Eventually safe languages

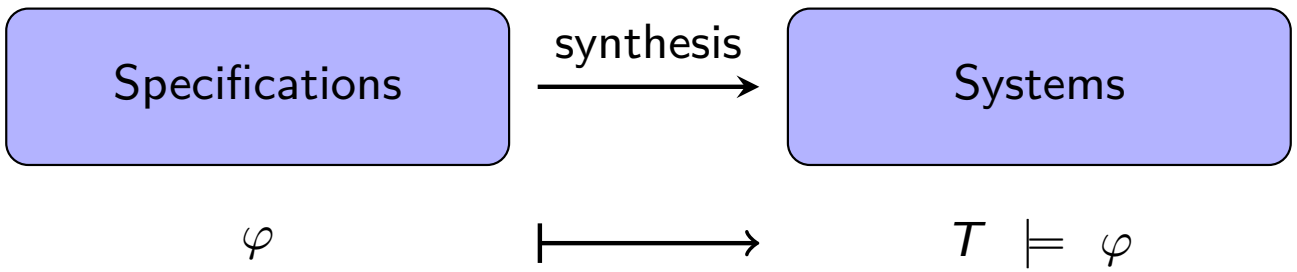
Simon Iosti [1] Denis Kuperberg [2]

[1]Verimag, Université Grenoble-Alpes

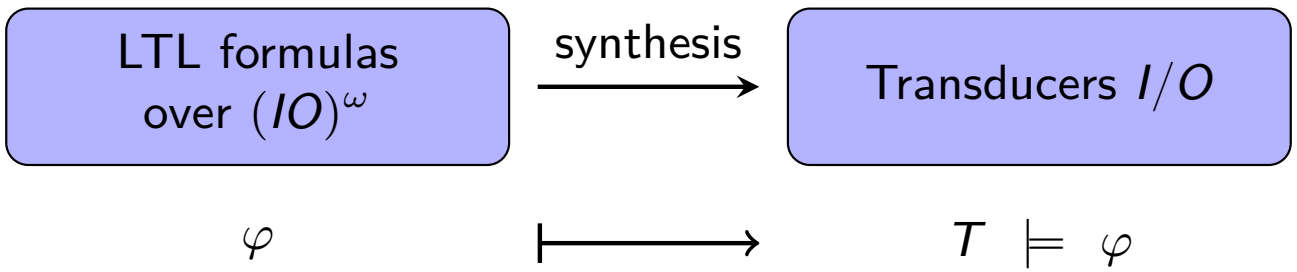
[2]CNRS, LIP, ENS Lyon

Plan

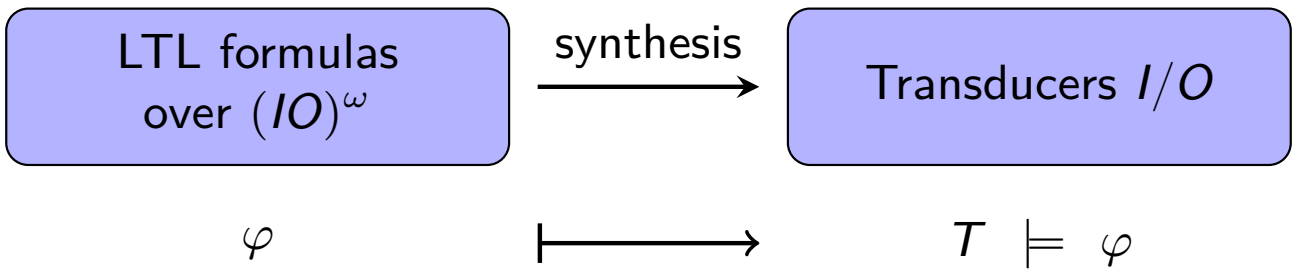
Synthesis problem (Church)



Synthesis problem (Church)

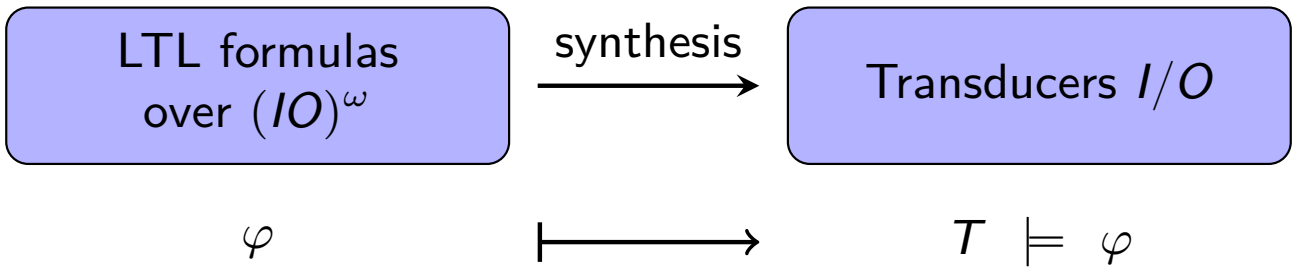


Synthesis problem (Church)

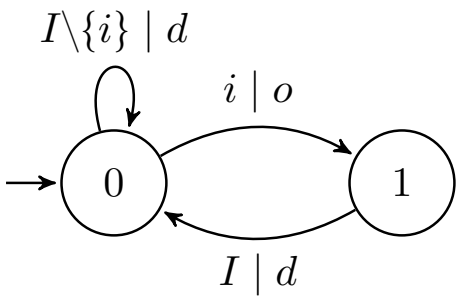


$$\mathbf{F}(i \wedge \mathbf{X}o \wedge \mathbf{XXX}d) \vee \neg\mathbf{F}i$$

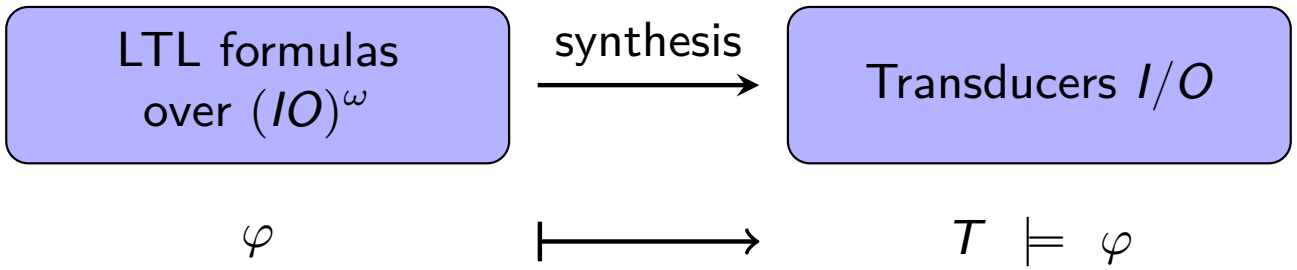
Synthesis problem (Church)



$$\mathbf{F}(i \wedge \mathbf{X}o \wedge \mathbf{XXX}d) \vee \neg\mathbf{F}i$$

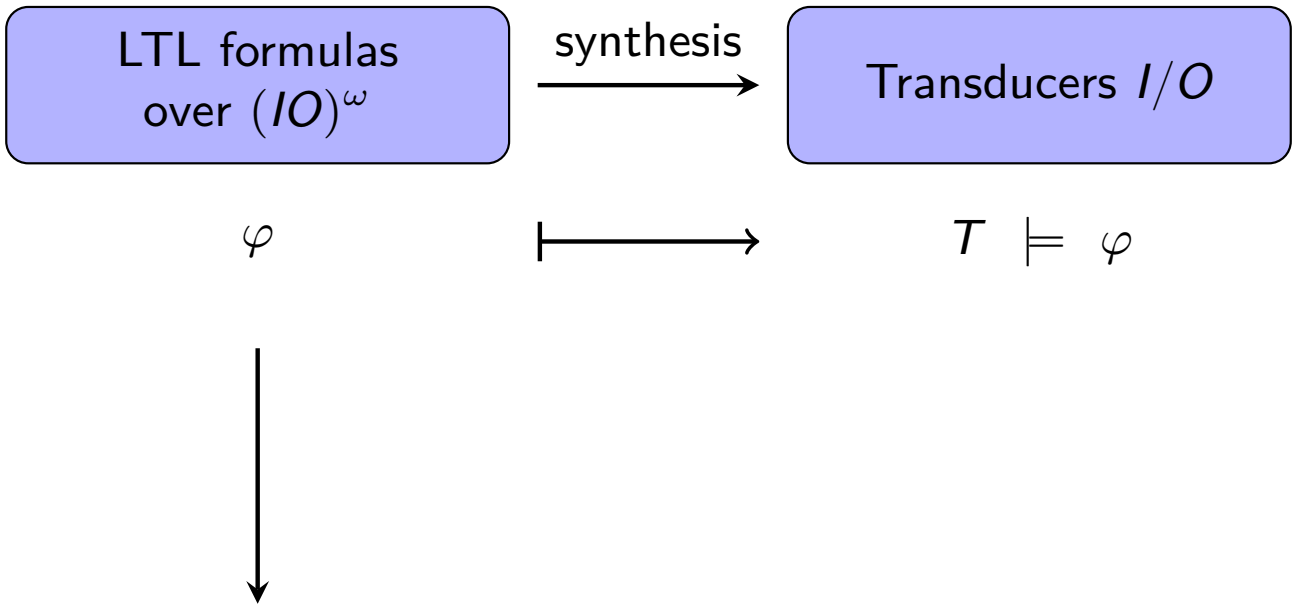


Synthesis problem (Church)



