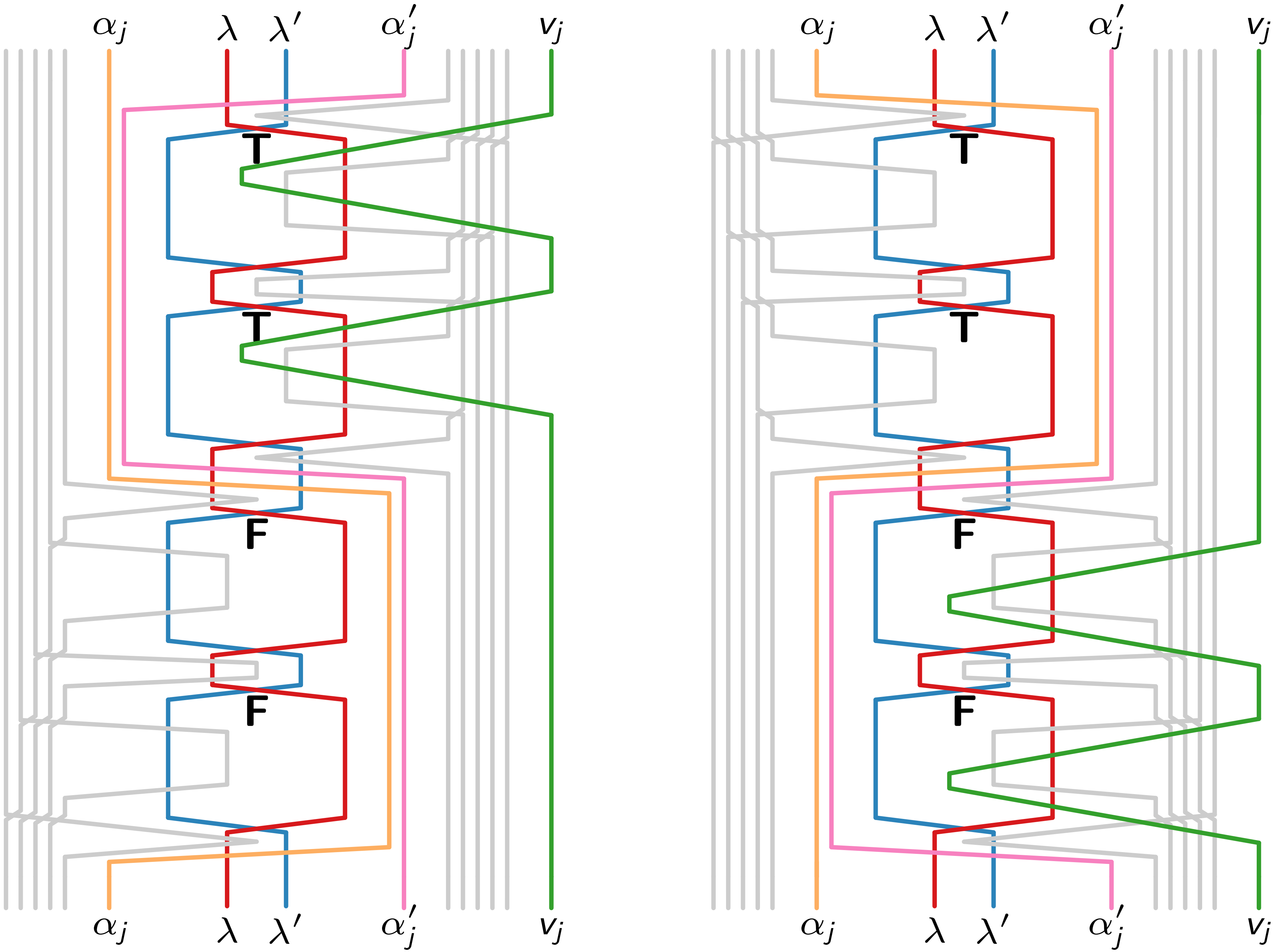


λ, λ': central loop structure

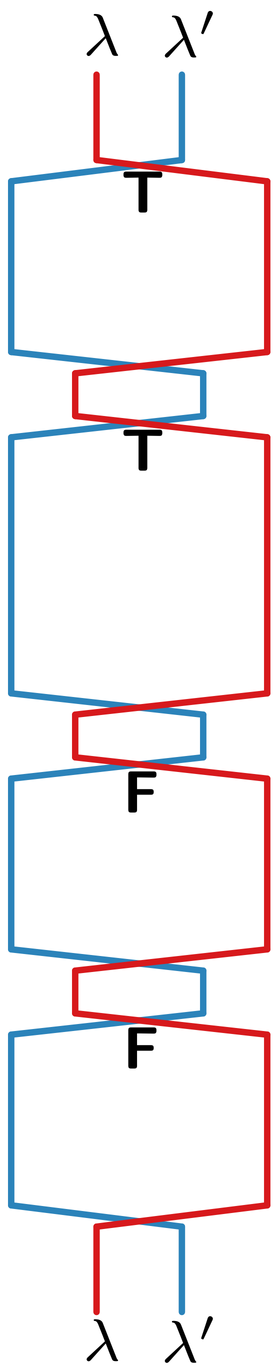
v_j: variable wire of j-th variable

α_j, α'_j: make v_j appear only in *true* or in *false* loops

Variable Gadget

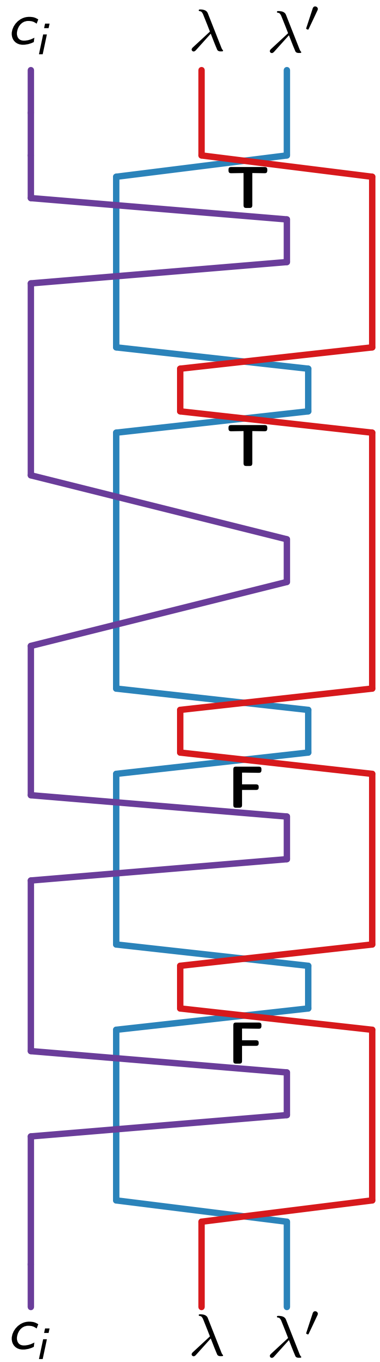


Clause Gadget



λ, λ': central loop structure

