

Past is for Free:
on the Complexity of Verifying
Linear Temporal Properties with Past

N. Markey

`markey@lifo.univ-orleans.fr`

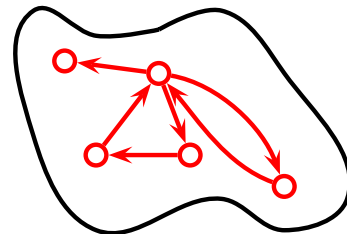
LIFO, Univ. Orléans & CNRS FRE 2490

Introduction: Model checking

Does this given **system** satisfy that **property** ?

Introduction: Model checking

Does this given **system** satisfy that **property** ?



\models aUb
?

Introduction: Temporal Logic

- Temporal logics are nice specification languages.
“Any problem is followed eventually by an alarm”

$G(\text{problem} \Rightarrow F \text{alarm})$

Introduction: Temporal Logic

- Temporal logics are nice specification languages.

“Any problem is followed eventually by an alarm”

$$\mathbf{G}(\text{problem} \Rightarrow \mathbf{F} \text{alarm})$$

- Past operators make specification easier to write.

“Whenever the alarm rings, then there **has been** some problem in the past”

$$\mathbf{G}(\text{alarm} \Rightarrow \mathbf{F}^{-1} \text{problem})$$

Introduction: Temporal Logic

- Temporal logics are nice specification languages.

“Any problem is followed eventually by an alarm”

$$\mathbf{G}(\text{problem} \Rightarrow \mathbf{F} \text{alarm})$$

- Past operators make specification easier to write.

“Whenever the alarm rings, then there **has been** some problem in the past”

$$\mathbf{G}(\text{alarm} \Rightarrow \mathbf{F}^{-1} \text{problem})$$

We can express this property without \mathbf{F}^{-1} :

$$\neg \left((\neg \text{problem}) \mathbf{U} (\text{alarm} \wedge \neg \text{problem}) \right)$$

Outline

- Definitions
- Past is more expressive
 - Same expressive power
 - More succinctness
- Past is for free
 - Main results
 - Upper and lower bounds
 - NP-easiness for $\mathbf{L}(\mathbf{F}, \mathbf{F}^{-1})$
 - PSPACE-hardness for $\mathbf{L}^+(\mathcal{U})$
- Conclusion

Definition

PLTL formulae are built from:

- atomic propositions (For ex. **problem**, **alarm**),
- boolean combinators (\wedge , \vee , \neg)

Definition

PLTL formulae are built from:

- atomic propositions (For ex. **problem**, **alarm**),
- boolean combinators (\wedge , \vee , \neg)
- future modalities: **X**, **U**

(LTL)

Definition

PLTL formulae are built from:

- atomic propositions (For ex. **problem**, **alarm**),
- boolean combinators (\wedge , \vee , \neg)
- future modalities: X , U
- past modalities: X^{-1} , S

Definition

PLTL formulae are built from:

- atomic propositions (For ex. **problem**, **alarm**),
- boolean combinators (\wedge , \vee , \neg)
- future modalities: \mathbf{X} , \mathcal{U}
- past modalities: \mathbf{X}^{-1} , \mathcal{S}
- **plus** all the standard abbreviations:

$$\begin{array}{ll} \mathbf{F} \varphi \stackrel{\text{def}}{=} \top \mathcal{U} \varphi & \mathbf{G} \varphi \stackrel{\text{def}}{=} \neg \mathbf{F} \neg \varphi \\ \mathbf{F}^{-1} \varphi \stackrel{\text{def}}{=} \top \mathcal{S} \varphi & \mathbf{G}^{-1} \varphi \stackrel{\text{def}}{=} \neg \mathbf{F}^{-1} \neg \varphi \end{array}$$

Fragments of PLTL

We consider fragments of PLTL built using the following restrictions:

- restriction on the allowed modalities: for instance, $\mathbf{L}(\mathcal{U}, \mathcal{S})$ is the fragment of PLTL where \mathbf{X} and \mathbf{X}^{-1} are not allowed;

Fragments of PLTL

We consider fragments of PLTL built using the following restrictions:

- restriction on the allowed modalities: for instance, $\mathbf{L}(\mathcal{U}, \mathcal{S})$ is the fragment of PLTL where \mathbf{X} and \mathbf{X}^{-1} are not allowed;
- restriction on the use of negations: in $\mathbf{L}^+(\dots)$, negation cannot be applied to modalities;

Fragments of PLTL

We consider **fragments of PLTL** built using the following restrictions:

- **restriction on the allowed modalities**: for instance, $\mathbf{L}(\mathcal{U}, \mathcal{S})$ is the fragment of **PLTL** where \mathbf{X} and \mathbf{X}^{-1} are not allowed;
- **restriction on the use of negations**: in $\mathbf{L}^+(\dots)$, negation cannot be applied to modalities;
- **restriction on the nesting of modalities**: in $\mathbf{L}_s(\dots)$, **future-time modalities** cannot appear in the scope of **past-time modalities**.

Semantics

PLTL formulas are interpreted at some position i along an infinite labelled path (π, ξ) .

classical semantics for atomic propositions and boolean combinators

$$\pi, i \models \mathbf{X}\phi \quad \Leftrightarrow \quad \pi, i + 1 \models \phi$$

$$\pi, i \models \phi \mathbf{U} \psi \quad \Leftrightarrow \quad \exists j \geq i \left((\pi, j \models \psi) \wedge (\forall i \leq k < j, \pi, k \models \phi) \right)$$

$$\pi, i \models \mathbf{X}^{-1}\phi \quad \Leftrightarrow \quad i \geq 1 \wedge \pi, i - 1 \models \phi$$

$$\pi, i \models \phi \mathbf{S} \psi \quad \Leftrightarrow \quad \exists j \leq i \left((\pi, j \models \psi) \wedge (\forall j < k \leq i, \pi, k \models \phi) \right)$$

Expressive power

Two formulas ϕ and ψ are *initially* equivalent ($\phi \equiv_i \psi$) if, and only if,

$$\forall \pi, \pi, 0 \models \phi \Leftrightarrow \pi, 0 \models \psi$$

Expressive power

Two formulas ϕ and ψ are *initially* equivalent ($\phi \equiv_i \psi$) if, and only if,

$$\forall \pi, \pi, 0 \models \phi \Leftrightarrow \pi, 0 \models \psi$$

Two formulas ϕ and ψ are *globally* equivalent ($\phi \equiv \psi$) if, and only if,

$$\forall \pi, \forall i \in \mathbb{N}, \pi, i \models \phi \Leftrightarrow \pi, i \models \psi$$

Expressive power

Two formulas ϕ and ψ are *initially* equivalent ($\phi \equiv_i \psi$) if, and only if,

$$\forall \pi, \pi, 0 \models \phi \Leftrightarrow \pi, 0 \models \psi$$

Two formulas ϕ and ψ are *globally* equivalent ($\phi \equiv \psi$) if, and only if,

$$\forall \pi, \forall i \in \mathbb{N}, \pi, i \models \phi \Leftrightarrow \pi, i \models \psi$$

For example:

$$\phi \mathcal{S} \psi \equiv_i \psi \quad \mathbf{X}^{-1} \phi \equiv_i \perp$$

$$\mathbf{F}^{-1}(a \vee b) \equiv \mathbf{F}^{-1}a \vee \mathbf{F}^{-1}b$$

Expressive power

Two formulas ϕ and ψ are *initially* equivalent ($\phi \equiv_i \psi$) if, and only if,

$$\forall \pi, \pi, 0 \models \phi \Leftrightarrow \pi, 0 \models \psi$$

Two formulas ϕ and ψ are *globally* equivalent ($\phi \equiv \psi$) if, and only if,

$$\forall \pi, \forall i \in \mathbb{N}, \pi, i \models \phi \Leftrightarrow \pi, i \models \psi$$

For example:

$$\phi \mathcal{S} \psi \equiv_i \psi \quad \mathbf{X}^{-1} \phi \equiv_i \perp$$

$$\mathbf{F}^{-1}(a \vee b) \equiv \mathbf{F}^{-1}a \vee \mathbf{F}^{-1}b$$

These equivalences can be relativised to a **class Π of paths**.

Verification problems

- **validity**: A formula ϕ is *valid* if, and only if, it is equivalent to \top .

Verification problems

- **validity**: A formula ϕ is *valid* if, and only if, it is equivalent to \top .
- **satisfiability**: A formula ϕ is *satisfiable* if, and only if, its **negation** is **not valid**.

Verification problems

- **validity**: A formula ϕ is *valid* if, and only if, it is equivalent to \top .
- **satisfiability**: A formula ϕ is *satisfiable* if, and only if, its **negation** is **not valid**.

Several flavours: *initial* or *global*, relativization to a **restricted class of paths**.

Verification problems

- **validity**: A formula ϕ is *valid* if, and only if, it is equivalent to \top .
- **satisfiability**: A formula ϕ is *satisfiable* if, and only if, its **negation** is **not valid**.

Several flavours: *initial* or *global*, relativization to a **restricted class of paths**.

→ **initial satisfiability**

Verification problems

- **validity**: A formula ϕ is *valid* if, and only if, it is equivalent to \top .
- **satisfiability**: A formula ϕ is *satisfiable* if, and only if, its **negation** is **not valid**.

Several flavours: *initial* or *global*, relativization to a **restricted class of paths**.

→ initial satisfiability

→ existential model checking

Verification problems

- **validity**: A formula ϕ is *valid* if, and only if, it is equivalent to \top .
- **satisfiability**: A formula ϕ is *satisfiable* if, and only if, its **negation** is **not valid**.

Several flavours: *initial* or *global*, relativization to a **restricted class of paths**.

→ initial satisfiability

→ existential model checking

→ universal model checking

Expressive power

- It is well known that past operators **do not add** expressive power:

“Any **PLTL** formula is *initially equivalent* to an **LTL** formula.” [Kam68, GPSS80]

Expressive power

- It is well known that past operators **do not add** expressive power:

“Any **PLTL** formula is *initially equivalent* to an **LTL** formula.” [Kam68, GPSS80]

- This result is based on the **separation property**:

“Any **PLTL** formula is *equivalent* to a boolean combination of pure-future and pure-past formulae.” [Gab89]

It is assumed that this translation is **non-elementary**.

Expressive power

- It is well known that past operators **do not add** expressive power:

“Any **PLTL** formula is *initially equivalent* to an **LTL** formula.” [Kam68, GPSS80]

- This result is based on the **separation property**:

“Any **PLTL** formula is *equivalent* to a boolean combination of pure-future and pure-past formulae.” [Gab89]

It is assumed that this translation is **non-elementary**.

- Using **automata theory**, it is possible to translate a **PLTL** formula into an equivalent **LTL** formula. [Mal90, Wil99]

This translation is **triple-exponential**.

Expressive power - succinctness

Theorem: PTL can be exponentially more succinct than LTL .

[LMS02]

Expressive power - succinctness

Theorem: **PLTL** can be exponentially more succinct than **LTL**.

[LMS02]

Proof: Let $\{p_0, p_1, \dots, p_n\}$ be a set of atomic propositions.

Expressive power - succinctness

Theorem: **PLTL** can be exponentially more succinct than **LTL**. [LMS02]

Proof: Let $\{p_0, p_1, \dots, p_n\}$ be a set of atomic propositions.

- The property “any two future states that agree on p_1, \dots, p_n also agree on p_0 ” can only be expressed by **PLTL** (or **LTL**) formulae of size at least $\Omega(2^n)$. [EVW97].

Expressive power - succinctness

Theorem: PLTL can be exponentially more succinct than LTL. [LMS02]

Proof: Let $\{p_0, p_1, \dots, p_n\}$ be a set of atomic propositions.

- The property “any two future states that agree on p_1, \dots, p_n also agree on p_0 ” can only be expressed by PLTL (or LTL) formulae of size at least $\Omega(2^n)$. [EVW97].
- The PLTL formula

$$\Phi \stackrel{\text{def}}{=} \mathbf{G} \left[\left(\bigwedge_{i=1}^n (p_i \Leftrightarrow \mathbf{F}^{-1}(\neg \mathbf{X}^{-1} \top \wedge p_i)) \right) \Rightarrow \left(p_0 \Leftrightarrow \mathbf{F}^{-1}(\neg \mathbf{X}^{-1} \top \wedge p_0) \right) \right]$$

states that “any future state that agrees with the initial state on p_1, \dots, p_n also agrees on p_0 ”. Let Ψ be an LTL formula initially equivalent to Φ .

Expressive power - succinctness

Theorem: **PLTL** can be exponentially more succinct than **LTL**. [LMS02]

Proof: Let $\{p_0, p_1, \dots, p_n\}$ be a set of atomic propositions.

- The property “any two future states that agree on p_1, \dots, p_n also agree on p_0 ” can only be expressed by **PLTL** (or **LTL**) formulae of size at least $\Omega(2^n)$. [EVW97].