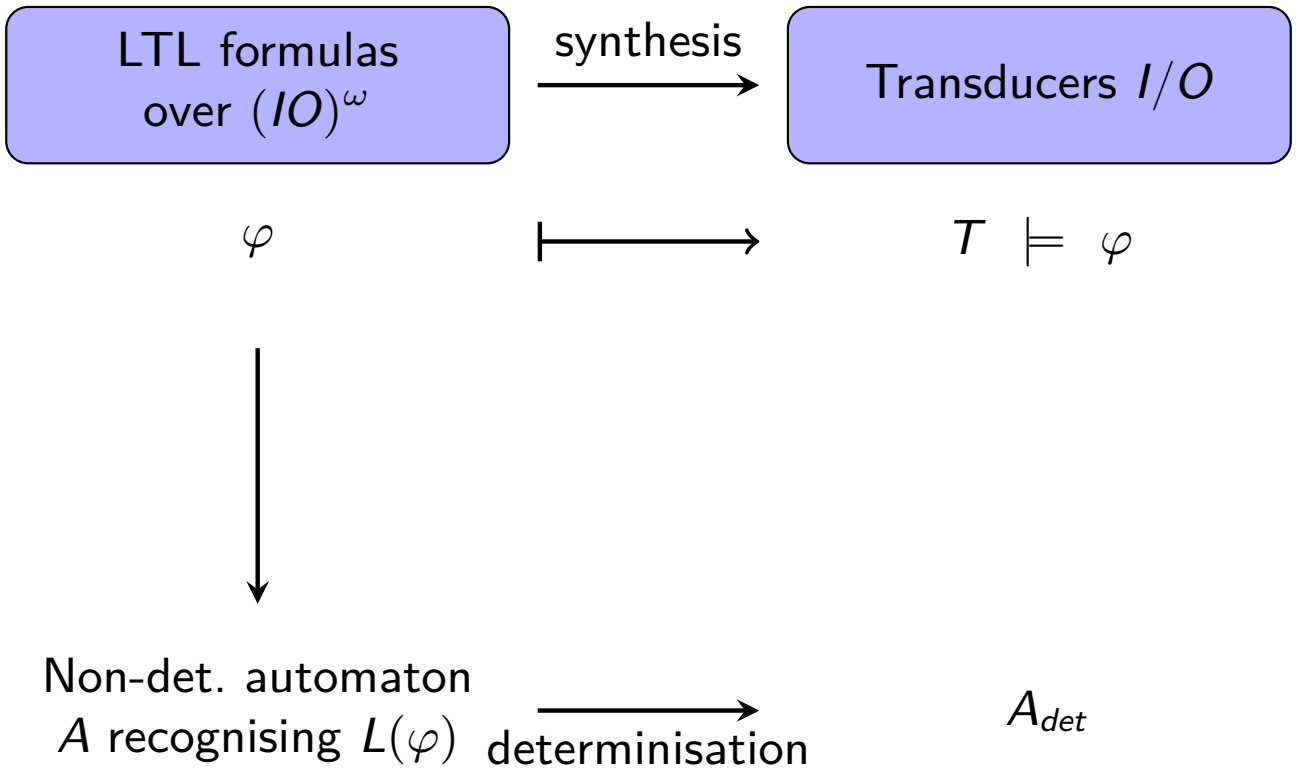
Standard solution (Büchi, Landweber)

Synthesis problem (Church)



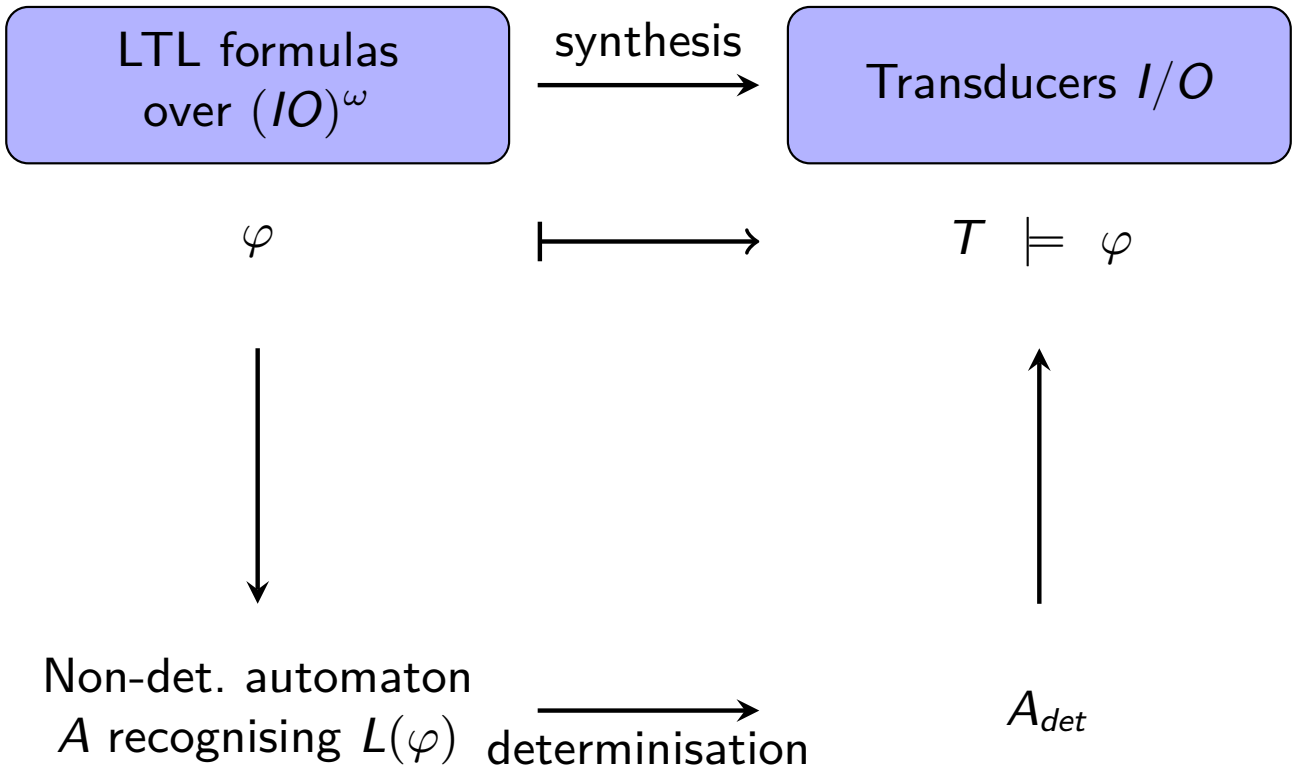
Standard solution (Büchi, Landweber)

Synthesis problem (Church)