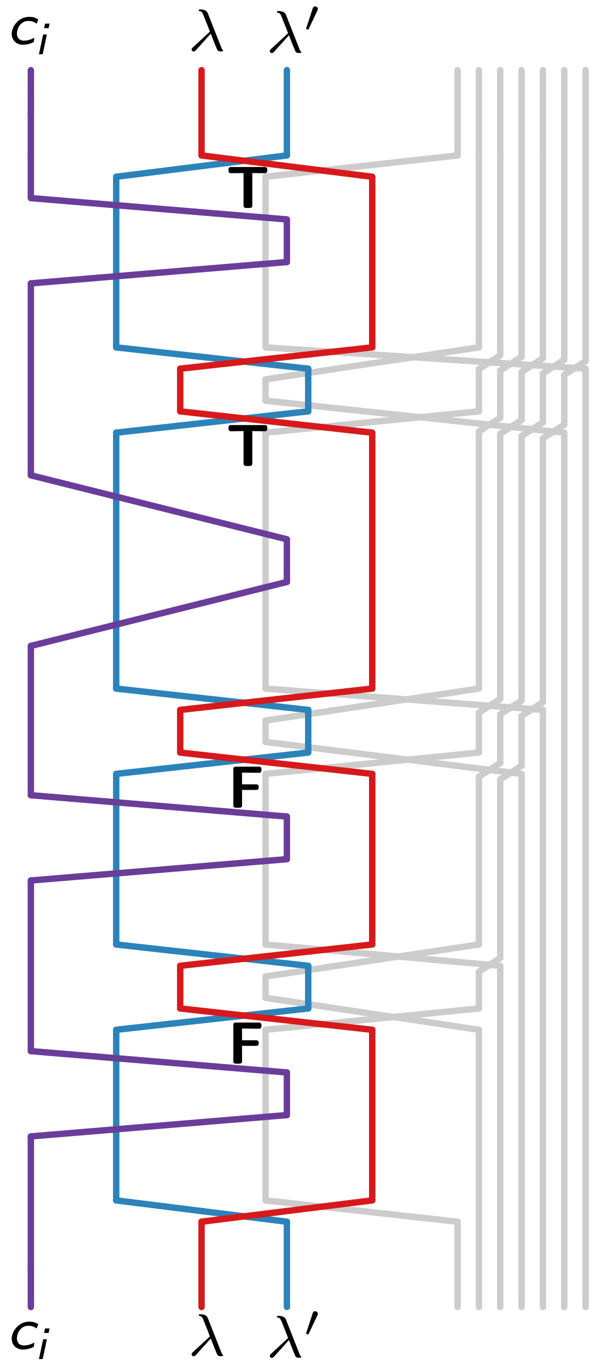
Clause Gadget



λ, λ': central loop structure

c_i: clause wire of i-th clause

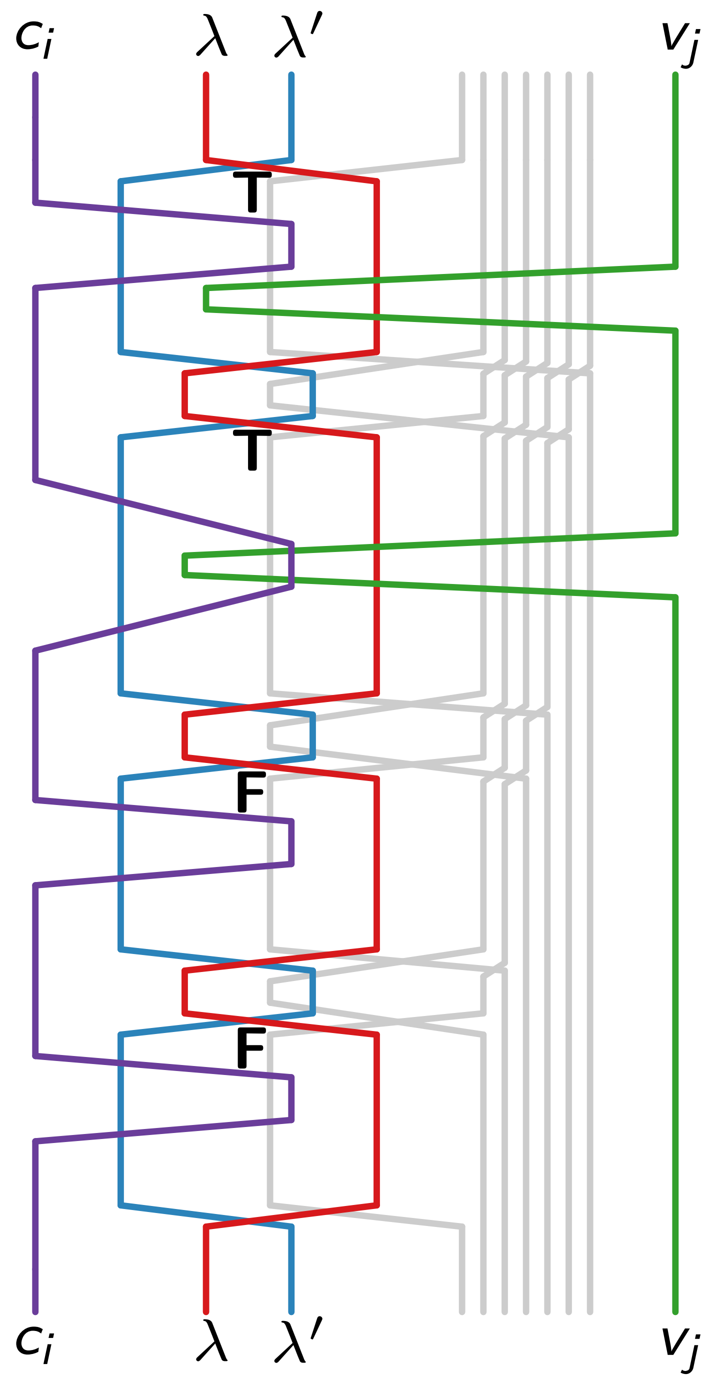
Clause Gadget



λ, λ': central loop structure

c_i: clause wire of i-th clause

Clause Gadget

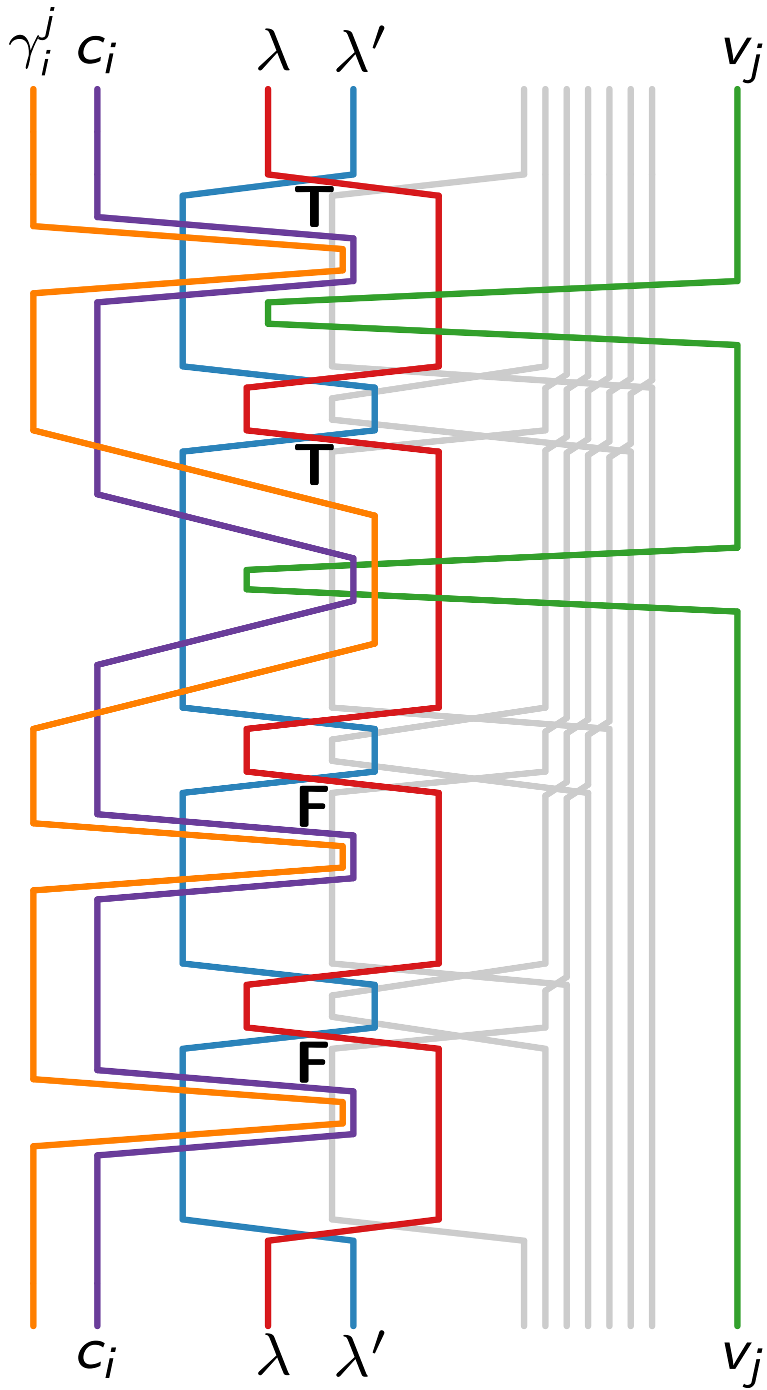


