Using Non-Deterministic Constraint Logic to prove 1×1 Rush-Hour is **PSPACE**-*complete*

Danish Alvi

Radboud Universitiet, Nijmegen

January 24, 2022

Plan

Reducing **TQBF** to Nondeterministic Constraint Logic (NCL)

2-color Oriented Subway Shuffle is PSPACE-complete (by reduction from NCL)

Reducing 2-color Oriented Subway Shuffle to 1×1 Rush Hour

Complexity Recap

Definition of **PSPACE**

Denoting **SPACE**$(t(n))$ as the set of problems that can be solved by a *deterministic* TM using $\mathcal{O}(t(n))$ space for some polynomial t of the input size n, then we can define **PSPACE** formally as

$$\mathbf{PSPACE} := \bigcup_{k \in \mathbb{N}} \mathbf{SPACE}(n^k)$$

NPSPACE is defined similarly with a NDTM, vis-à-vis **NSPACE**.

Definition of Completeness

For a complexity class **C**, a problem $L \in \mathbf{C}$ is **C**-*complete* if $\forall$ $L' \in L$, we have $L' \leq_p L$ (where $\leq_p$ is a poly-time reduction).

Complexity Recap

Definition **QBF**

A Quantified Boolean Formula (**QBF**) is a first-order logic formula φ of the form

$$\varphi(\vec{x}) := Q_1 Q_2 \ldots Q_n \; \phi(x_1, \ldots, x_n)$$

where $Q_1 \ldots Q_n$ are either $\exists$ or $\forall$, ϕ is a propositional logic formula in n variables, and $\vec{x}$ is an assignment to those n variables.

Definition

TQBF is defined to be the set of true **QBF** formulae.

Complexity Recap

Savitch's Theorem

$$\mathbf{PSPACE} = \mathbf{NPSPACE}$$

Proof.

Computational Complexity: A Modern Approach, Thm 4.12 □

Theorem

TQBF *is* **PSPACE***-complete problem.*

Proof.

Computational Complexity: A Modern Approach, Thm 4.11 □

Tic Tac Toe

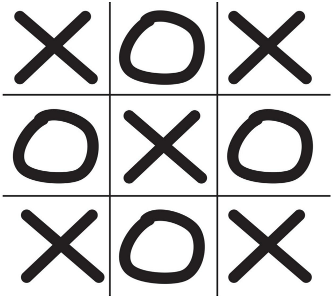


Figure: Tic-Tac-Toe is a popular game which is PSPACE-complete

Reducing **TQBF** to Nondeterministic Constraint Logic (NCL)

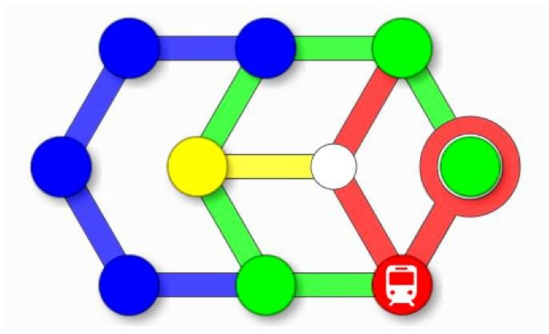
Theorem

NCL *is a* **NP***-complete and* **PSPACE***-complete problem.*

Proof.

Games, puzzles, and computation. Robert Hearn, Erik Demaine. □

Subway Shuffle



Subway Shuffle

Definition

Given a planar, unoriented graph where each edge is colored and some (not all) vertices contain a colored token,

- A *legal move* is to move a token across an edge of the same color to an empty vertex
- The *goal* is to move the designated token (bus) to the designated target vertex (station)

2-Color Oriented Subway Shuffle

Definition

Given a planar *oriented* graph where each edge is either colored orange or purple and some (not all) vertices contain a token (colored either orange or purple),

- A *legal move* is to move a token across an edge of the same color to an empty vertex, in direction of the edge, and flipping the direction of the edge
- The *goal* is to move the designated token (bus) to the designated target vertex (station)
- A vertex is *valid* if it has atmost degree 3, and has atmost two edges of the same color attached to it

2-Color Oriented Subway Shuffle

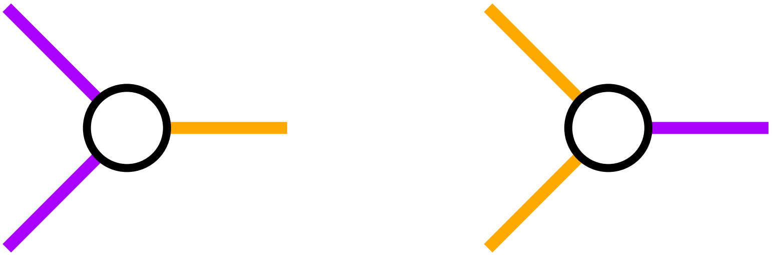


Figure: The set of *valid* vertices of degree 3. Any vertex of degree two is trivially valid.