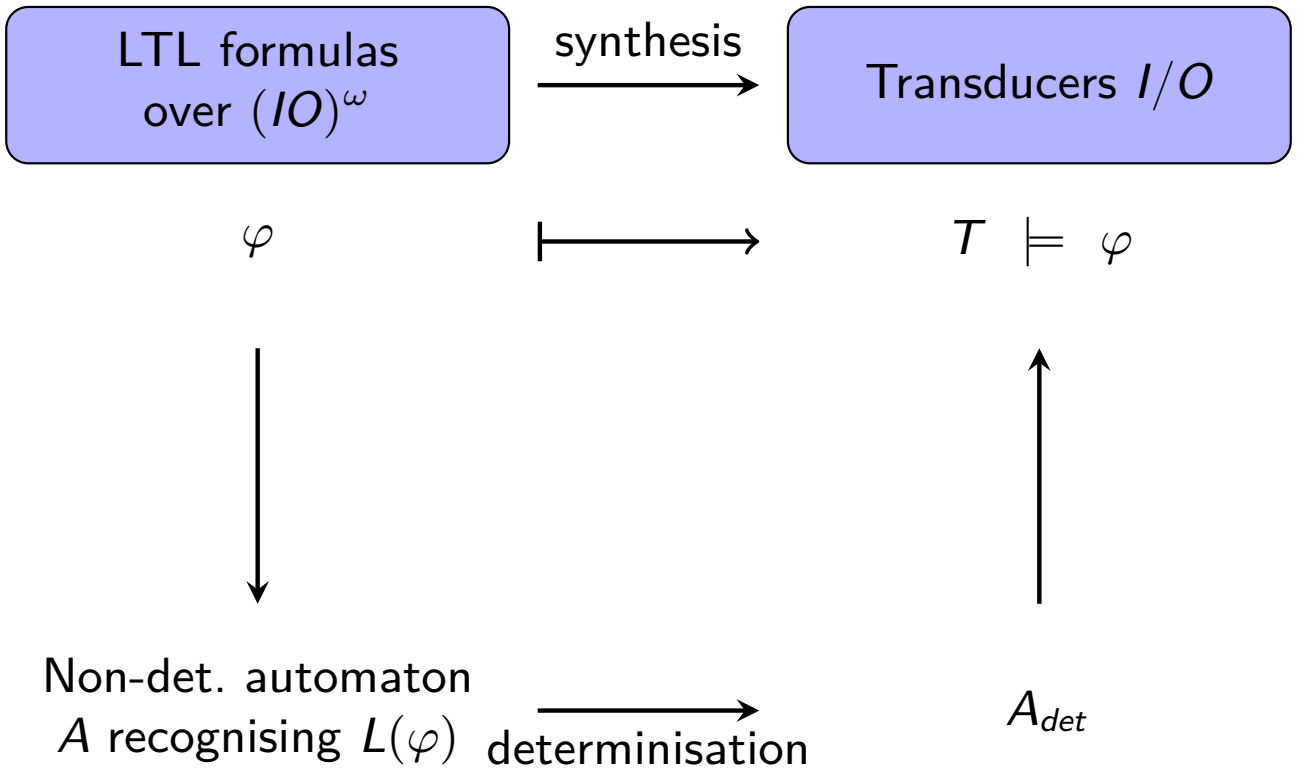
Standard solution (Büchi, Landweber)

Synthesis problem (Church)



Standard solution (Büchi, Landweber)

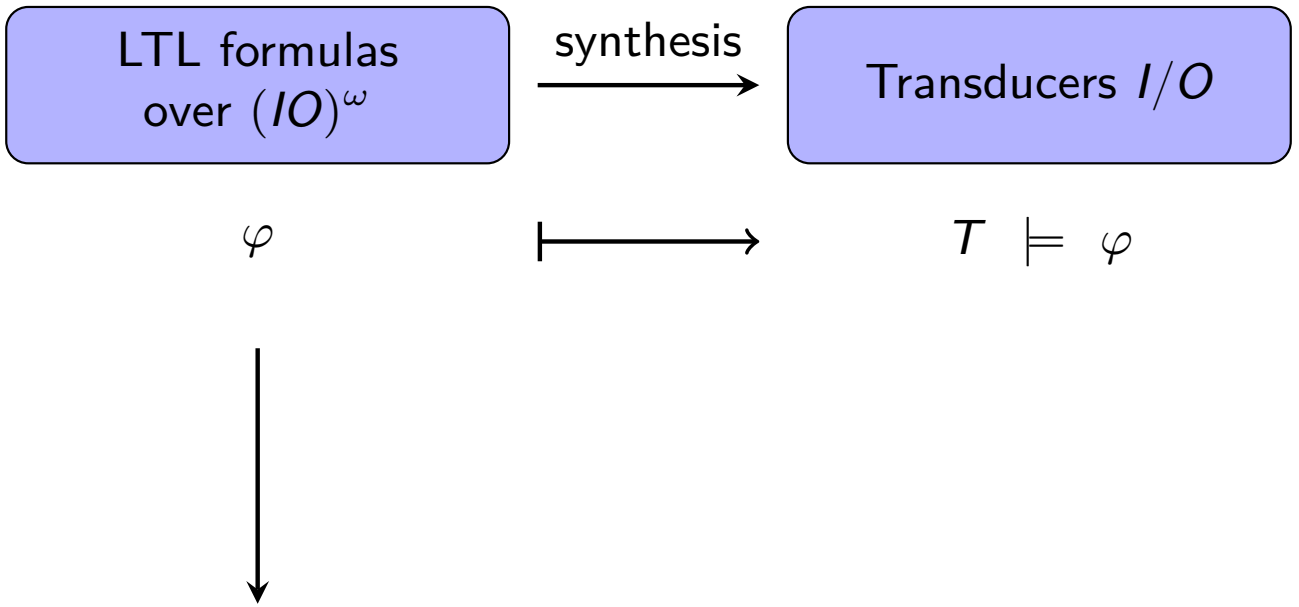
Synthesis problem (Church)



Standard solution (Büchi, Landweber)

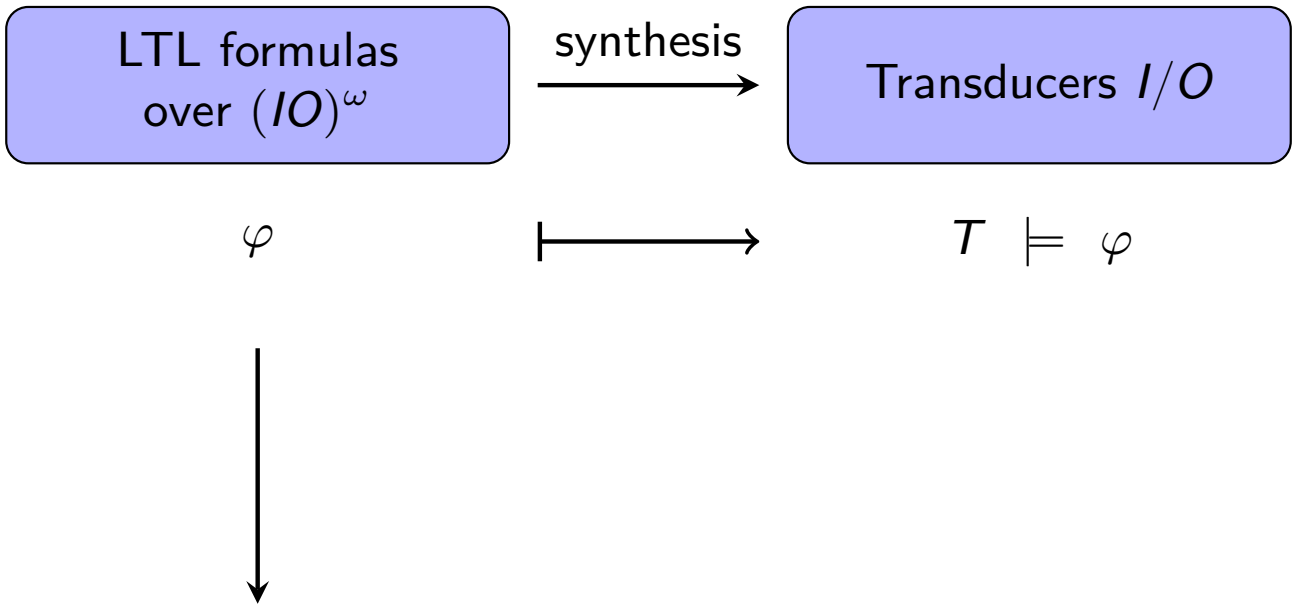
Problem : this is doubly exponential in the size of φ.

Synthesis problem (Church)



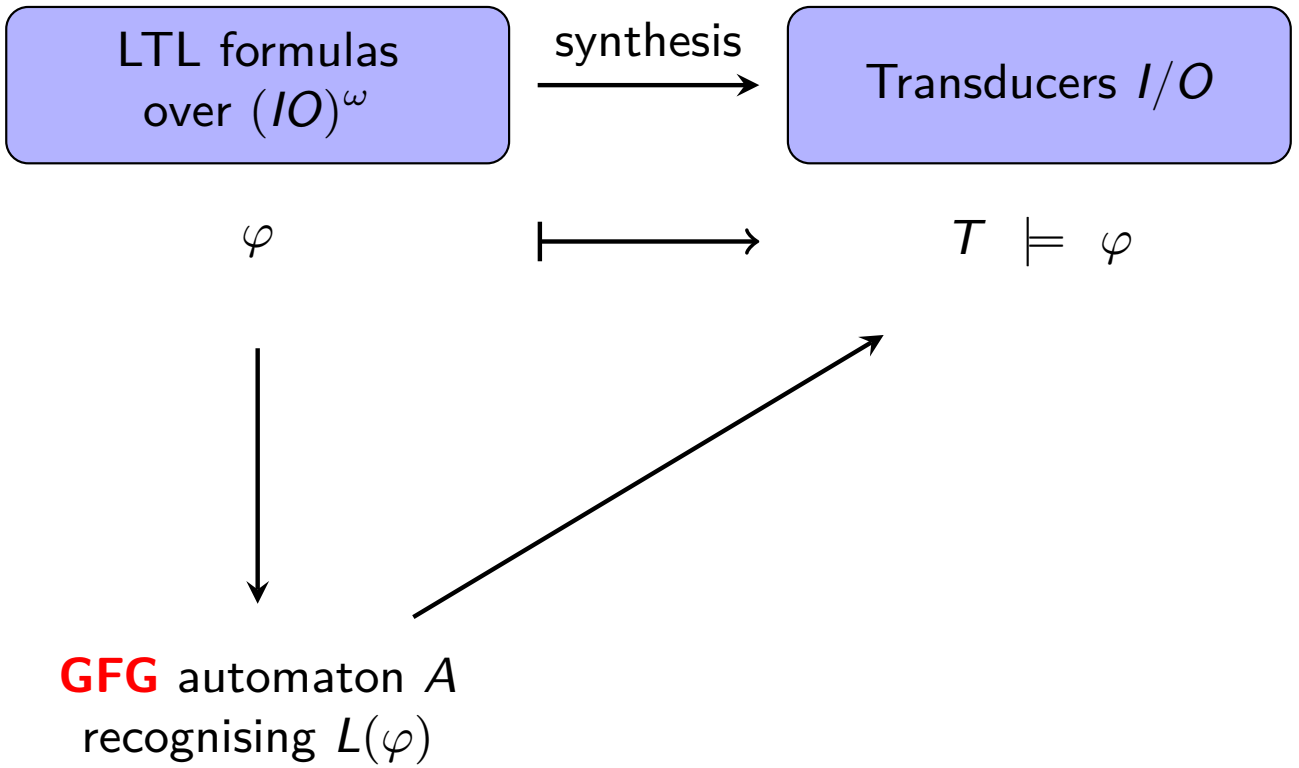
Alternative solution ?