λ, λ': central loop structure

c_i: clause wire of i-th clause

v_j: variable wire of j-th variable

Clause Gadget



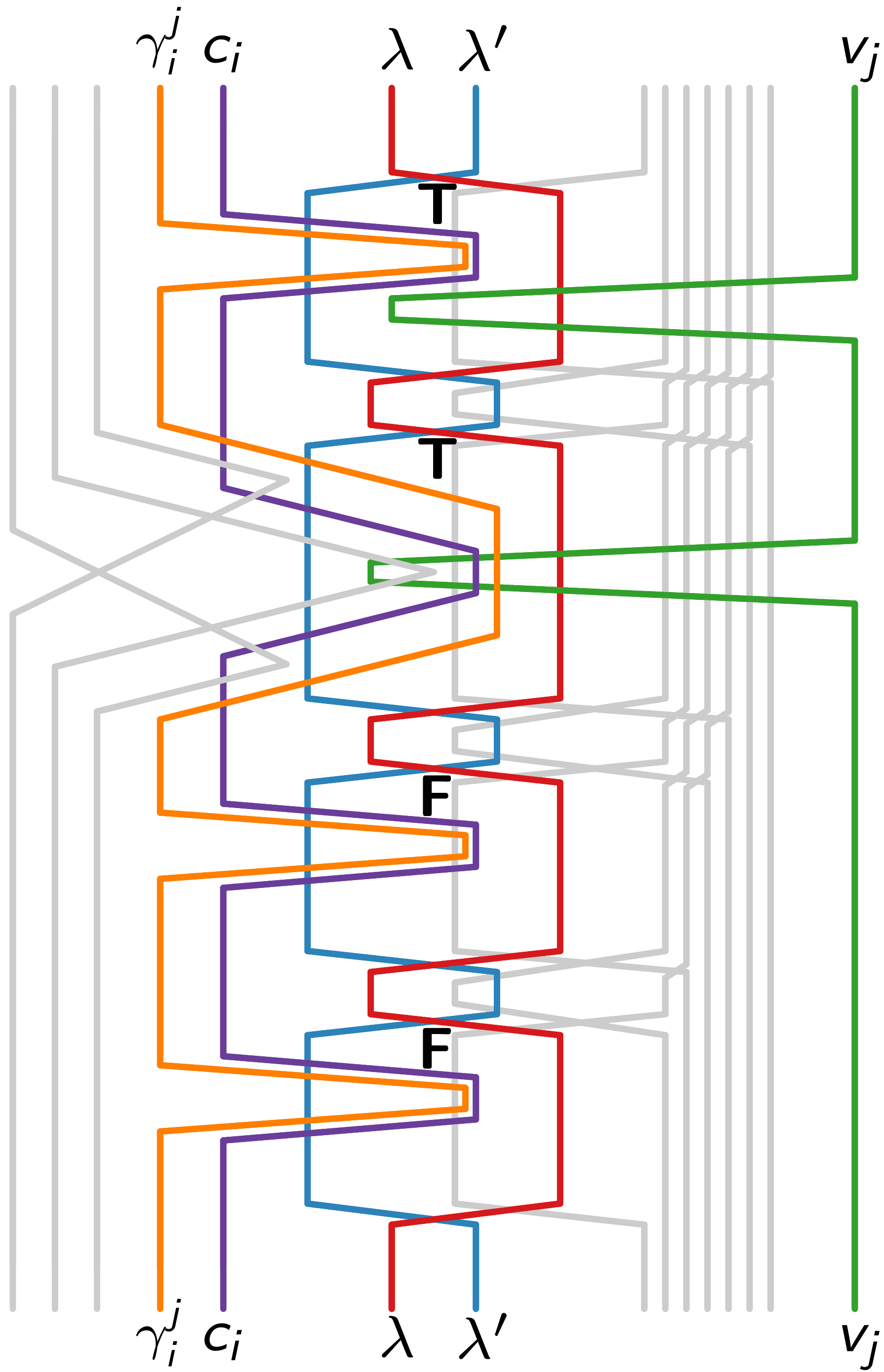
λ, λ': central loop structure

c_i: clause wire of i-th clause

v_j: variable wire of j-th variable

γ_i^j: protects the arm of c_i that intersects v_j from other variable wires

Clause Gadget



λ, λ': central loop structure

c_i: clause wire of i-th clause

v_j: variable wire of j-th variable

γ_i^j: protects the arm of c_i that intersects v_j from other variable wires

Open Problems

Problem 1

Can we decide the feasibility of a list L faster than finding an optimal-height tangle of L?

Open Problems

Problem 1

Can we decide the feasibility of a list L faster than finding an optimal-height tangle of L?

Problem 2

For lists where all entries are 0 or 1, we can find a tangle that has height at most OPT$+$ 1 in polynomial time. Can we also always find a tangle of height OPT efficiently?