- The **PLTL** formula

$$\Phi \stackrel{\text{def}}{=} \mathbf{G} \left[\left(\bigwedge_{i=1}^n (p_i \Leftrightarrow \mathbf{F}^{-1}(\neg \mathbf{X}^{-1} \top \wedge p_i)) \right) \Rightarrow \left(p_0 \Leftrightarrow \mathbf{F}^{-1}(\neg \mathbf{X}^{-1} \top \wedge p_0) \right) \right]$$

states that “any future state that agrees with the initial state on p_1, \dots, p_n also agrees on p_0 ”. Let Ψ be an **LTL** formula initially equivalent to Φ .

- Therefore $(\mathbf{G} \Psi)$ expresses the first property and $|\Psi|$ is in $\Omega(2^n)$!

Main results

The **complexity** of all the fragments we considered can be obtained from the following table:

	satisfiability	exist. model ch.	univ. model ch.
$L^+(\mathbf{F}), L^+(\mathbf{G}), L^+(\mathbf{X})$	NP-complete		coNP-complete
$L(\mathbf{F}, \mathbf{F}^{-1})$			
$L^+(\mathbf{F}, \mathbf{X}), L_s^+(\mathbf{F}, \mathbf{X}^{-1})$	NP-complete		PSPACE-complete
$L^+(\mathbf{F}, \mathbf{X}, \mathbf{F}^{-1}, \mathbf{X}^{-1})$			
$L^+(\mathbf{G}, \mathbf{X}), L_s^+(\mathbf{G}, \mathbf{X}^{-1})$	PSPACE-complete		coNP-complete
$L^+(\mathbf{G}, \mathbf{X}, \mathbf{G}^{-1}, \mathbf{X}^{-1})$			
$L_s^+(\mathbf{G}, \mathcal{S})$	NP-complete	PSPACE-complete	
$L^+(\mathcal{U}), L_s^+(\mathbf{F}, \mathcal{S})$	PSPACE-complete		
PLTL			

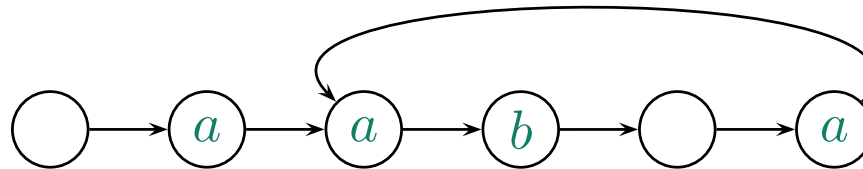
NP-easiness proofs

General method for proving NP-easiness:

- a satisfiable formula has an ultimately periodic witness [SC85],
- we provide a polynomial algorithm for model checking PLTL formulas along an ultimately periodic path,
- for some fragments, we prove that the ultimately periodic witness can be of polynomial size. Hence these fragments will have NP-easy satisfiability problems.

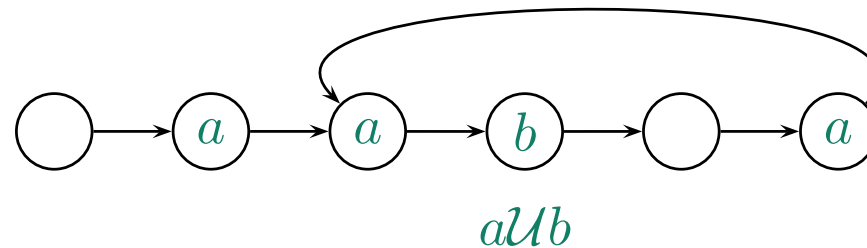
Model checking PLTL formulas along a path

- for **LT**L formulas, we can apply the same **labelling algorithm** as for **CTL**:



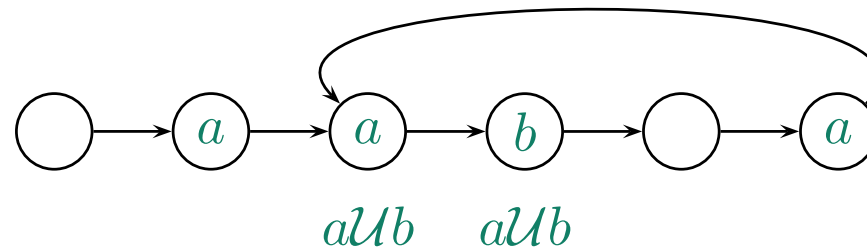
Model checking PLTL formulas along a path

- for **LT**L formulas, we can apply the same **labelling algorithm** as for **CTL**:



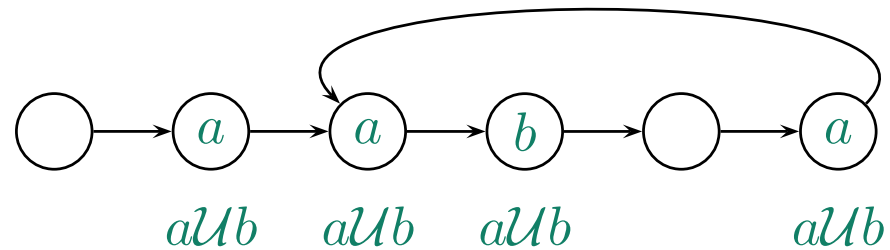
Model checking PLTL formulas along a path

- for **LT**L formulas, we can apply the same **labelling algorithm** as for **CT**L:



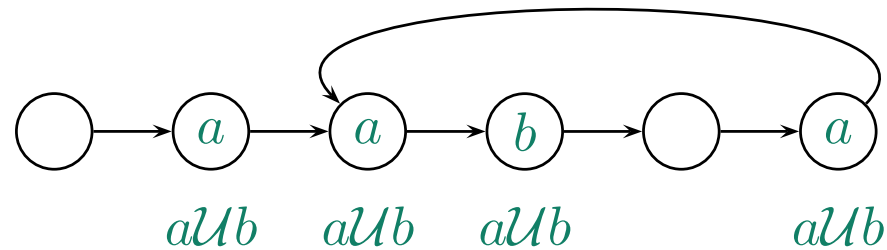
Model checking PLTL formulas along a path

- for **LT**L formulas, we can apply the same **labelling algorithm** as for **CTL**:

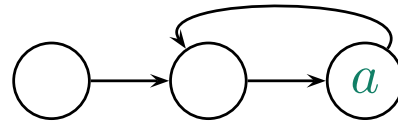


Model checking PLTL formulas along a path

- for **LTL** formulas, we can apply the same **labelling algorithm** as for **CTL**:

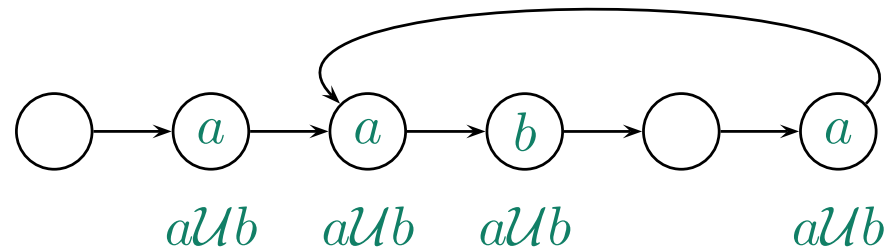


- this algorithm does not apply for **PLTL** formulas:

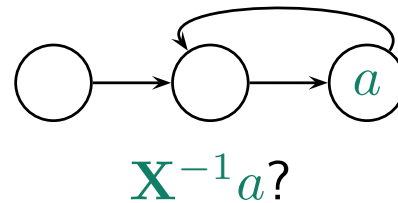


Model checking PLTL formulas along a path

- for **LTL** formulas, we can apply the same **labelling algorithm** as for **CTL**:

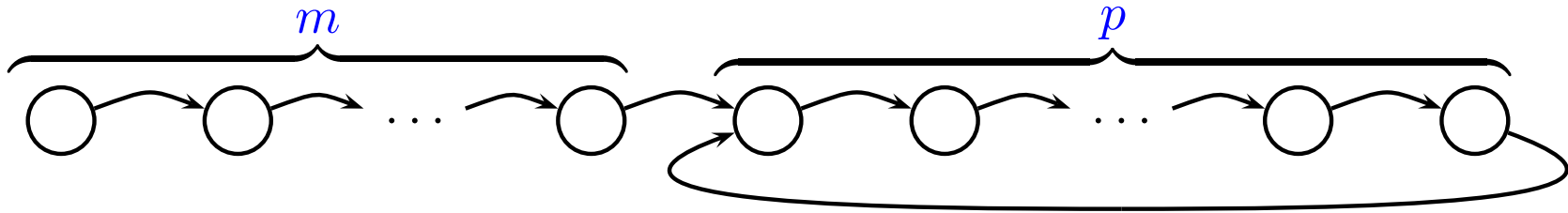


- this algorithm does not apply for **PLTL** formulas:



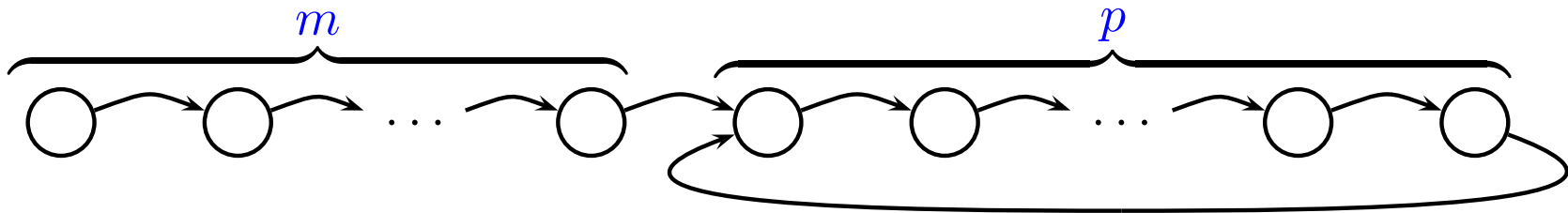
Model checking PLTL formulas along a path

A *loop of type (m, p)* is an ultimately periodic Kripke structure where the **initial part** has length m and the **periodic part** has length p .



Model checking PLTL formulas along a path

A *loop of type* (m, p) is an ultimately periodic Kripke structure where the **initial part** has length m and the **periodic part** has length p .

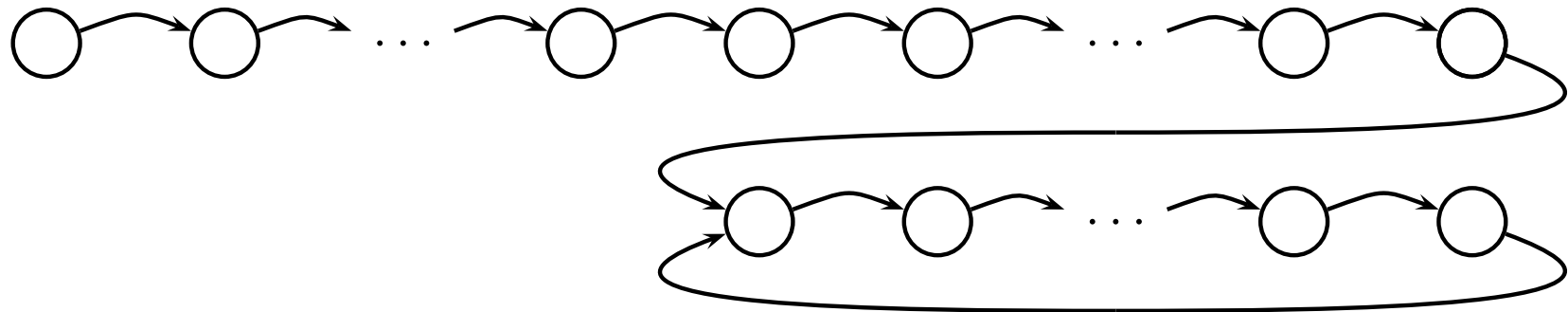


Lemma: For any loop L of type (m, p) , for any PLTL formula ϕ with at most $h(\phi)$ nested past-time modalities, and for any $k \geq m + p \cdot h(\phi)$, we have

$$\pi, k \models \phi \quad \Leftrightarrow \quad \pi, k + p \models \phi.$$

Model checking PLTL formulas along a path

A *loop of type* (m, p) is an ultimately periodic Kripke structure where the **initial part** has length m and the **periodic part** has length p .