Synthesis problem (Church)



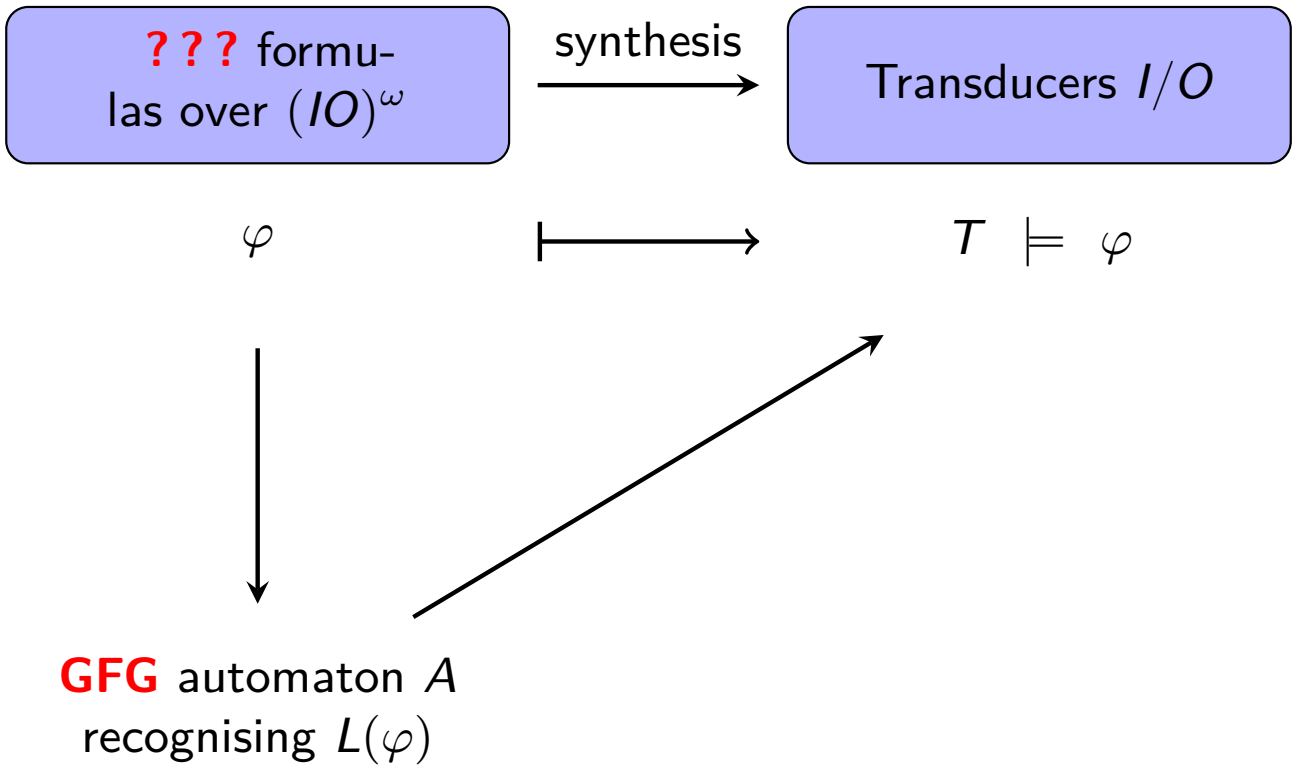
Alternative solution ?

Synthesis problem (Church)



Alternative solution ?

Synthesis problem (Church)



Alternative solution ?

Plan

Plan

Let $A = (\Sigma, Q, \Delta, Q_{init}, Acc)$ be a (non-deterministic) automaton for ω-words over Σ, with any kind of acceptance condition.

Let $A = (\Sigma, Q, \Delta, Q_{init}, Acc)$ be a (non-deterministic) automaton for ω-words over Σ, with any kind of acceptance condition.

Definition (Good-for-Games)

A is said to be **Good-for-Games** (GFG) if there is a strategy σ that resolves the nondeterminism by looking only at the part of the word read so far.

Let $A = (\Sigma, Q, \Delta, Q_{init}, Acc)$ be a (non-deterministic) automaton for ω-words over Σ, with any kind of acceptance condition.

Definition (Good-for-Games)

A is said to be **Good-for-Games** (GFG) if there is a strategy $\sigma : \Sigma^* \to Q$ such that :

- for every $w \in \Sigma^*$ and $a \in \Sigma$, $\sigma(wa) \in \Delta(\sigma(w), a)$ (respect structure of A) ;
- for every word $w \in \Sigma^\omega$, the run generated by σ over w is accepting if and only if $w \in L(A)$.

Let $A = (\Sigma, Q, \Delta, Q_{init}, Acc)$ be a (non-deterministic) automaton for ω-words over Σ, with any kind of acceptance condition.

Definition (Good-for-Games)

A is said to be **Good-for-Games** (GFG) if there is a strategy $\sigma : \Sigma^* \to Q$ such that :

- for every $w \in \Sigma^*$ and $a \in \Sigma$, $\sigma(wa) \in \Delta(\sigma(w), a)$ (respect structure of A) ;
- for every word $w \in \Sigma^\omega$, the run generated by σ over w is accepting if and only if $w \in L(A)$.

Run generated by σ over $w = w_1 \dots w_n \dots$:
$\sigma(\epsilon)$
$\sigma(w_1)$
$\sigma(w_1 w_2)$
$\vdots$
$\sigma(w_1 \dots w_n)$
$\vdots$

Let $A = (\Sigma, Q, \Delta, Q_{init}, Acc)$ be a (non-deterministic) automaton for ω-words over Σ, with any kind of acceptance condition.

Definition (Good-for-Games)

A is said to be **Good-for-Games** (GFG) if there is a strategy $\sigma : \Sigma^* \to Q$ such that :

- for every $w \in \Sigma^*$ and $a \in \Sigma$, $\sigma(wa) \in \Delta(\sigma(w), a)$ (respect structure of A) ;
- for every word $w \in \Sigma^\omega$, the run generated by σ over w is accepting if and only if $w \in L(A)$.

In the context of synthesis :

- We can build a transducer directly from a GFG automaton.
- The strategy σ does **not** need to be known for this algorithm.
- Useful if there are small GFG automata without small deterministic equivalent.

Plan

Usefulness of GFG automata for synthesis depend on the acceptance condition.

Usefulness of GFG automata for synthesis depend on the acceptance condition.

Theorem (Kuperberg, Skrzypczak)

A Büchi GFG automaton of size n has an equivalent deterministic automaton with size $O(n^2)$.

Usefulness of GFG automata for synthesis depend on the acceptance condition.

Theorem (Kuperberg, Skrzypczak)

A Büchi GFG automaton of size n has an equivalent deterministic automaton with size $O(n^2)$.

Theorem (Existence of succinct coBüchi GFG automata)

There is a family $(K_n)_n$ of LTL-definable languages such that :

Usefulness of GFG automata for synthesis depend on the acceptance condition.

Theorem (Kuperberg, Skrzypczak)

A Büchi GFG automaton of size n has an equivalent deterministic automaton with size $O(n^2)$.

Theorem (Existence of succinct coBüchi GFG automata)

There is a family $(K_n)_n$ of LTL-definable languages such that :

- *K_n is recognized by a GFG coBüchi automaton of size $O(n)$.*

Usefulness of GFG automata for synthesis depend on the acceptance condition.