A variant of 2-Color Oriented Subway Shuffle in **PSPACE**

Theorem

NCL, 2-Color Oriented Subway Shuffle with valid vertices and only one empty vertex, and Rush Hour are in PSPACE.

Proof.

Non-deterministically search the set of moves and accept when the special CL edge / token / car is at the designated configuration. Therefore, it is in **NPSPACE**, which by Savitch's Theorem, is equal to **PSPACE**. □

2-Color Oriented Subway Shuffle is **PSPACE**-Complete

Theorem

2-Color Oriented Subway Shuffle with valid vertices and only one empty vertex is **PSPACE***-hard.*

Proof.

We show a polynomial time reduction *from* non-deterministic constraint logic (NCL) *to* 2-Color Oriented Subway Shuffle. □

2-Color Oriented Subway Shuffle is **PSPACE**-Complete

Theorem

2-Color Oriented Subway Shuffle with valid vertices and only one empty vertex is **PSPACE***-hard.*

Proof

We show a polynomial time reduction *from* non-deterministic constraint logic (NCL) *to* 2-Color Oriented Subway Shuffle.

NCL $\leq_p$ 2-Color Oriented Subway Shuffle

Sketch of reduction

Given an instance of planar, non-deterministic constraint logic,

- Replace every edge in the NCL graph with a corresponding **EDGE** gadget (except the final edge to be flipped, which is replaced by a **WIN** gadget)
- Replace every **AND** vertex with the corresponding **AND** gadget; for other vertices construct the corresponding vertex using the **protected OR**
- Construct a mechanism within the 2-Color Oriented Subway Shuffle instance whereby each of these gadgets can be individually manipulated (as seen earlier in NCL).

EDGE gadgets

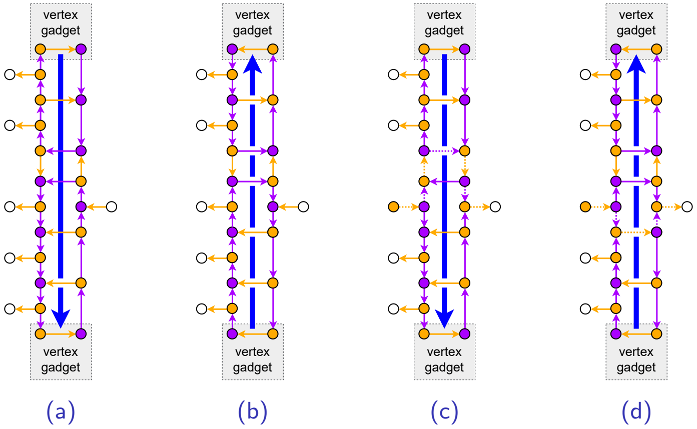


Figure: The edge gadget for 2-color oriented Subway Shuffle, shown (a) directed down and unlocked, (b) directed up and unlocked, (c) directed down after the bubble has passed through, and (d) directed up after the bubble has passed through. The red CL edge is same, but reflected.