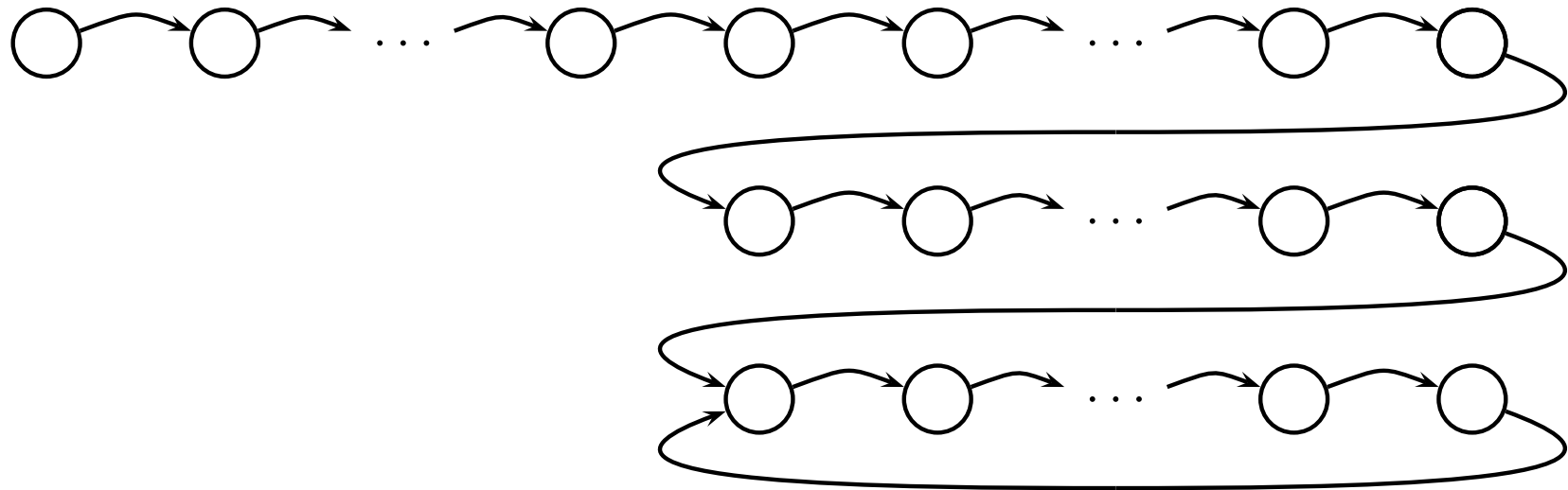


Lemma: For any loop L of type (m, p) , for any PLTL formula ϕ with at most $h(\phi)$ nested past-time modalities, and for any $k \geq m + p \cdot h(\phi)$, we have

$$\pi, k \models \phi \quad \Leftrightarrow \quad \pi, k + p \models \phi.$$

Model checking PLTL formulas along a path

A *loop of type (m, p)* is an ultimately periodic Kripke structure where the **initial part** has length m and the **periodic part** has length p .

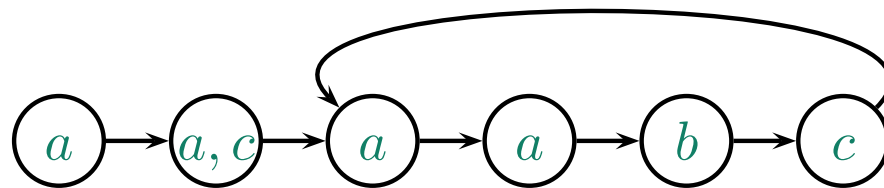


Lemma: For any loop L of type (m, p) , for any PLTL formula ϕ with at most $h(\phi)$ nested past-time modalities, and for any $k \geq m + p \cdot h(\phi)$, we have

$$\pi, k \models \phi \quad \Leftrightarrow \quad \pi, k + p \models \phi.$$

Satisfiability for $L(\mathbf{F}, \mathbf{F}^{-1})$ is in NP

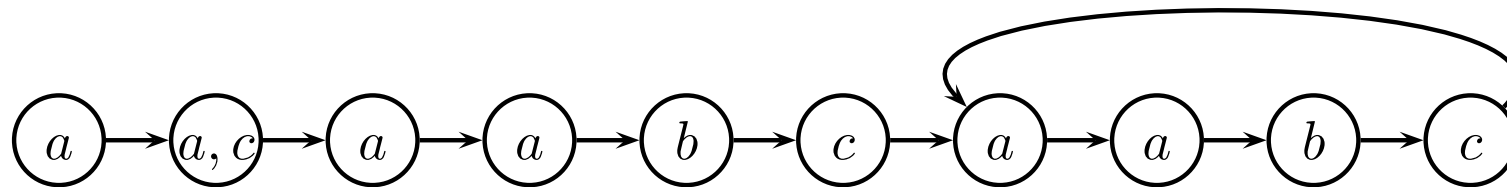
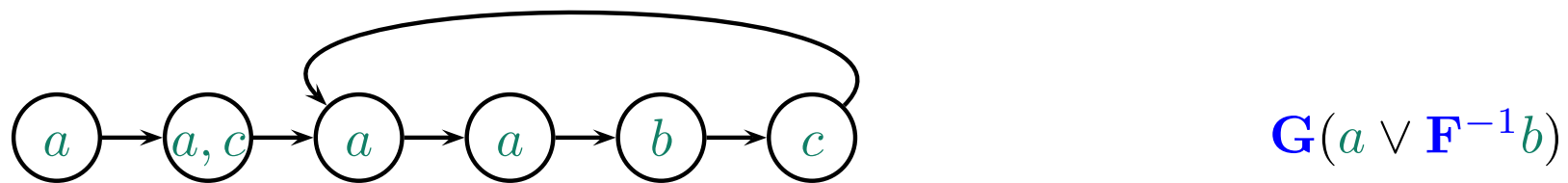
We keep only **polynomially** many states from an **ultimately periodic witness**: we first unwind the loop; then for each **subformula**, we keep the **first and the last state** satisfying that **subformula**, **if they exist**.



$$\mathbf{G}(a \vee \mathbf{F}^{-1}b)$$

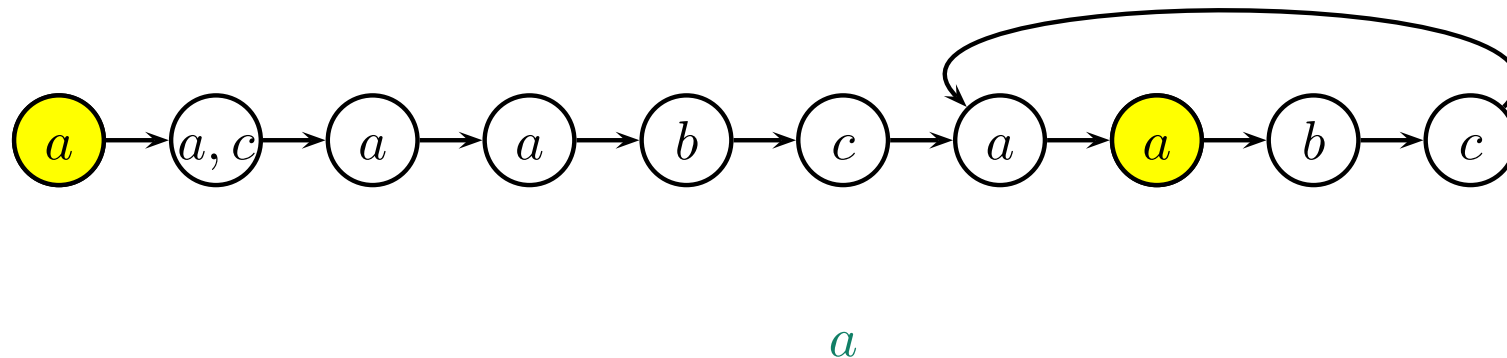
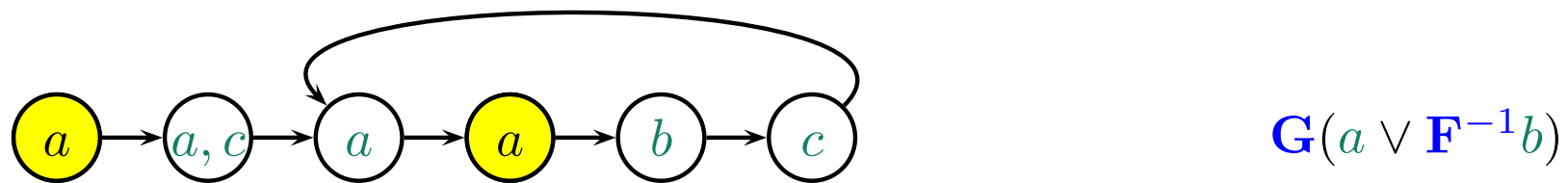
Satisfiability for $L(\mathbf{F}, \mathbf{F}^{-1})$ is in NP

We keep only **polynomially** many states from an **ultimately periodic witness**: we first unwind the loop; then for each **subformula**, we keep the **first and the last state** satisfying that **subformula**, **if they exist**.



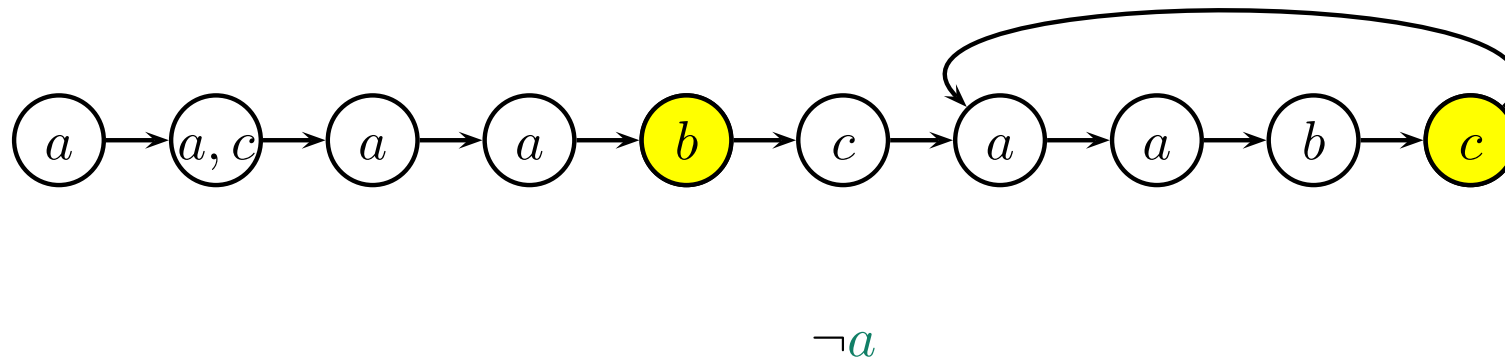
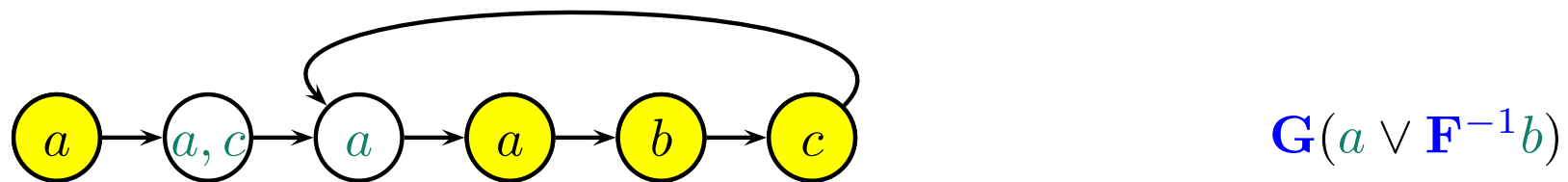
Satisfiability for $L(\mathbf{F}, \mathbf{F}^{-1})$ is in NP

We keep only **polynomially** many states from an **ultimately periodic witness**: we first unwind the loop; then for each **subformula**, we keep the **first and the last state** satisfying that **subformula**, **if they exist**.



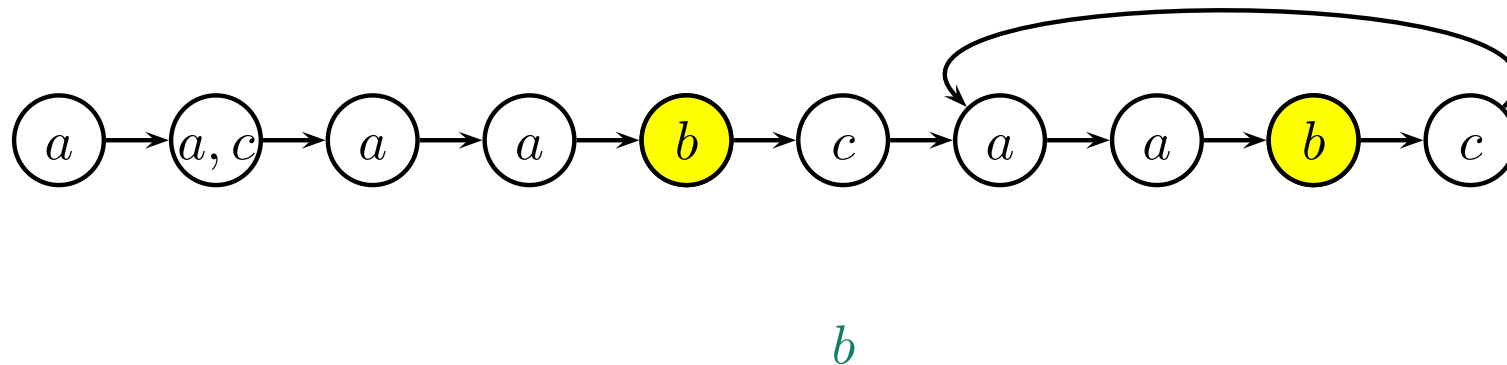
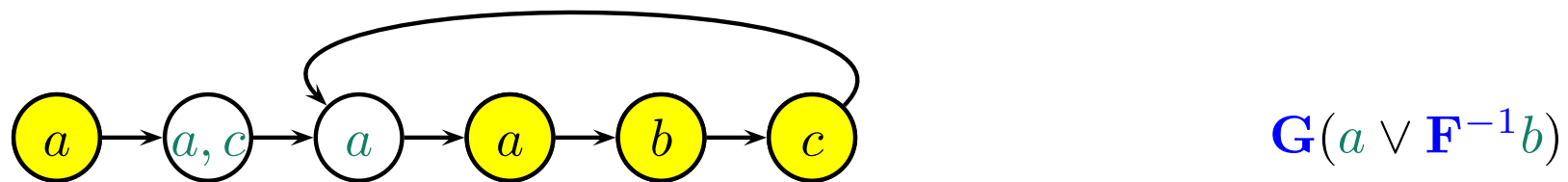
Satisfiability for $L(F, F^{-1})$ is in NP

We keep only **polynomially** many states from an **ultimately periodic witness**: we first unwind the loop; then for each **subformula**, we keep the **first and the last state** satisfying that **subformula**, **if they exist**.