Theorem (Kuperberg, Skrzypczak)

A Büchi GFG automaton of size n has an equivalent deterministic automaton with size $O(n^2)$.

Theorem (Existence of succinct coBüchi GFG automata)

There is a family $(K_n)_n$ of LTL-definable languages such that :

- *K_n is recognized by a GFG coBüchi automaton of size $O(n)$.*
- *Every deterministic automaton for K_n has size $2^{\Omega(n)}$.*

Usefulness of GFG automata for synthesis depend on the acceptance condition.

Theorem (Kuperberg, Skrzypczak)

A Büchi GFG automaton of size n has an equivalent deterministic automaton with size $O(n^2)$.

Theorem (Existence of succinct coBüchi GFG automata)

There is a family $(K_n)_n$ of LTL-definable languages such that :

- *K_n is recognized by a GFG coBüchi automaton of size $O(n)$.*
- *Every deterministic automaton for K_n has size $2^{\Omega(n)}$.*

- First example of succinct GFG coBüchi by Kuperberg and Skrzypczak, but **not LTL-definable**

Usefulness of GFG automata for synthesis depend on the acceptance condition.

Theorem (Kuperberg, Skrzypczak)

A Büchi GFG automaton of size n has an equivalent deterministic automaton with size $O(n^2)$.

Theorem (Existence of succinct coBüchi GFG automata)

There is a family $(K_n)_n$ of LTL-definable languages such that :

- *K_n is recognized by a GFG coBüchi automaton of size $O(n)$.*
- *Every deterministic automaton for K_n has size $2^{\Omega(n)}$.*

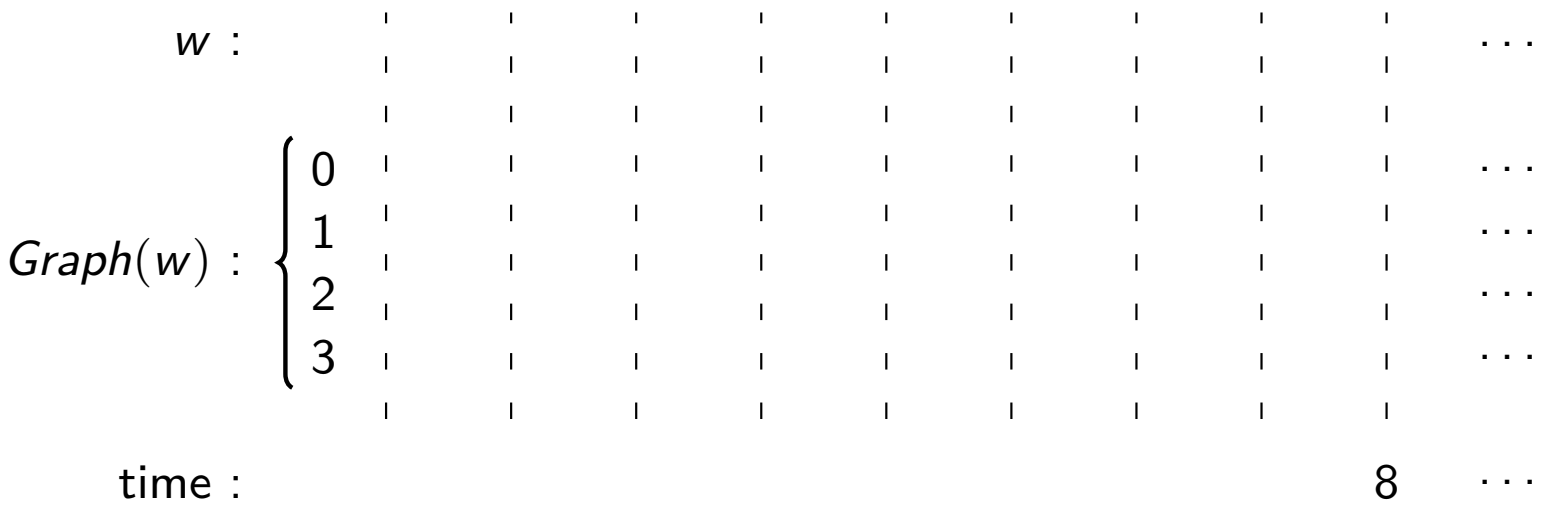
- First example of succinct GFG coBüchi by Kuperberg and Skrzypczak, but **not LTL-definable**
- The languages K_n are a variation of this first example.

The languages K_n

$\Sigma = \{\iota, a_0, a_1, a_2, b_0, b_1, b_2\}$ (for $n = 2$)

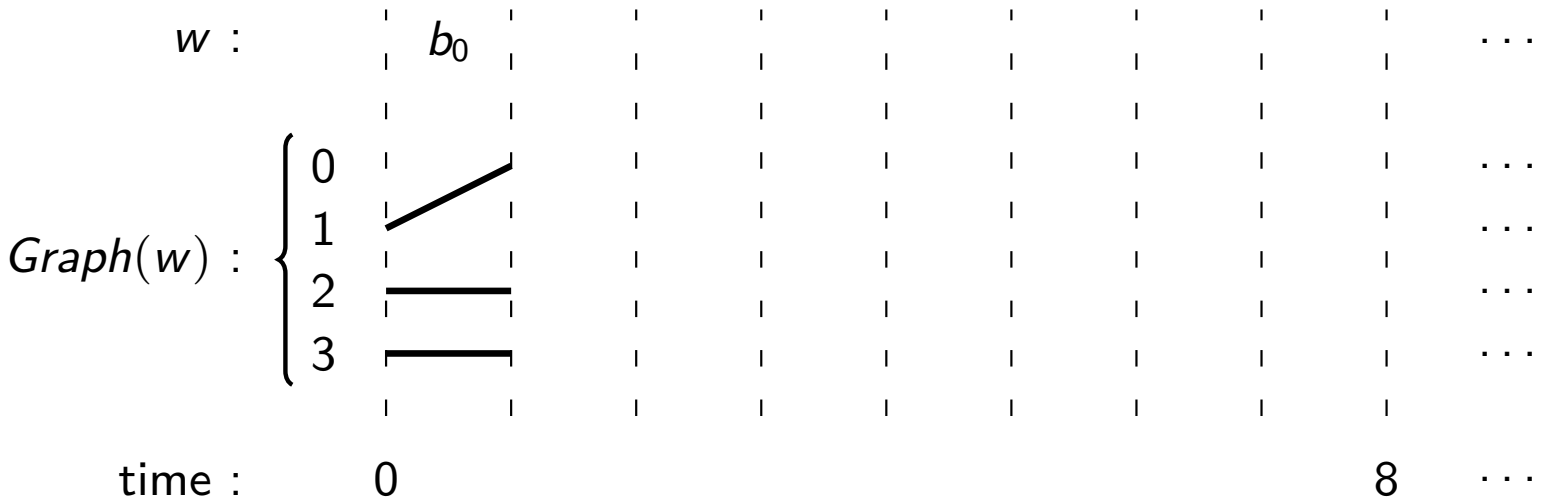
The languages K_n

$\Sigma = \{\iota, a_0, a_1, a_2, b_0, b_1, b_2\}$ (for $n = 2$)



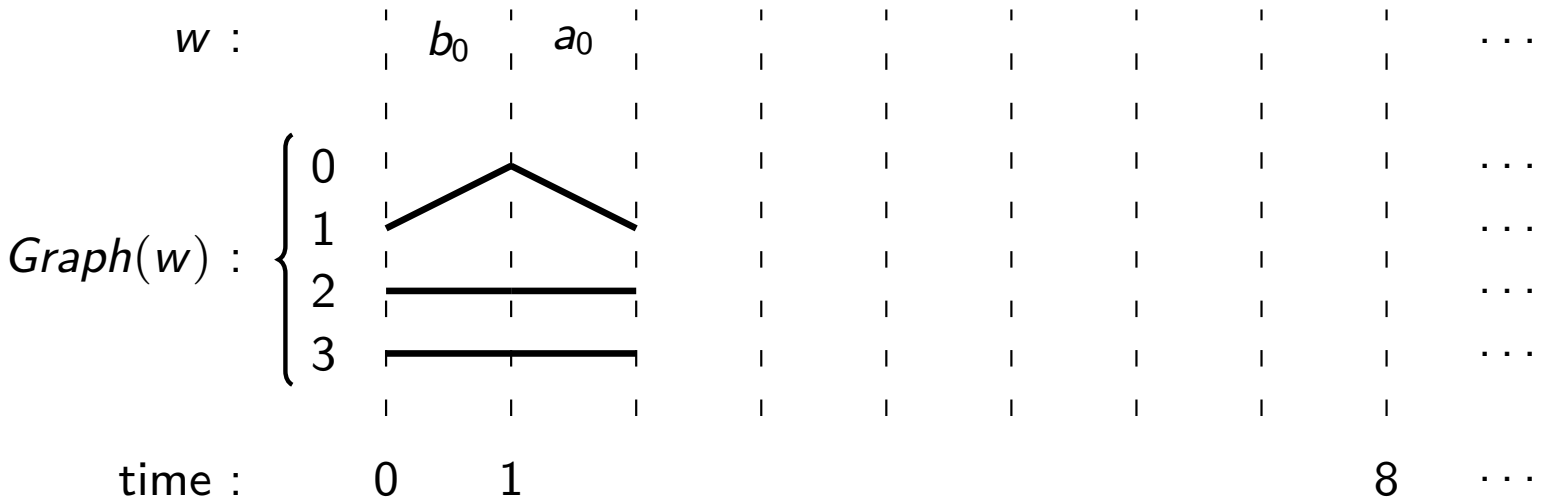
The languages K_n

$\Sigma = \{\iota, a_0, a_1, a_2, b_0, b_1, b_2\}$ (for $n = 2$)