Rotating cycles in the **EDGE** gadget

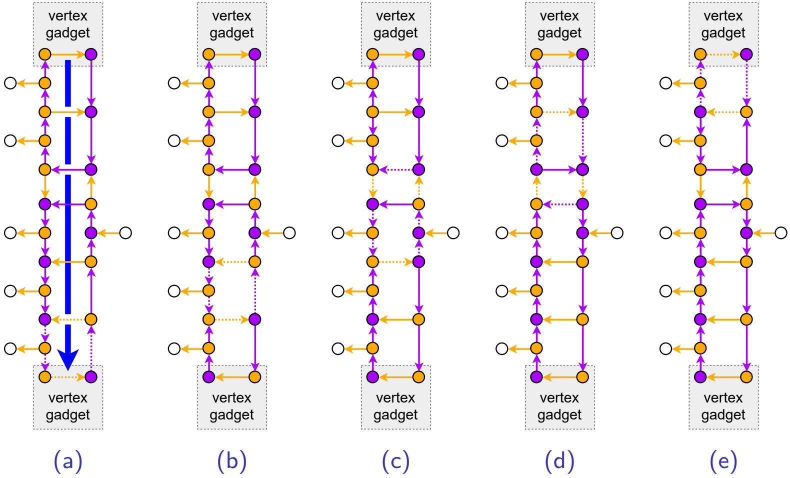
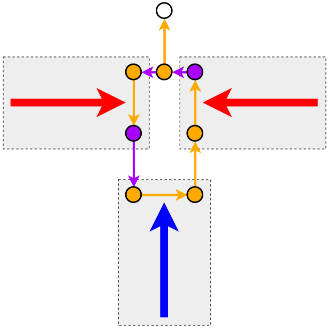
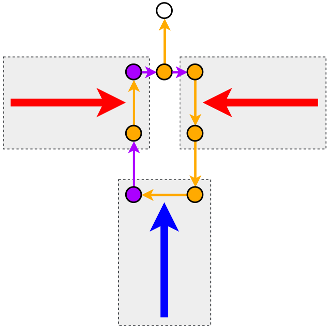


Figure: The orientation of edge gadget directed down is changed into the edge gadget directed up by rotating the five cycles (shown above)

AND gadget



(a) All edges oriented in, with the blue edge locked.



(b) All edges oriented in, with both red edges locked.

Figure: The AND vertex gadget for 2-color oriented Subway Shuffle.

protected OR gadget

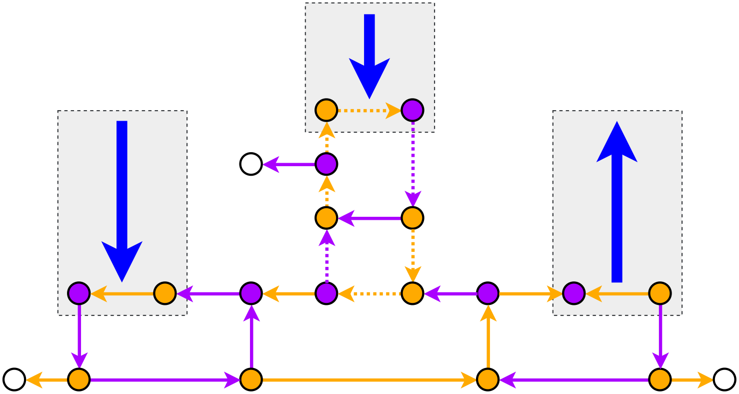


Figure: State 1: The left edge is locked. The middle edge is unlocked and pointing in. The right edge is pointing out.

protected OR gadget

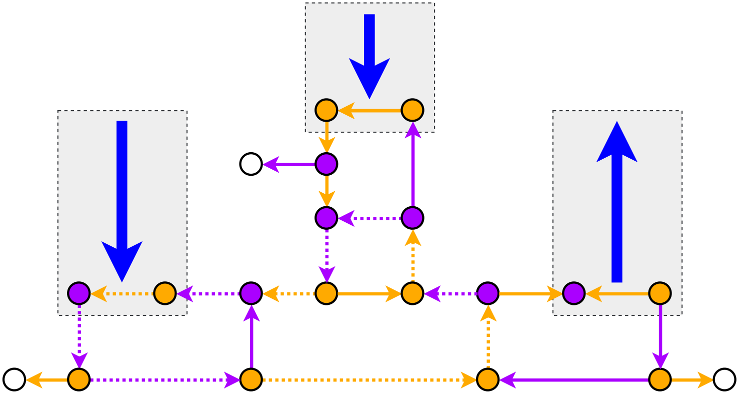


Figure: State 2: The left and middle edges are locked. The right edge is pointing out.

protected OR gadget

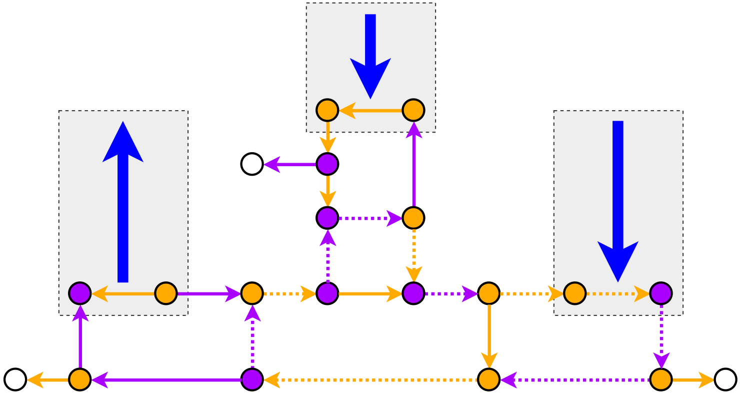


Figure: State 3: The left edge is pointing out. The middle edge is locked. The right edge is unlocked and pointing in.