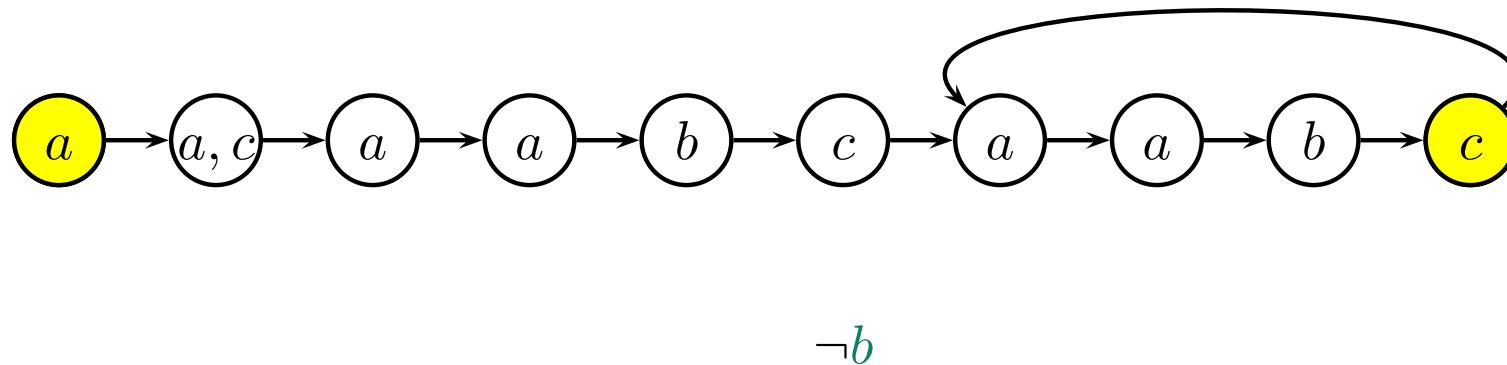
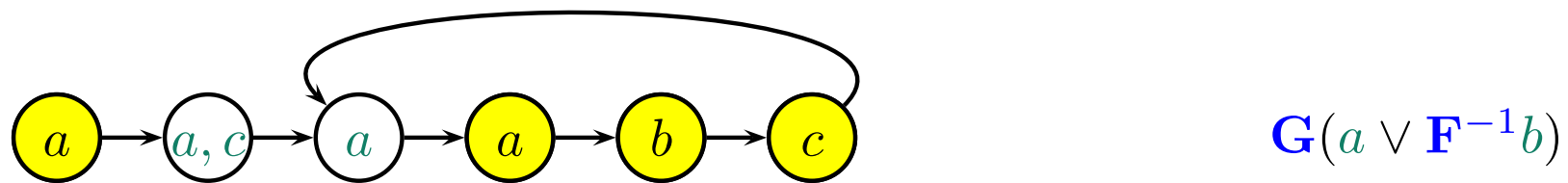
Satisfiability for $L(\mathbf{F}, \mathbf{F}^{-1})$ is in NP

We keep only **polynomially** many states from an **ultimately periodic witness**: we first unwind the loop; then for each **subformula**, we keep the **first and the last state** satisfying that **subformula**, **if they exist**.



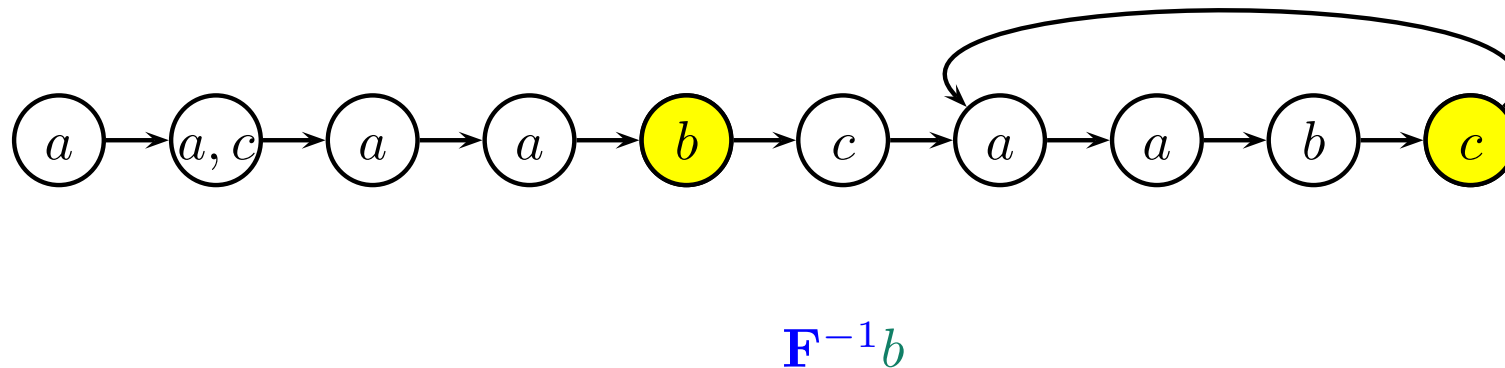
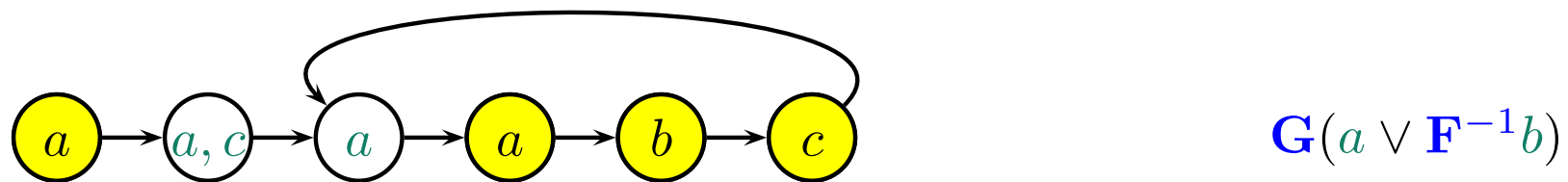
Satisfiability for $L(\mathbf{F}, \mathbf{F}^{-1})$ is in NP

We keep only **polynomially** many states from an **ultimately periodic witness**: we first unwind the loop; then for each **subformula**, we keep the **first and the last state** satisfying that **subformula**, **if they exist**.



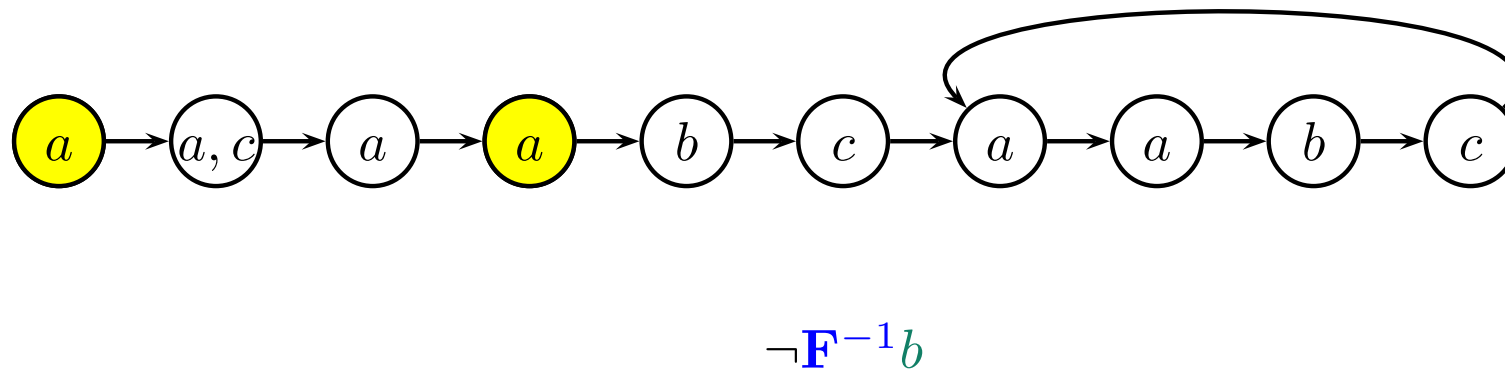
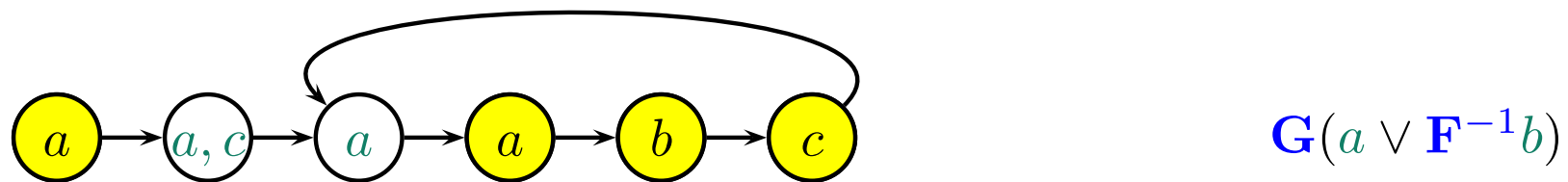
Satisfiability for $L(\mathbf{F}, \mathbf{F}^{-1})$ is in NP

We keep only **polynomially** many states from an **ultimately periodic witness**: we first unwind the loop; then for each **subformula**, we keep the **first and the last state** satisfying that **subformula**, **if they exist**.



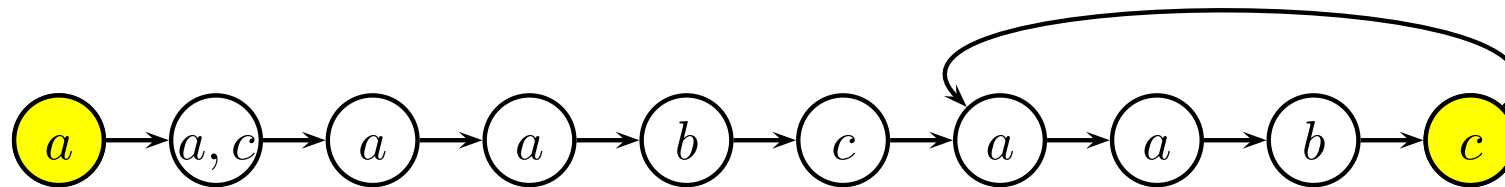
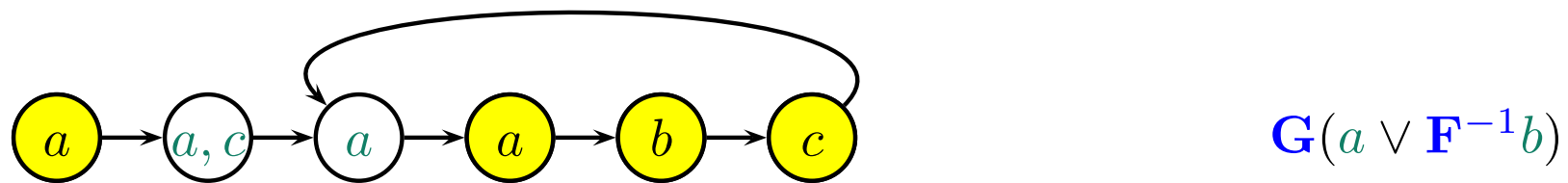
Satisfiability for $L(\mathbf{F}, \mathbf{F}^{-1})$ is in NP

We keep only **polynomially** many states from an **ultimately periodic witness**: we first unwind the loop; then for each **subformula**, we keep the **first and the last state** satisfying that **subformula**, **if they exist**.