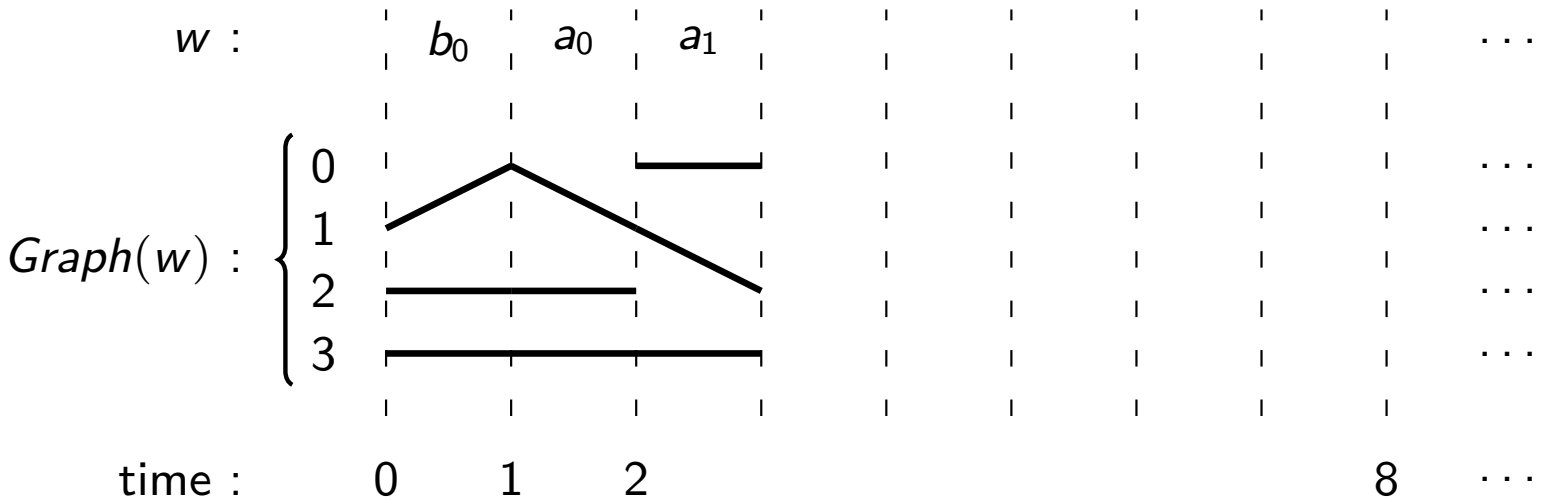
The languages K_n

$\Sigma = \{\iota, a_0, a_1, a_2, b_0, b_1, b_2\}$ (for $n = 2$)



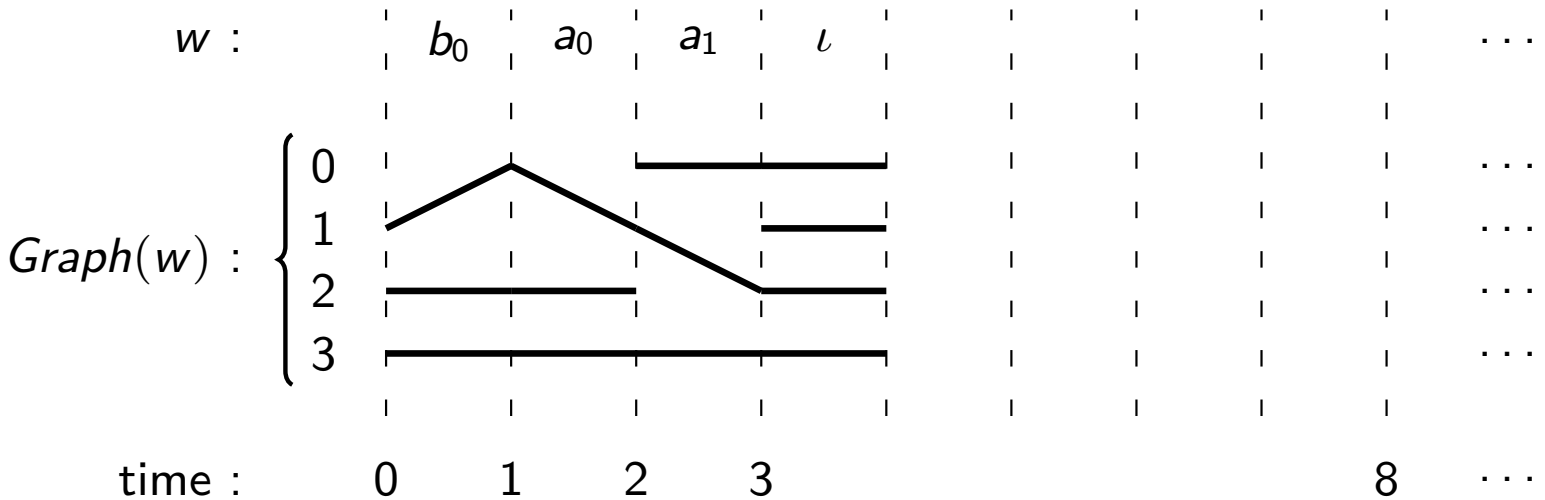
The languages K_n

$\Sigma = \{\iota, a_0, a_1, a_2, b_0, b_1, b_2\}$ (for $n = 2$)



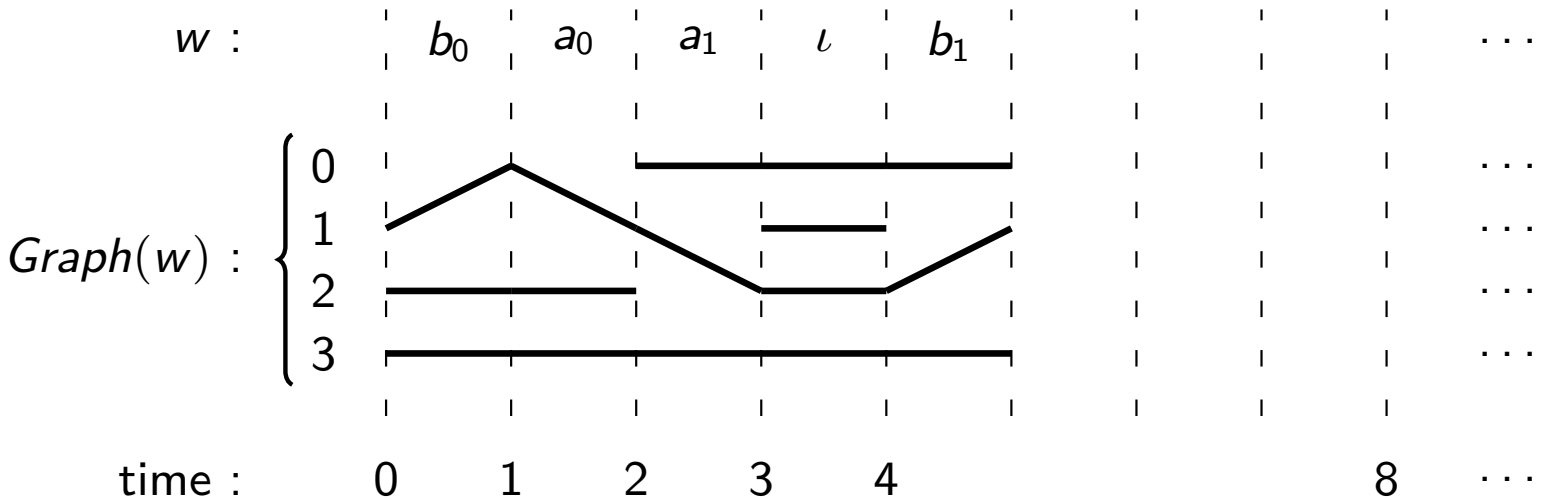
The languages K_n

$\Sigma = \{\iota, a_0, a_1, a_2, b_0, b_1, b_2\}$ (for $n = 2$)