protected OR gadget

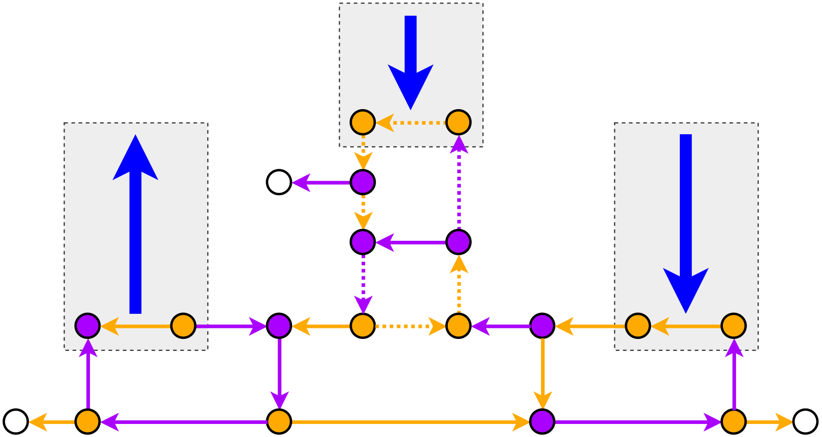


Figure: State 4: The left edge is pointing out. The middle and right edges are locked.

protected OR gadget

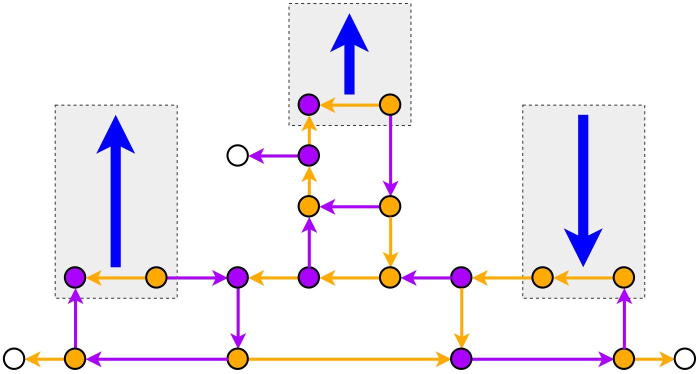


Figure: State 5: The left and middle edges are pointing out. The right edge is locked.

protected OR gadget

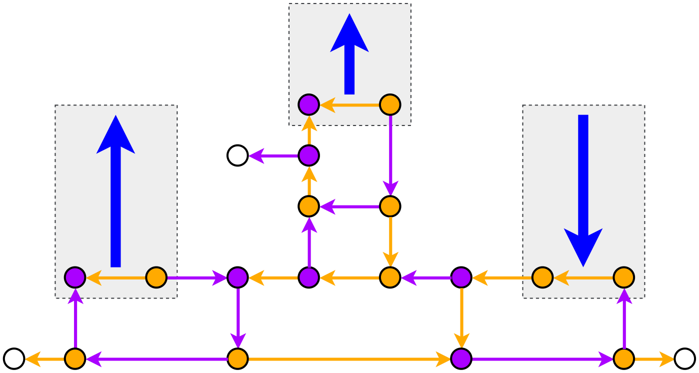
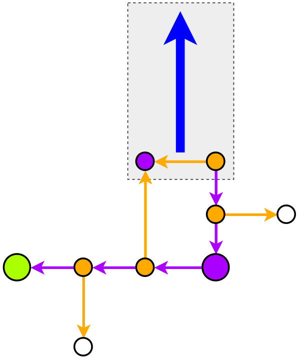
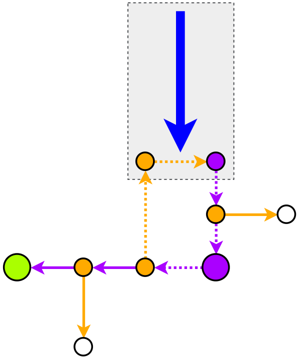


Figure: State 5: The left and middle edges are pointing out. The right edge is locked.