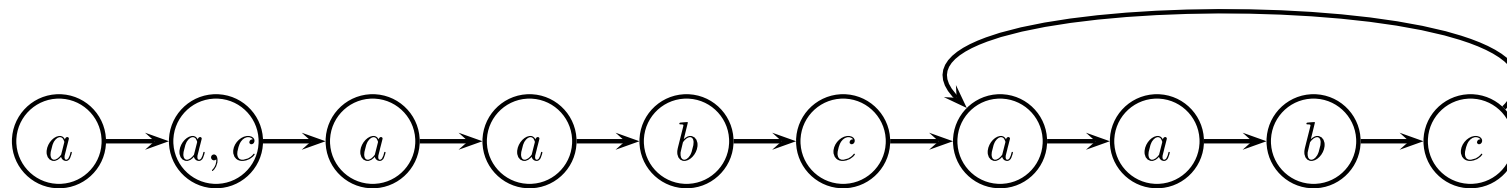
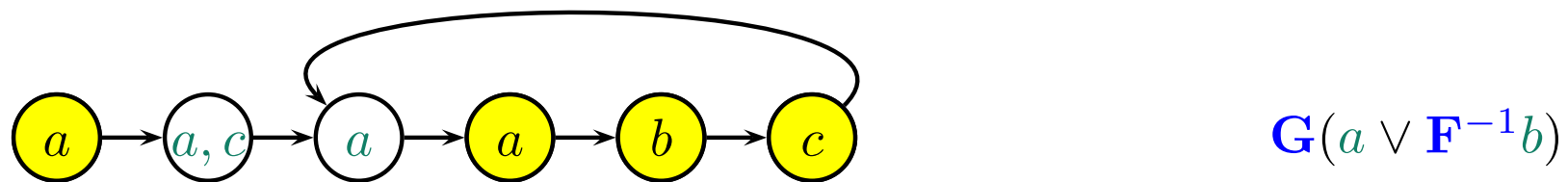
Satisfiability for $L(\mathbf{F}, \mathbf{F}^{-1})$ is in NP

We keep only **polynomially** many states from an **ultimately periodic witness**: we first unwind the loop; then for each **subformula**, we keep the **first and the last state** satisfying that **subformula**, **if they exist**.



Satisfiability for $L(\mathbf{F}, \mathbf{F}^{-1})$ is in NP

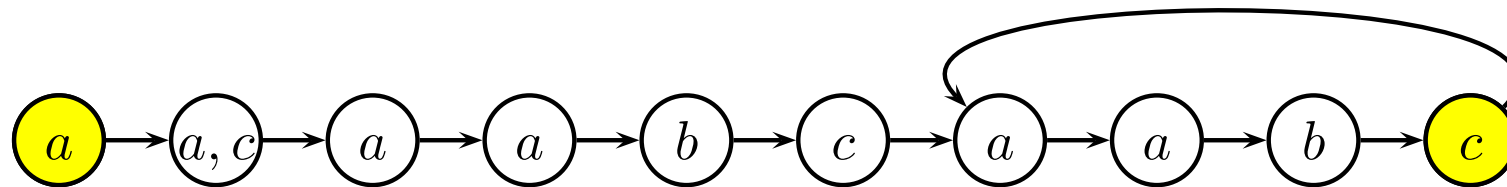
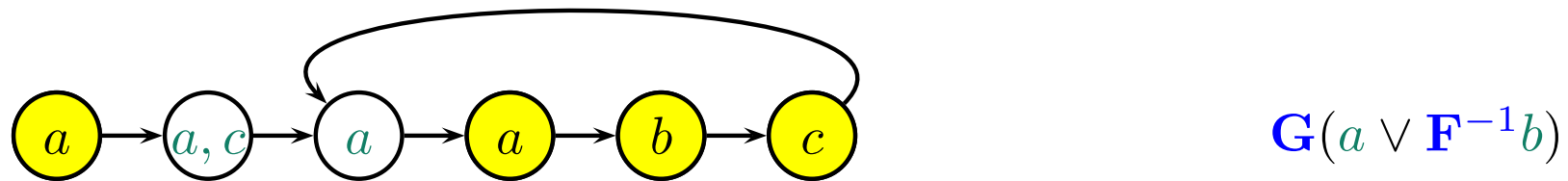
We keep only **polynomially** many states from an **ultimately periodic witness**: we first unwind the loop; then for each **subformula**, we keep the **first and the last state** satisfying that **subformula**, **if they exist**.



$$\neg(a \vee \mathbf{F}^{-1}b)$$

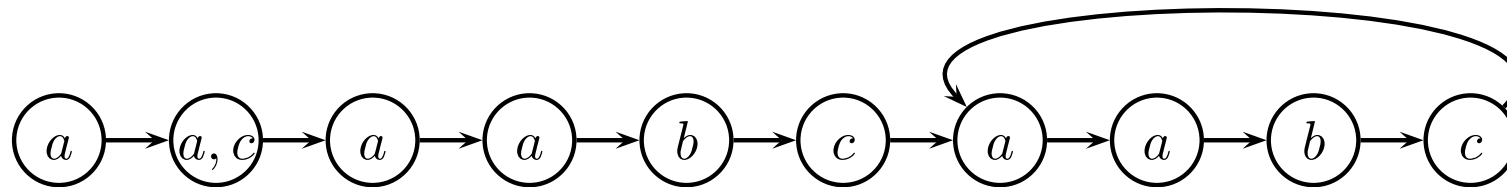
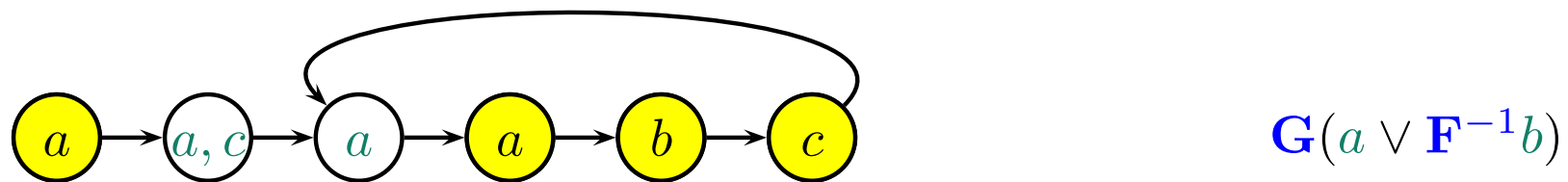
Satisfiability for $L(\mathbf{F}, \mathbf{F}^{-1})$ is in NP

We keep only **polynomially** many states from an **ultimately periodic witness**: we first unwind the loop; then for each **subformula**, we keep the **first and the last state** satisfying that **subformula**, **if they exist**.



Satisfiability for $L(F, F^{-1})$ is in NP

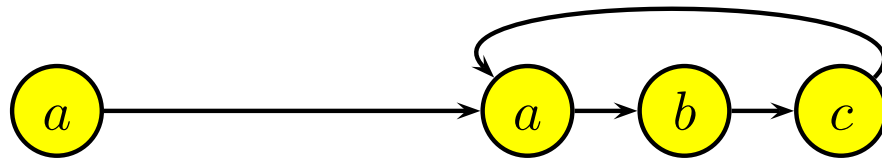
We keep only **polynomially** many states from an **ultimately periodic witness**: we first unwind the loop; then for each **subformula**, we keep the **first and the last state** satisfying that **subformula**, **if they exist**.



$$\neg G(a \vee F^{-1}b)$$

Satisfiability for $L(\mathbf{F}, \mathbf{F}^{-1})$ is in NP

We keep only **polynomially** many states from an **ultimately periodic witness**: we first unwind the loop; then for each **subformula**, we keep the **first and the last state** satisfying that **subformula**, **if they exist**.

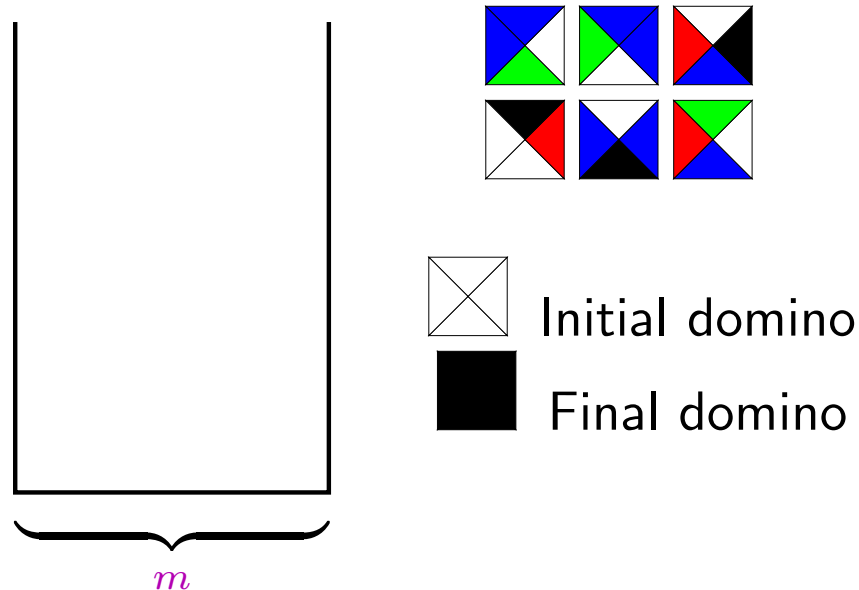


$$\mathbf{G}(a \vee \mathbf{F}^{-1}b)$$

PSPACE-hardness proofs

Reductions from the following tiling problem [Har83]:

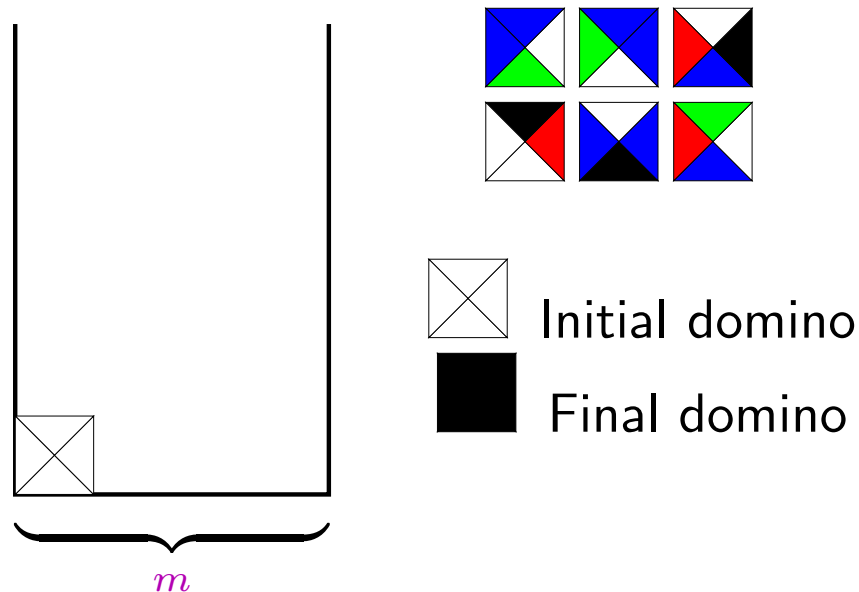
Given a set of colors, a set of domino-types, an integer m (in unary), and an initial and a final domino-type, can we tile a grid of length m so that the bottom leftmost domino is the initial one and the top rightmost domino is the final one?



PSPACE-hardness proofs

Reductions from the following tiling problem [Har83]:

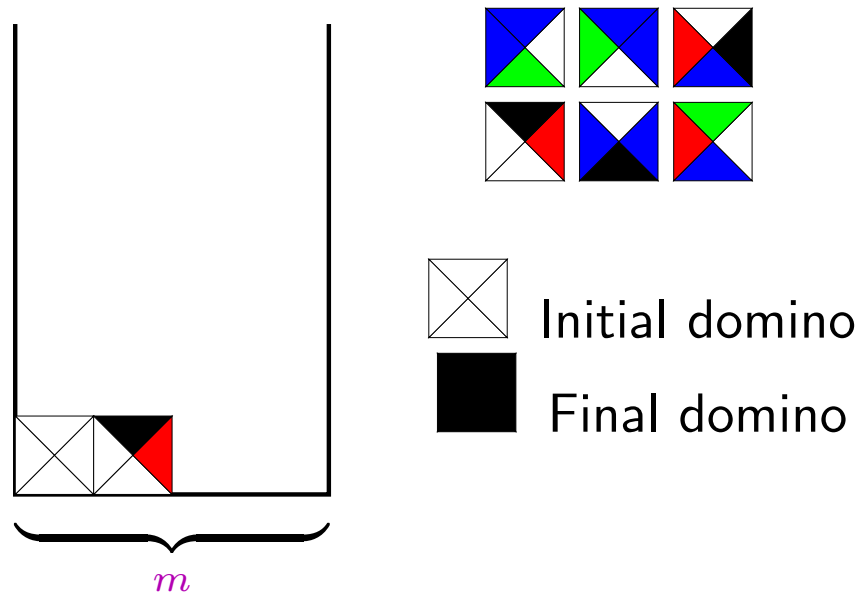
Given a set of colors, a set of domino-types, an integer m (in unary), and an initial and a final domino-type, can we tile a grid of length m so that the bottom leftmost domino is the initial one and the top rightmost domino is the final one?