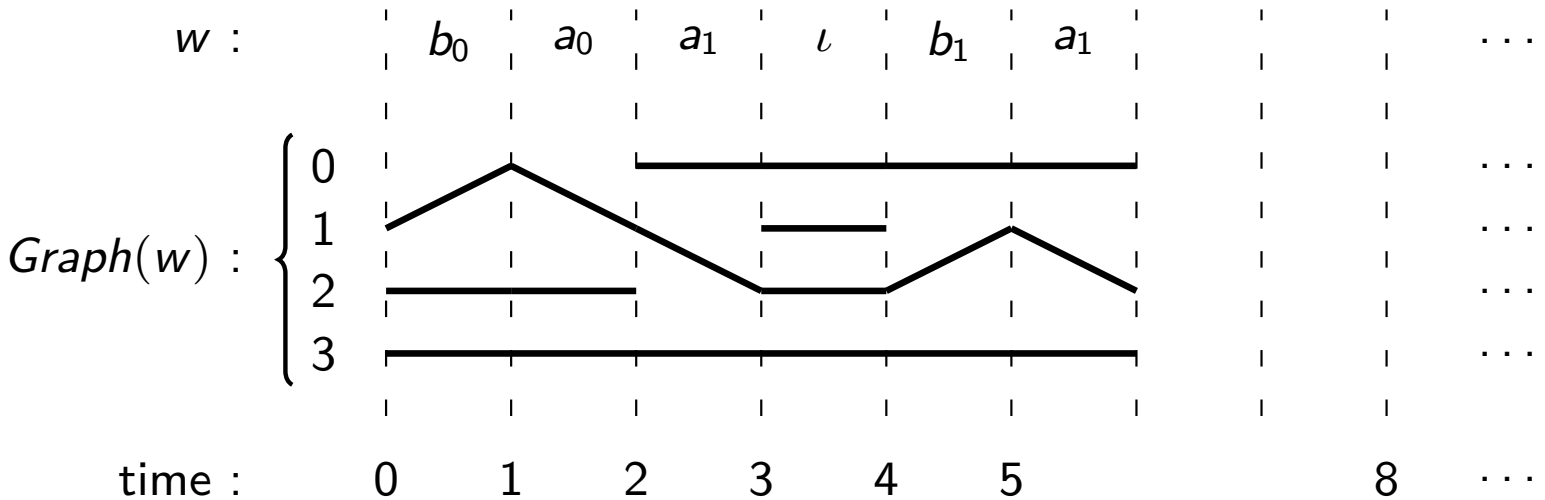
The languages K_n

$\Sigma = \{\iota, a_0, a_1, a_2, b_0, b_1, b_2\}$ (for $n = 2$)



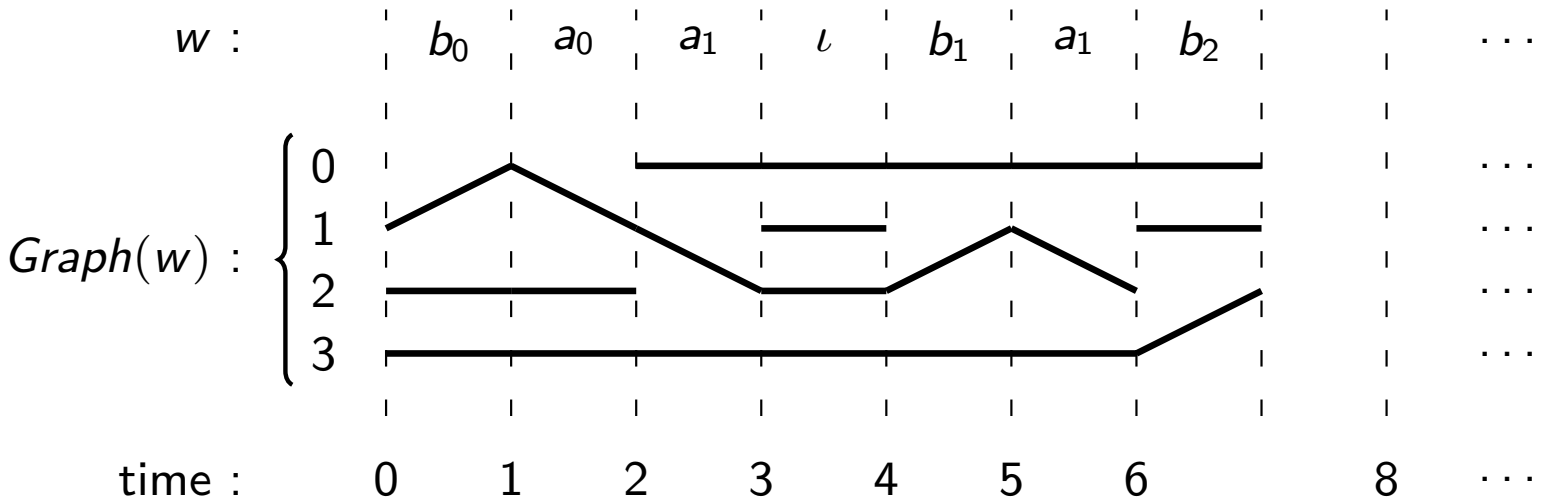
The languages K_n

$\Sigma = \{\iota, a_0, a_1, a_2, b_0, b_1, b_2\}$ (for $n = 2$)



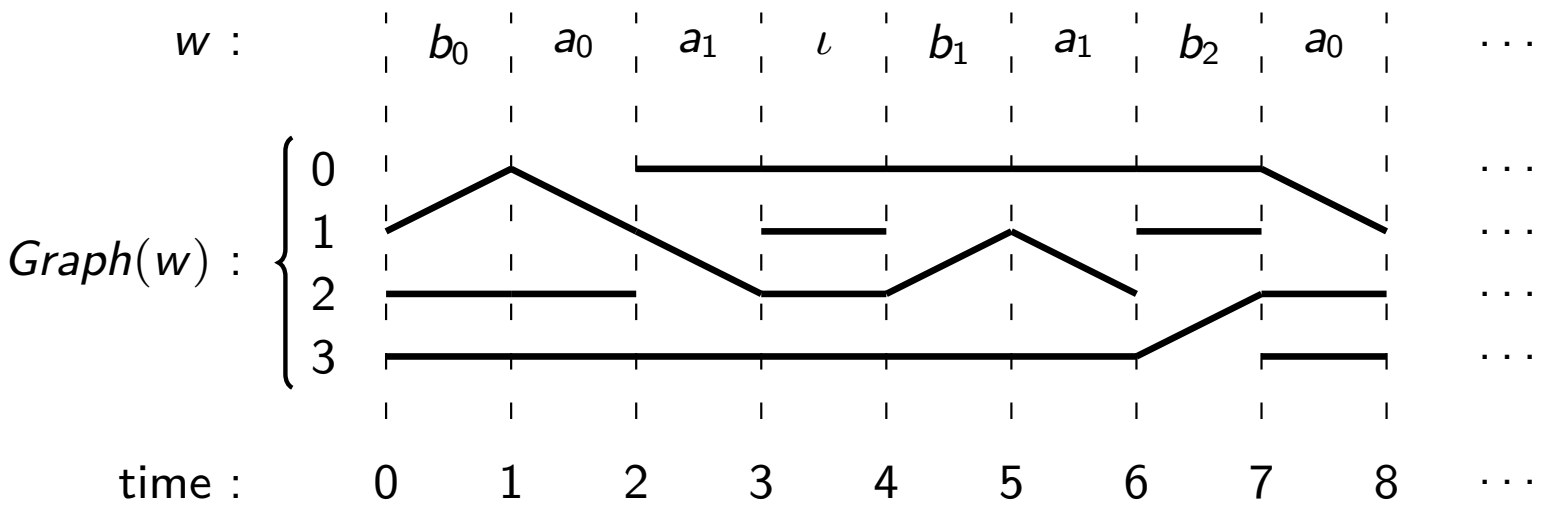
The languages K_n

$\Sigma = \{\iota, a_0, a_1, a_2, b_0, b_1, b_2\}$ (for $n = 2$)