WIN gadget



(a) The locked win gadget. The bubble cannot do anything here.



(b) The unlocked win gadget. Now that the blue edge is pointing into the gadget, the indicated cycle can be rotated.

Figure: The target edge starts pointing away. If it is flipped, the bubble can enter the win gadget once at each entrance to move the (bottom right) purple special token to the (bottom left, green) target vertex.

An important observations of gadgets

Lemma 1

It is never useful for the bubble to enter an **EDGE** gadget directly from a vertex gadget through the shared vertices.

Proof

Lemma 1

It is never useful for the bubble to enter an **EDGE** gadget (in any configuration) directly from a vertex gadget through the shared vertices.

Proof.

In any of the (four) *initial* configurations the **EDGE** gadget is in, there is no path the bubble can follow except in the upwards pointing edge gadget (except from the top-left purple token where the bubble can travel five tokens before getting stuck). Also, reconfirm the same for the partial **EDGE** configurations. □

An important observations of gadgets

Lemma 2

It is never useful to turn some of the cycles in an edge gadget without turning all of them.

An important observations of gadgets

Lemma 2

It is never useful to turn some of the cycles in an edge gadget without turning all of them.

Proof.

Before turning the any cycle, we need to turn the previous cycle (as shown in the rotation of the five cycles), as each cycle shares at least one edge with the previous one (initially in the opp. direction). At the first cycle, we reverse the edge shared with the connecting vertex gadget, and thus make the gadget pointing in the outward configuration (at that vertex), which is not needed. □

Important observations of gadgets

Lemma 3

Entering any vertex gadget from an **EDGE** gadget is not useful.

Proof.

Skipped. □

Important observations of gadgets

Lemma 4

The **EDGE** gadget cannot (usefully) perform any other moves other than turning all cycles (hence simulating the NCL instance by flipping the underlying NCL edge)

Important observations of gadgets

Lemma 4

The **EDGE** gadget cannot (usefully) perform any other moves other than turning all cycles (hence simulating the NCL instance by flipping the underlying NCL edge)

Proof.

There are two possible legal moves that can be done (in between turning cycles). The first one is exiting during the third cycle, which by Lemma 2, is not useful. The second one is the bubble entering a connecting vertex during the any of the cycles except the second cycle. By Lemma 3, this is not useful. □

Important observations of gadgets

Lemma 5

The vertex gadgets cannot perform any other moves other than the ones which simulate an NCL instance by simulating an NCL vertex)

Proof.

Skipped. □

Important observations of gadgets

Lemma 6

If the bubble takes any path from any vertex to the same vertex which does not complete a nontrivial loop, then the state of the Subway Shuffle instance must not have changed.

Proof.

If the bubble never completed a loop, then the only way for it to get back to where it started is to take the same path in reverse. By the definition of Subway Shuffle moves, this exactly undoes these moves returning the instance back to its original state. □

Connecting the individual gadgets

Sketch of construction

- Consider (any) rooted spanning tree of the dual graph of the (unoriented) CL graph $\mathcal{T}$
- The vertices of $\mathcal{T}$ correspond to the faces of the CL graph, while each of the edges of $\mathcal{T}$ have an associated dual edge in the CL graph. For these edges $\mathcal{E}'$ in the CL graph, we also include the rightmost vertex called the *exit*.
- We place the edge gadget in the orientation so that the entrance is on the face closer to the root of the spanning tree of the dual graph, therefore ensuring each face has exactly one exit vertex
- This allows the bubble, when at the third entrance of the edge gadget in $\mathcal{E}'$ to pass through, to the face on the other side of the edge.

Connecting the individual gadgets

Sketch of Construction (... continued)

- *Within* each face F, we use (any) tree $\mathcal{T}_F$ to connect all the entrances of the gadgets in the face to and towards the exit of that face. The color of the edges is irrelevant but the token in the tail should match the color of the entrance and all vertices should be valid.
- For the face corresponding to the root of the spanning tree, the tree connecting entrances has one vertex without a token, and the edges point towards it; this is where the bubble starts
- Therefore, via $\mathcal{T}$, the bubble can begins at the root of the spanning tree, moves down the tree by crossing edge gadgets in $\mathcal{E}'$ until reaching a desired face. The bubble enters a vertex or edge gadget, goes around a cycle, and exits, and can proceed to either going to another gadget in $\mathcal{T}_F$ or another face via $\mathcal{T}$.