PSPACE-hardness proofs

Reductions from the following tiling problem [Har83]:

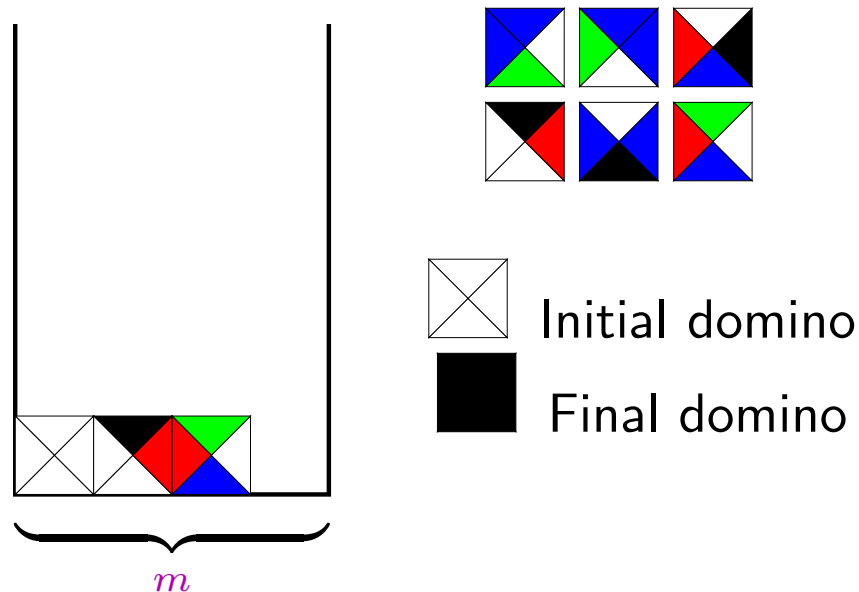
Given a set of colors, a set of domino-types, an integer m (in unary), and an initial and a final domino-type, can we tile a grid of length m so that the bottom leftmost domino is the initial one and the top rightmost domino is the final one?



PSPACE-hardness proofs

Reductions from the following tiling problem [Har83]:

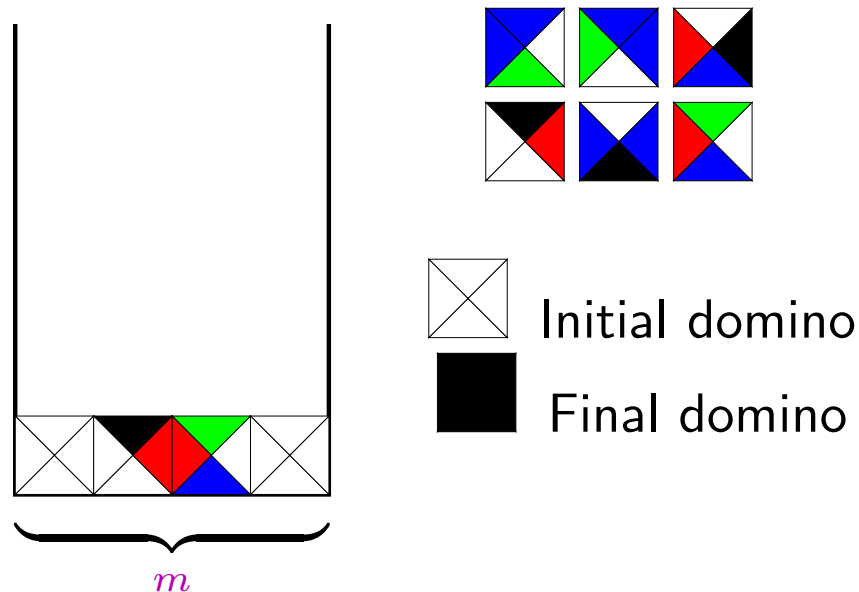
Given a set of colors, a set of domino-types, an integer m (in unary), and an initial and a final domino-type, can we tile a grid of length m so that the bottom leftmost domino is the initial one and the top rightmost domino is the final one?



PSPACE-hardness proofs

Reductions from the following tiling problem [Har83]:

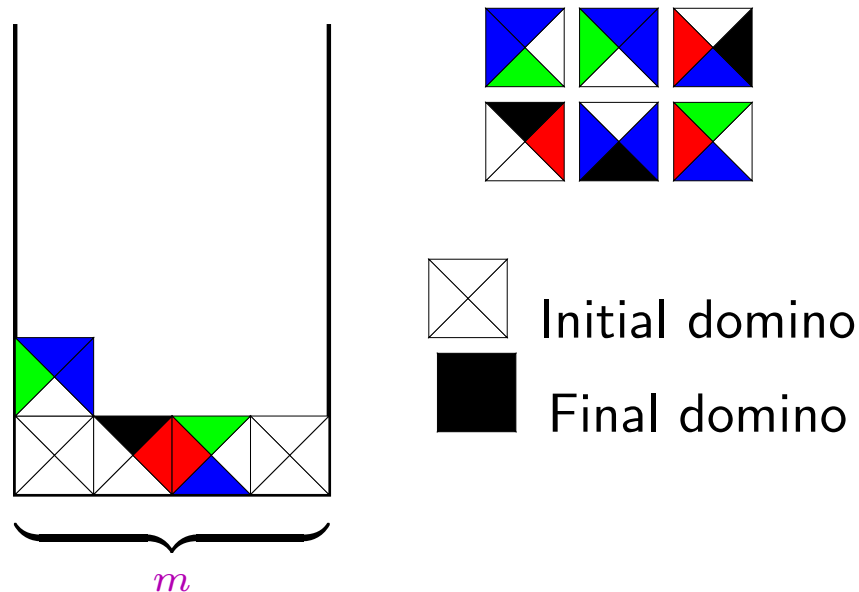
Given a set of colors, a set of domino-types, an integer m (in unary), and an initial and a final domino-type, can we tile a grid of length m so that the bottom leftmost domino is the initial one and the top rightmost domino is the final one?



PSPACE-hardness proofs

Reductions from the following tiling problem [Har83]:

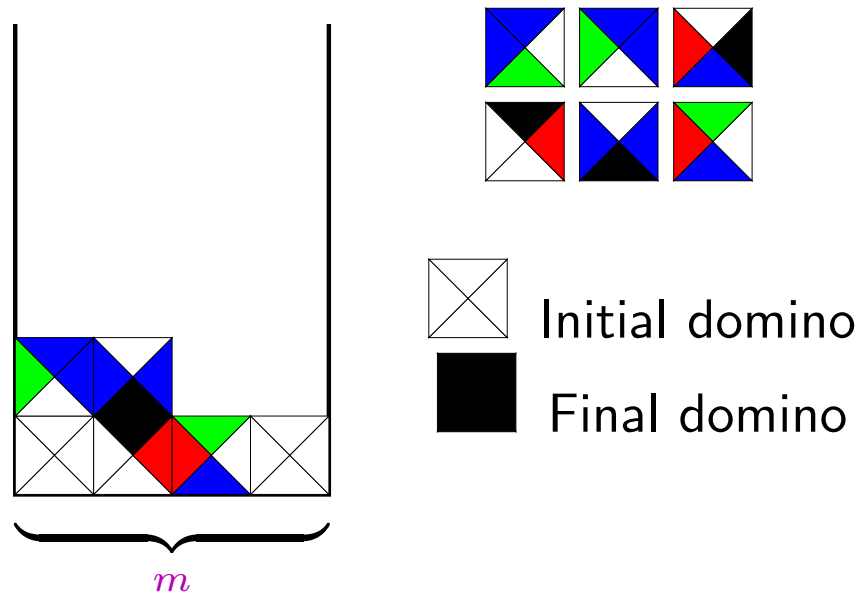
Given a set of colors, a set of domino-types, an integer m (in unary), and an initial and a final domino-type, can we tile a grid of length m so that the bottom leftmost domino is the initial one and the top rightmost domino is the final one?



PSPACE-hardness proofs

Reductions from the following tiling problem [Har83]:

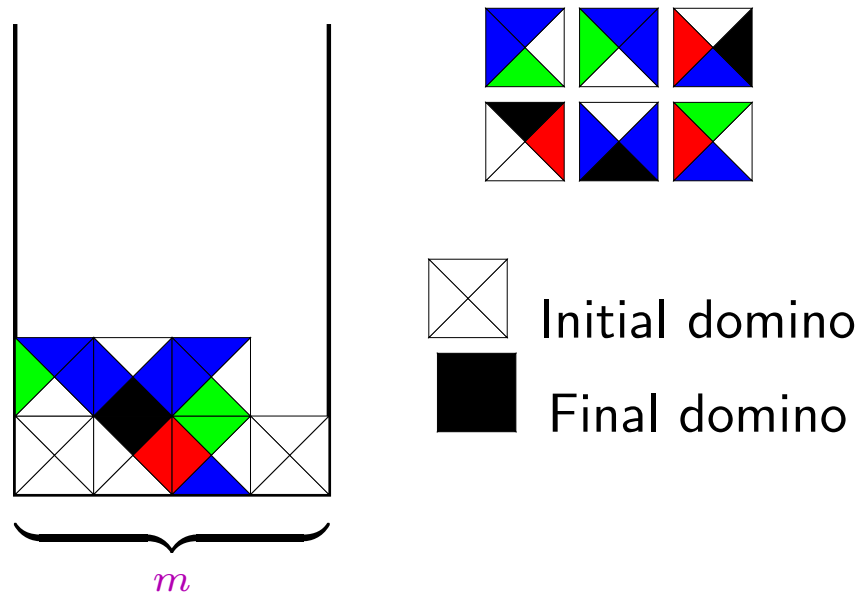
Given a set of colors, a set of domino-types, an integer m (in unary), and an initial and a final domino-type, can we tile a grid of length m so that the bottom leftmost domino is the initial one and the top rightmost domino is the final one?



PSPACE-hardness proofs

Reductions from the following tiling problem [Har83]:

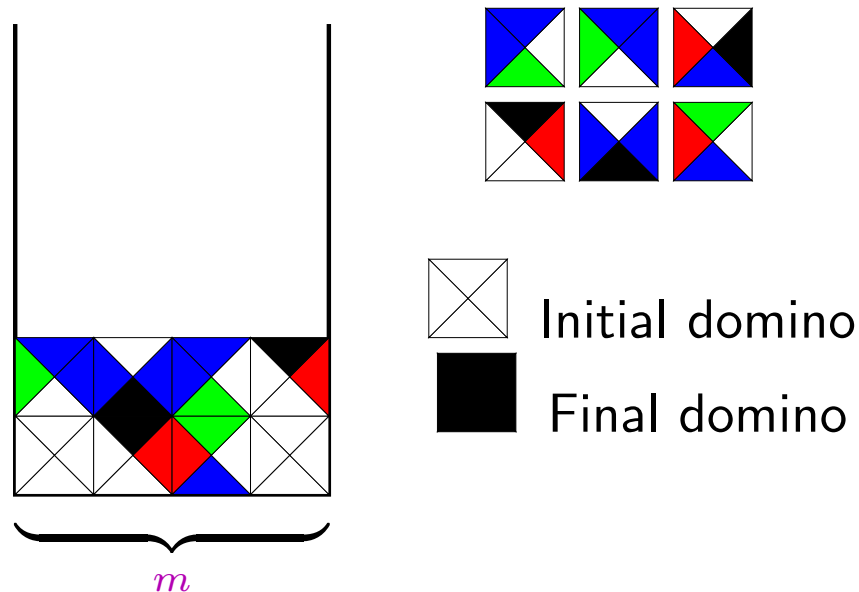
Given a set of colors, a set of domino-types, an integer m (in unary), and an initial and a final domino-type, can we tile a grid of length m so that the bottom leftmost domino is the initial one and the top rightmost domino is the final one?