Open Problems

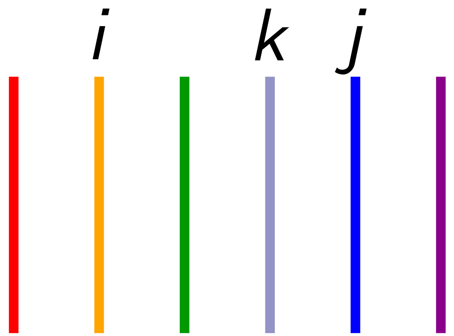
Problem 1

Can we decide the feasibility of a list L faster than finding an optimal-height tangle of L?

Problem 2

For lists where all entries are 0 or 1, we can find a tangle that has height at most OPT$+$ 1 in polynomial time. Can we also always find a tangle of height OPT efficiently?

Problem 3



A list is *non-separable* if $\forall i<k<j$: $\left(\ell_{ik} = \ell_{kj} = 0 \text{ implies } \ell_{ij} = 0\right)$.

Open Problems

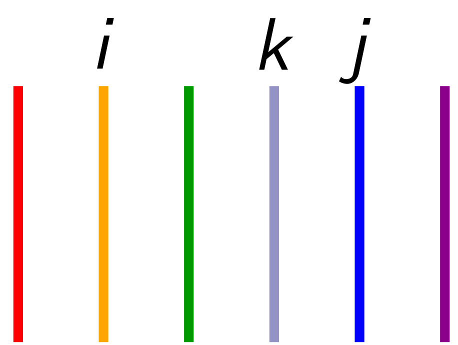
Problem 1

Can we decide the feasibility of a list L faster than finding an optimal-height tangle of L?

Problem 2

For lists where all entries are 0 or 1, we can find a tangle that has height at most $\mathrm{OPT}+1$ in polynomial time. Can we also always find a tangle of height OPT efficiently?

Problem 3



A list is *non-separable* if $\forall i<k<j$: $\left(\ell_{ik} = \ell_{kj} = 0 \text{ implies } \ell_{ij} = 0\right)$.

necessary

Open Problems

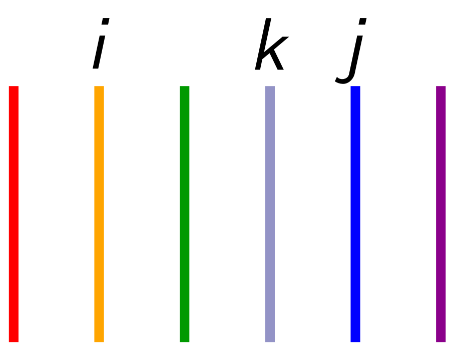
Problem 1

Can we decide the feasibility of a list L faster than finding an optimal-height tangle of L?

Problem 2

For lists where all entries are 0 or 1, we can find a tangle that has height at most OPT$+$ 1 in polynomial time. Can we also always find a tangle of height OPT efficiently?

Problem 3



A list is *non-separable* if $\forall i<k<j$: $\left(\ell_{ik} = \ell_{kj} = 0 \text{ implies } \ell_{ij} = 0\right)$.

necessary

For lists where all entries are even, is this sufficient?

Open Problems

Thank you!

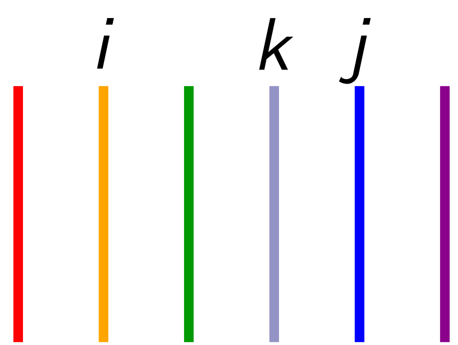
Problem 1

Can we decide the feasibility of a list L faster than finding an optimal-height tangle of L?

Problem 2

For lists where all entries are 0 or 1, we can find a tangle that has height at most OPT$+1$ in polynomial time. Can we also always find a tangle of height OPT efficiently?

Problem 3



A list is *non-separable* if $\forall i<k<j$: $\left(\ell_{ik} = \ell_{kj} = 0 \text{ implies } \ell_{ij} = 0\right)$.

necessary

For lists where all entries are even, is this sufficient?