1×1 Rush Hour

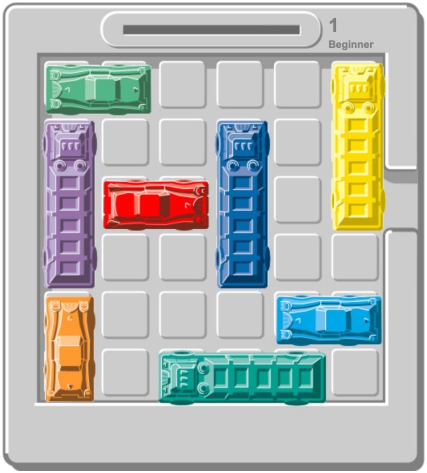


Figure: Taken from mathsonline.org

1×1 Rush Hour

Definition

- Given $n^2 - 1$ unit-square blocks in an $n \times n$ square board (hence there is one *bubble*)
- Each block is labeled as movable horizontally (only), movable vertically (only), or immovable (*fixed*)
- A legal move is to move a car one square in either direction along its orientation, provided that it remains within the square and does intersect another car
- The goal is for a designated special car to reach the left edge of the board.

Simulating Subway Shuffle using 1×1 Rush Hour

- We replace each vertex in our Subway Shuffle instance with a single car (of the same color) which is vertical if there is an orange token there, and horizontal if a purple token is there
- A Subway Shuffle edge is simulated by a path of cars. This path can make right-angle turns, allowing us to embed an arbitrary planar Subway Shuffle graph. The direction of a car at a turn in an edge defines which way the Subway Shuffle edge is oriented.

Example of a simulating an edge in Subway Shuffle

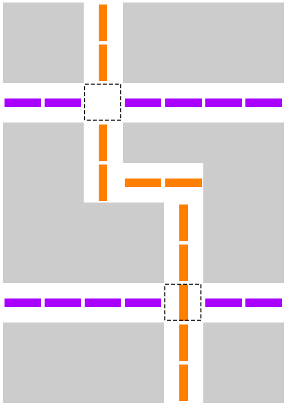


Figure: Simulating an orange edge with an oange token at the tail and the bubble in the head.

Example of a simulating an edge in Subway Shuffle

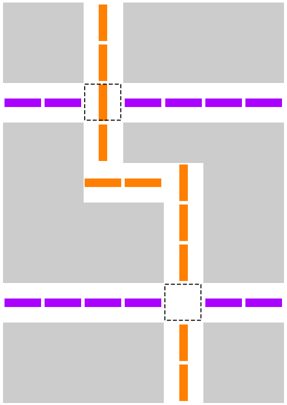


Figure: The orange token from the previous diagram travels into the bubble.

Directionality of Rush Hour Edge

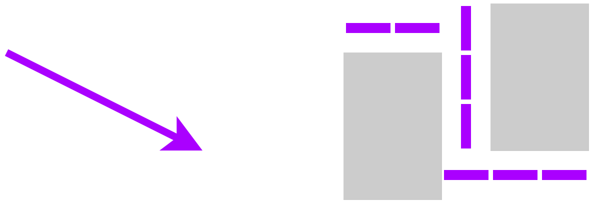


Figure: A Rush Hour simulation of a Subway Shuffle edge. This is a purple edge which points right.

WIN gadget

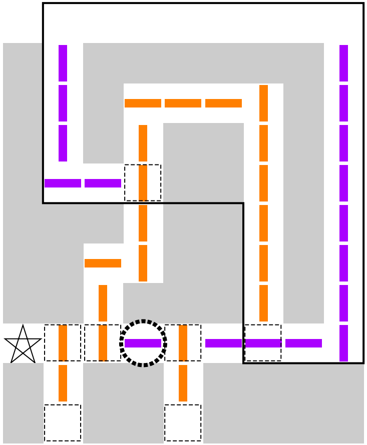


Figure: The *unlocked* **WIN** gadget. The region which is inside the black solid line corresponds to edge shared with the **EDGE** gadget.

Useful Resources

- Josh Brunner, Lily Chung, Erik D. Demaine, Dylan Hendrickson, Adam Hesterberg, Adam Suhl, and Avi Zeff. 2020. *1 x 1 Rush Hour with Fixed Blocks is PSPACE-complet*e.
- Robert A Hearn and Erik D Demaine. 2009. *Games, puzzles, and computation*. CRC Press.
- François Schwarzentruber. *Constraint Graphs.* people.irisa.fr