PSPACE-hardness proofs

Reductions from the following tiling problem [Har83]:

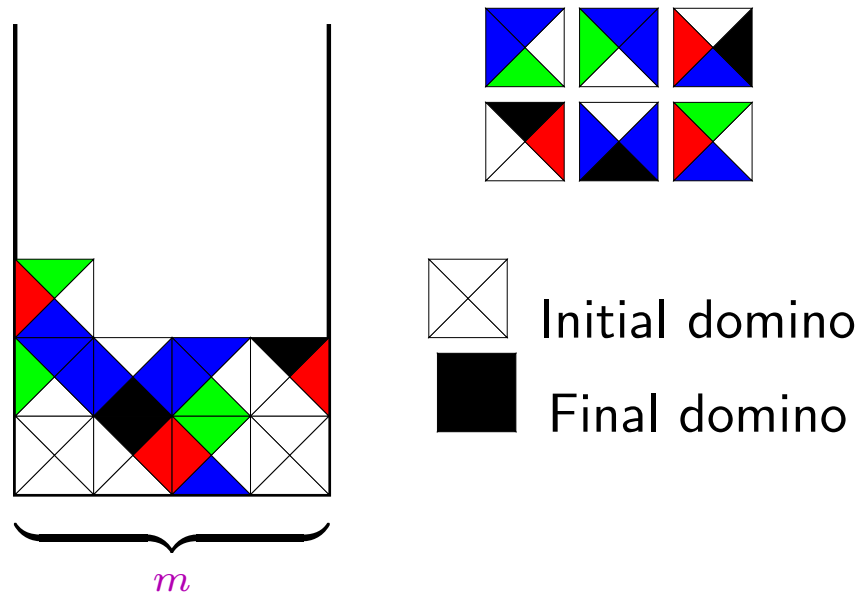
Given a set of colors, a set of domino-types, an integer m (in unary), and an initial and a final domino-type, can we tile a grid of length m so that the bottom leftmost domino is the initial one and the top rightmost domino is the final one?



PSPACE-hardness proofs

Reductions from the following tiling problem [Har83]:

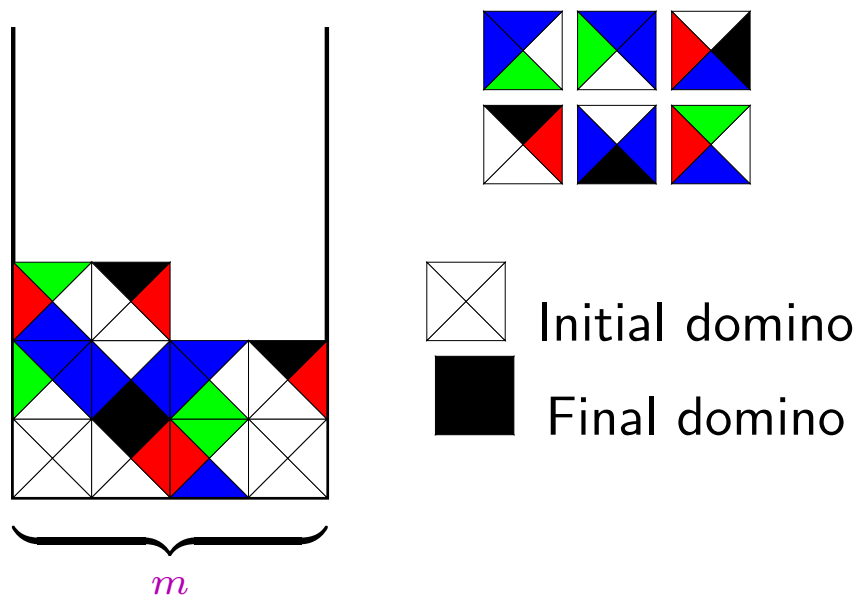
Given a set of colors, a set of domino-types, an integer m (in unary), and an initial and a final domino-type, can we tile a grid of length m so that the bottom leftmost domino is the initial one and the top rightmost domino is the final one?



PSPACE-hardness proofs

Reductions from the following tiling problem [Har83]:

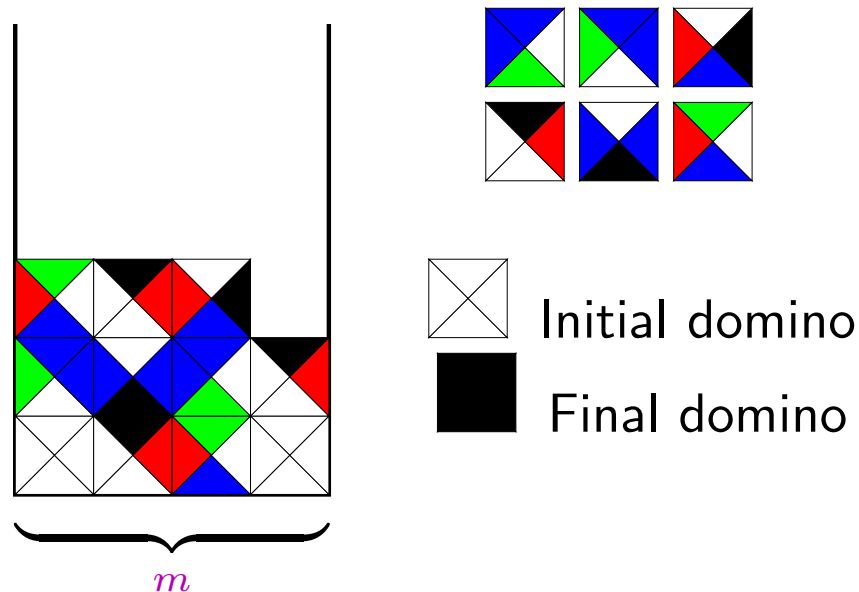
Given a set of colors, a set of domino-types, an integer m (in unary), and an initial and a final domino-type, can we tile a grid of length m so that the bottom leftmost domino is the initial one and the top rightmost domino is the final one?



PSPACE-hardness proofs

Reductions from the following tiling problem [Har83]:

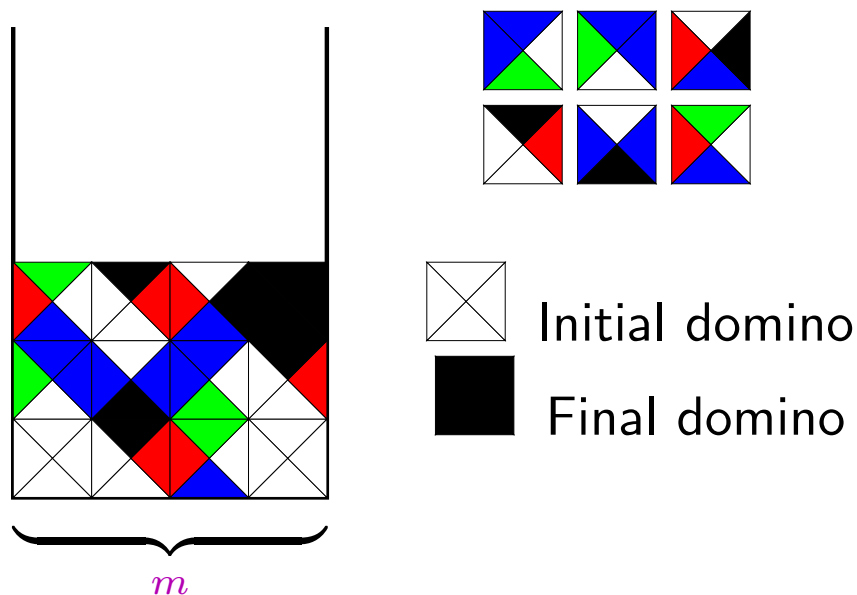
Given a set of colors, a set of domino-types, an integer m (in unary), and an initial and a final domino-type, can we tile a grid of length m so that the bottom leftmost domino is the initial one and the top rightmost domino is the final one?



PSPACE-hardness proofs

Reductions from the following tiling problem [Har83]:

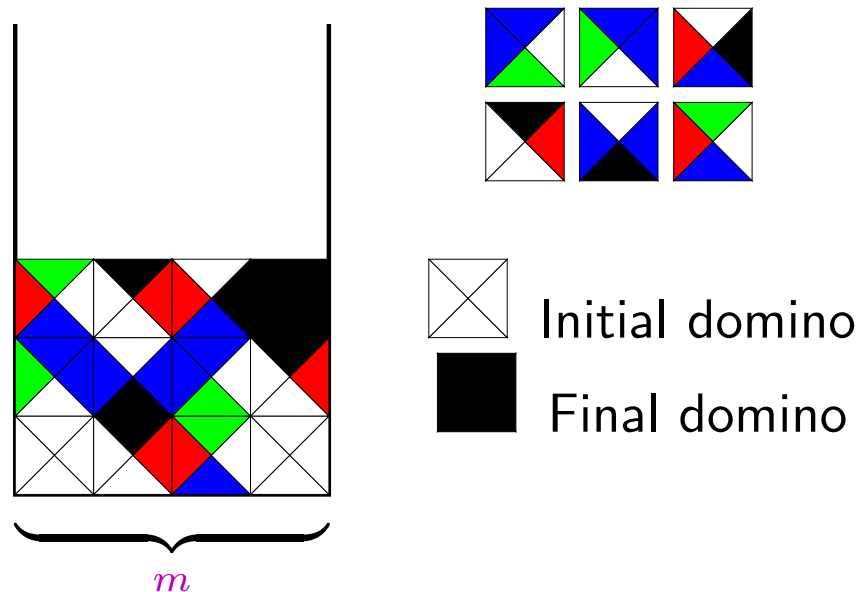
Given a set of colors, a set of domino-types, an integer m (in unary), and an initial and a final domino-type, can we tile a grid of length m so that the bottom leftmost domino is the initial one and the top rightmost domino is the final one?



PSPACE-hardness proofs

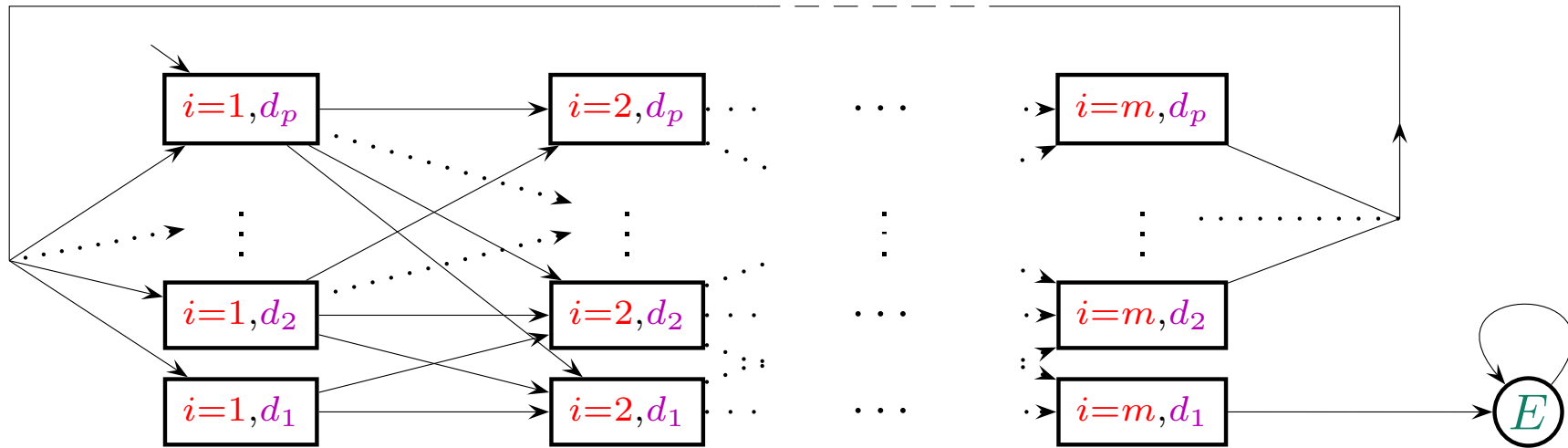
Reductions from the following tiling problem [Har83]:

Given a set of colors, a set of domino-types, an integer m (in unary), and an initial and a final domino-type, can we tile a grid of length m so that the bottom leftmost domino is the initial one and the top rightmost domino is the final one?

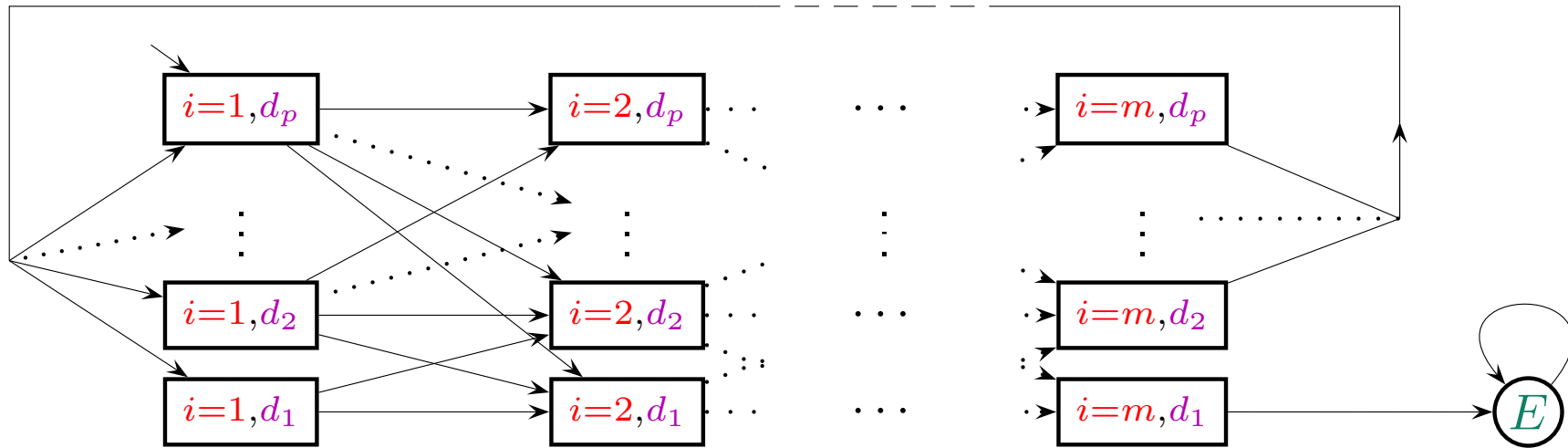


This problem is PSPACE-complete.

Reduction to model checking $L^+(\mathcal{U})$

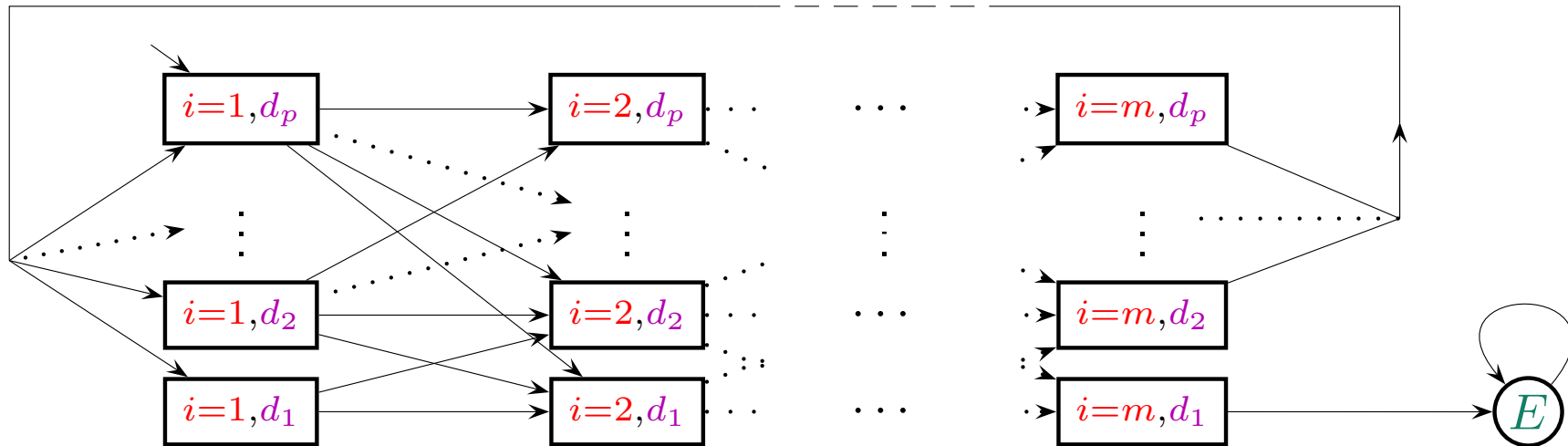


Reduction to model checking $L^+(\mathcal{U})$



We have to express that:

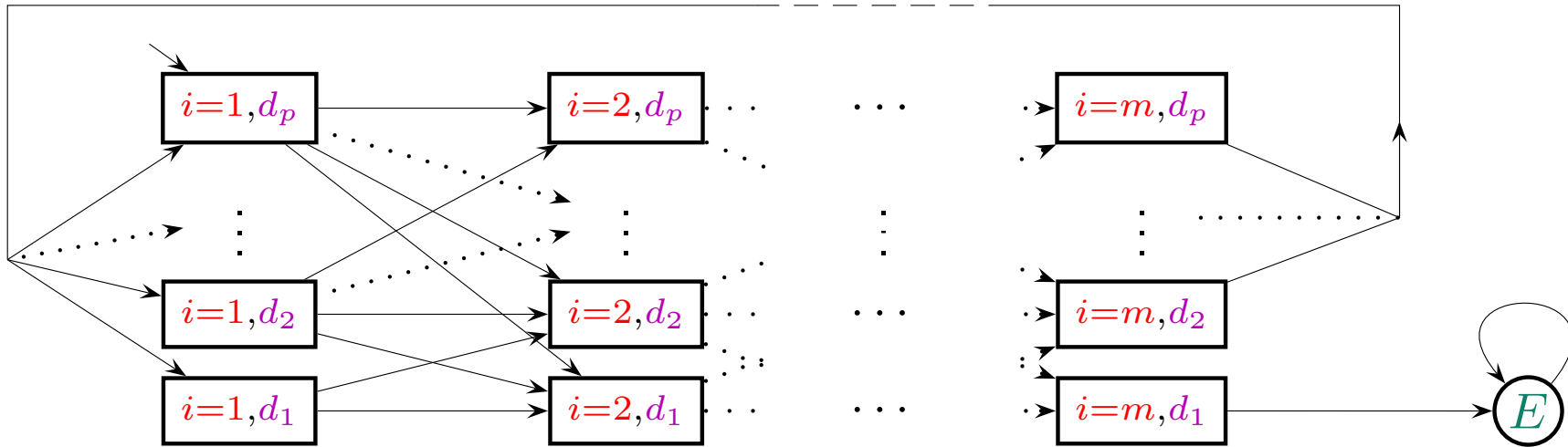
Reduction to model checking $L^+(\mathcal{U})$



We have to express that: The **initial and final conditions** are fulfilled.

$$\top \mathcal{U} E$$

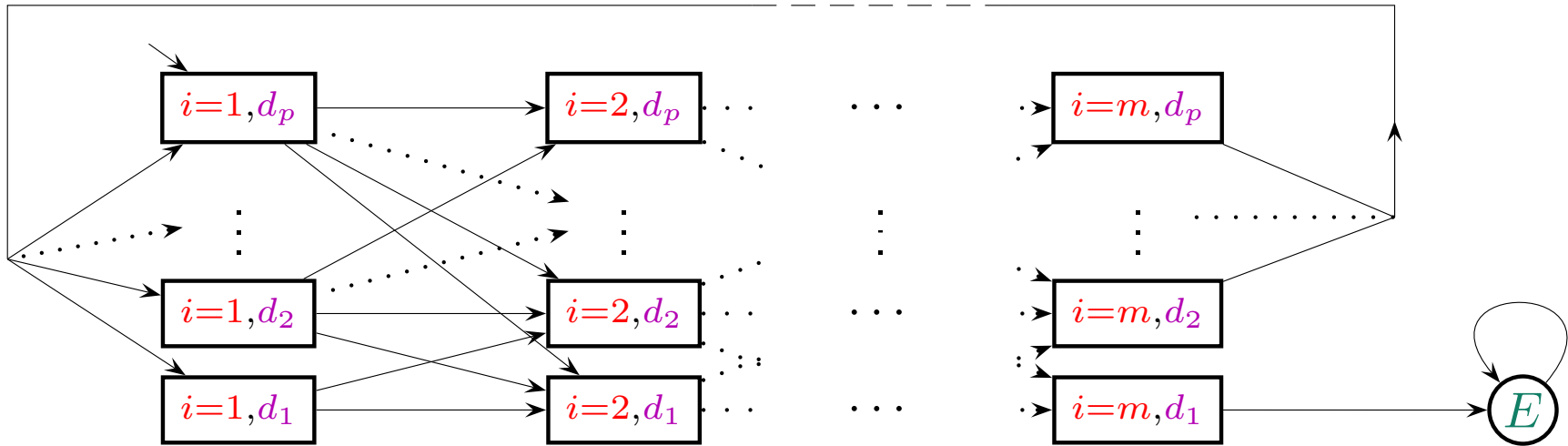
Reduction to model checking $L^+(\mathcal{U})$



We have to express that: The **horizontal tiling condition** is fulfilled.

$$\begin{array}{c} \top \mathcal{U} E \\ \left(\bigwedge_{k=1}^{m-1} \bigwedge_{d \in T} (i = k \wedge d) \Rightarrow (i = k \mathcal{U} (i = k + 1 \wedge \bigvee_{\substack{d' \in T \\ d'^{\text{left}} = d'^{\text{right}}}} d')) \right) \mathcal{U} E \end{array}$$

Reduction to model checking $L^+(\mathcal{U})$



We have to express that: The **vertical tiling condition** is fulfilled.

$\top \mathcal{U} E$

$$\left(\bigwedge_{k=1}^{m-1} \bigwedge_{d \in T} (i = k \wedge d) \Rightarrow (i = k \mathcal{U} (i = k + 1 \wedge \bigvee_{\substack{d' \in T \\ d'^{\text{left}} = d^{\text{right}}}} d')) \right) \mathcal{U} E$$

$$\left(\bigwedge_{k=1}^m \bigwedge_{d \in T} (i = k \wedge d) \Rightarrow \left(i = k \mathcal{U} \left(\neg i = k \wedge (\neg i = k) \mathcal{U} (E \vee (i = k \wedge \bigvee_{\substack{d' \in T \\ d'^{\text{down}} = d^{\text{up}}}} d')) \right) \right) \right) \mathcal{U} E$$

Conclusion

- Some surprising results;
- No obvious pattern for the NP/PSPACE frontier;
- Past is more natural, more succinct, and not more expensive;
- Does not extend to larger logics?

Bibliography

- [EVW97] Kousha Etessami, Moshe Y. Vardi, and Thomas Wilke. First-order logic with two variables and unary temporal logic. In *LICS'97*, pages 228–235, Warsaw, Poland, 1997. 12th Annual IEEE Symposium on Logic in Computer Science, IEEE Comp. Soc. Press.
- [Gab89] Dov M. Gabbay. The declarative past and imperative future: Executable temporal logic for interactive systems. In Behnam Banieqbal, Howard Barringer, and Amir Pnueli, editors, *Conference on Temporal Logic in Specification*, volume 398 of *Lect. Notes in Comp. Sci.*, pages 409–448. Springer, 1989.
- [GPSS80] Dov M. Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. On the temporal analysis of fairness. In *POPL'80*, pages 163–173, Las Vegas, Nevada, 1980. 7th ACM Symposium on Principles of Programming Languages, ACM Press.
- [Har83] David Harel. Recurring dominoes: Making the highly undecidable highly understandable. In Marek Karpinski, editor, *FCT'83*, volume 158 of *Lect. Notes in Comp. Sci.*, pages 177–194, Borgholm, Sweden, 1983. International Conference on Fundamentals of Computation Theory, Springer.
- [Kam68] Hans W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, UCLA, Los Angeles, CA, USA, 1968.
- [LMS02] François Laroussinie, Nicolas Markey, and Philippe Schnoebelen. Temporal logic with forgettable past. In *LICS'02*, pages 383–392, Copenhagen, Denmark, 2002. 17th Annual IEEE Symposium on Logic in Computer Science, IEEE Comp. Soc. Press.

Bibliography

- [Mal90] Oded Maler. *Finite Automata: Infinite Behavior, Learnability and Decomposition*. PhD thesis, The Weizmann Institute of Science, Israel, 1990.
- [SC85] A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.
- [VW86] Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In *LICS'86*, pages 332–344, Cambridge, Massachusetts, 1986. 1st Annual IEEE Symposium on Logic in Computer Science, IEEE Comp. Soc. Press.
- [Wil99] Thomas Wilke. Classifying discrete temporal properties. In Christoph Meinel and Sophie Tison, editors, *STACS'99*, volume 1563 of *Lect. Notes in Comp. Sci.*, Trier, Germany, 1999. 16th Annual Symposium on Theoretical Aspects of Computer Science, Springer.

Proof of the result of [EVW97]

Claim: Let ϕ_n be a PLTL formula expressing the following property:

“any two future positions that agree on p_1, \dots, p_n also agree on p_0 .”

We denote $L_n = \{u \in \{p_0, \dots, p_n\} \mid u \models \phi_n\}$. From [VW86], we know that L_n is recognized by a Generalized Büchi Automaton \mathcal{B} with $2^{O(|\phi_n|)}$ states.

Let $\{a_0, \dots, a_{2^n-1}\}$ be a sequence containing all subsets of $\{p_1, \dots, p_n\}$.

For $K \subseteq \{0, \dots, 2^n - 1\}$, $w_K = b_0 \cdots b_{2^n-1}$ with
$$\begin{cases} b_i = a_i & \text{if } i \in K \\ b_i = a_i \cup \{p_0\} & \text{otherwise} \end{cases}$$

There are 2^{2^n} such words.

Assume $K \neq K'$. Then $w_K^\omega \models \phi_n$ and $w_{K'} w_K^\omega \not\models \phi_n$. The executions of \mathcal{B} on w_K and $w_{K'}$ cannot lead to the same state. The automaton thus needs at least 2^{2^n} states...