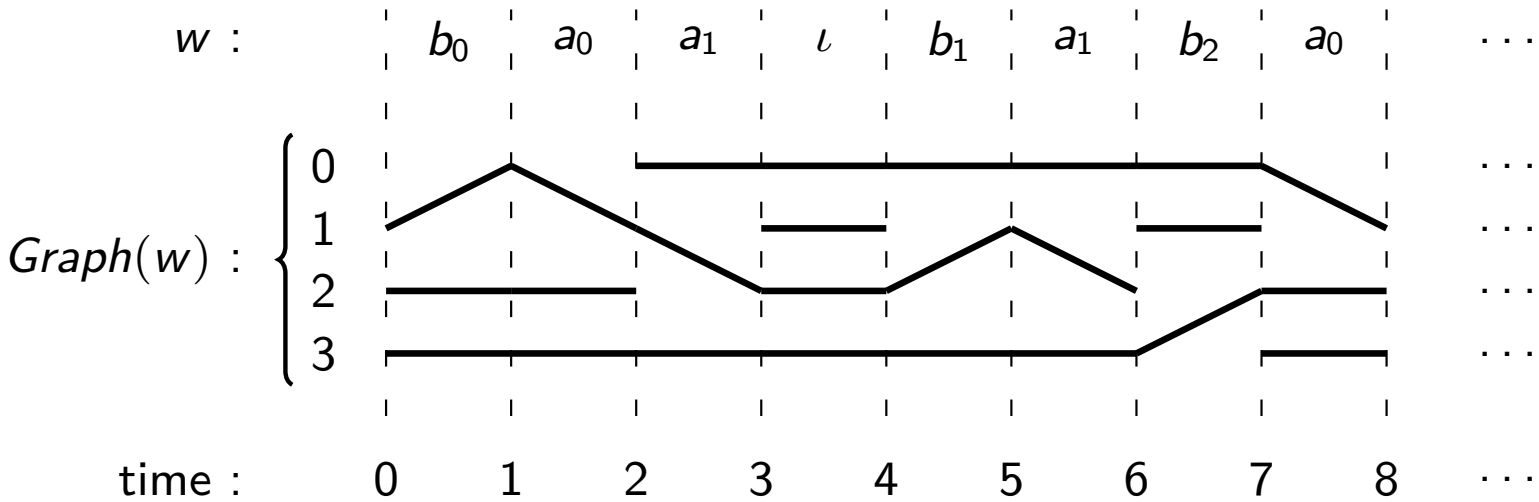
The languages K_n

$\Sigma = \{\iota, a_0, a_1, a_2, b_0, b_1, b_2\}$ (for $n = 2$)



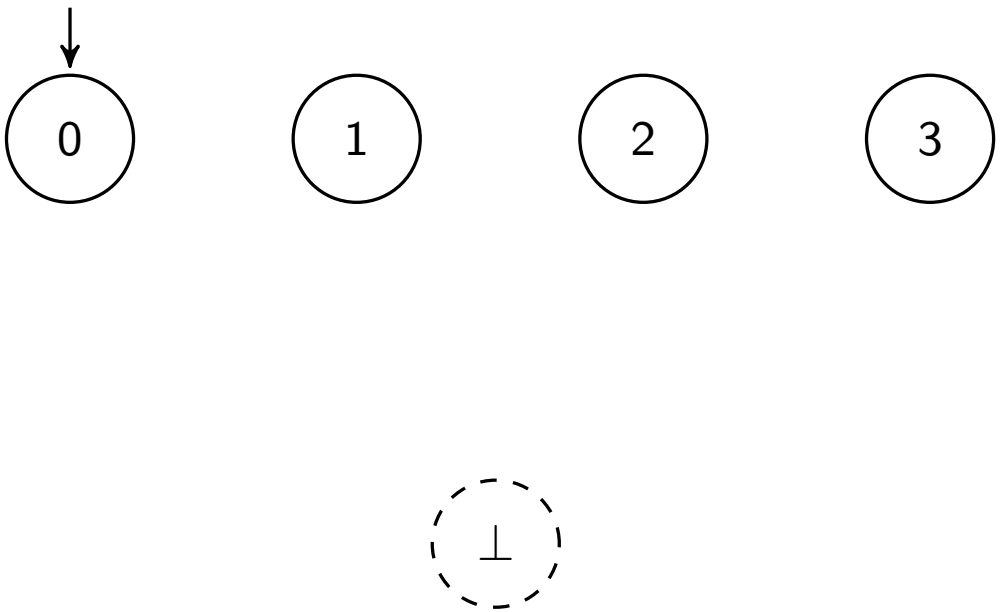
The languages K_n

$\Sigma = \{\iota, a_0, a_1, a_2, b_0, b_1, b_2\}$ (for $n = 2$)

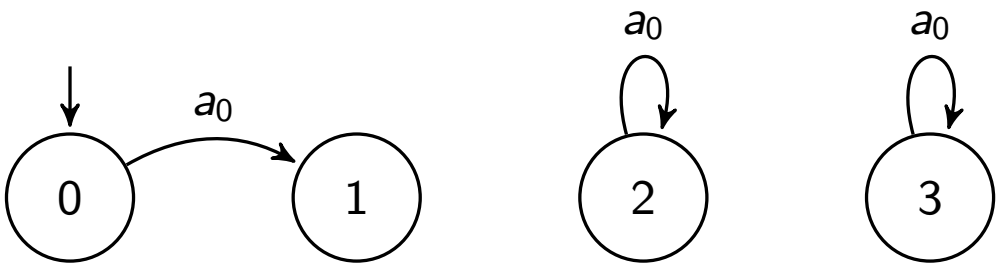


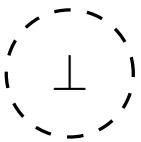
K_n is the set of words w such that $Graph(w)$ has an infinite path.

A GFG coBüchi automaton for K_n :

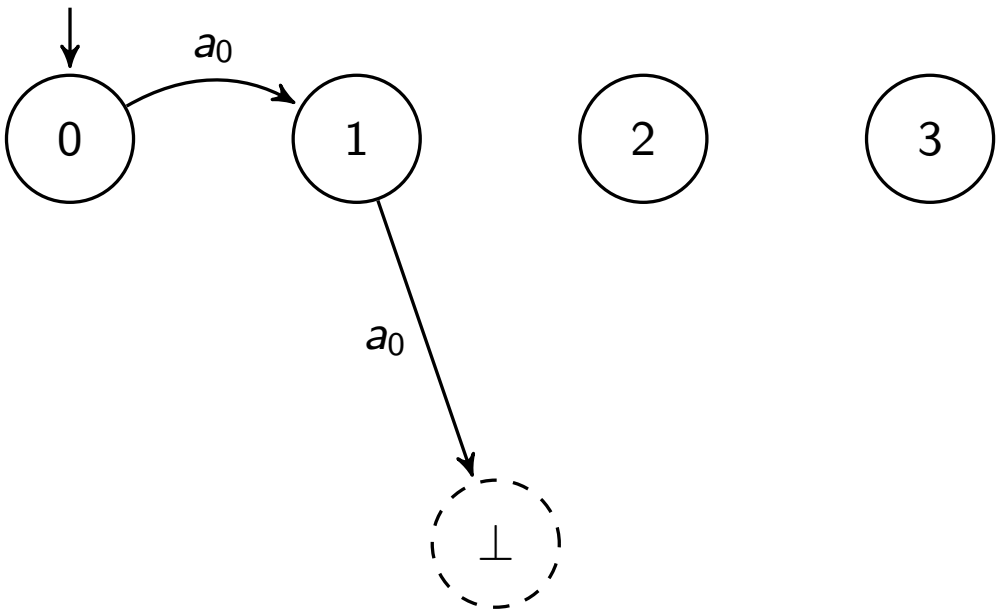


A GFG coBüchi automaton for K_n :

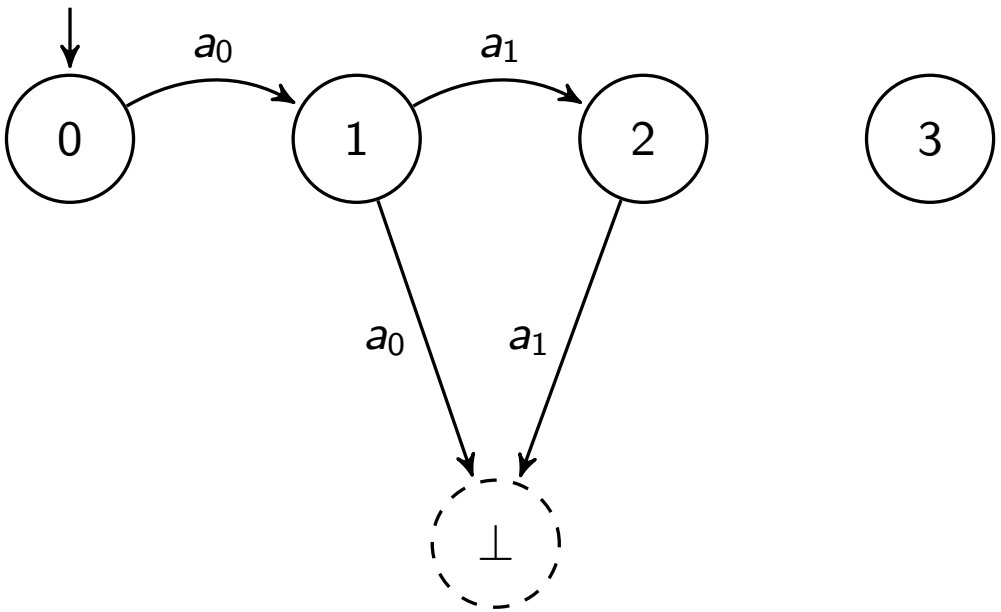




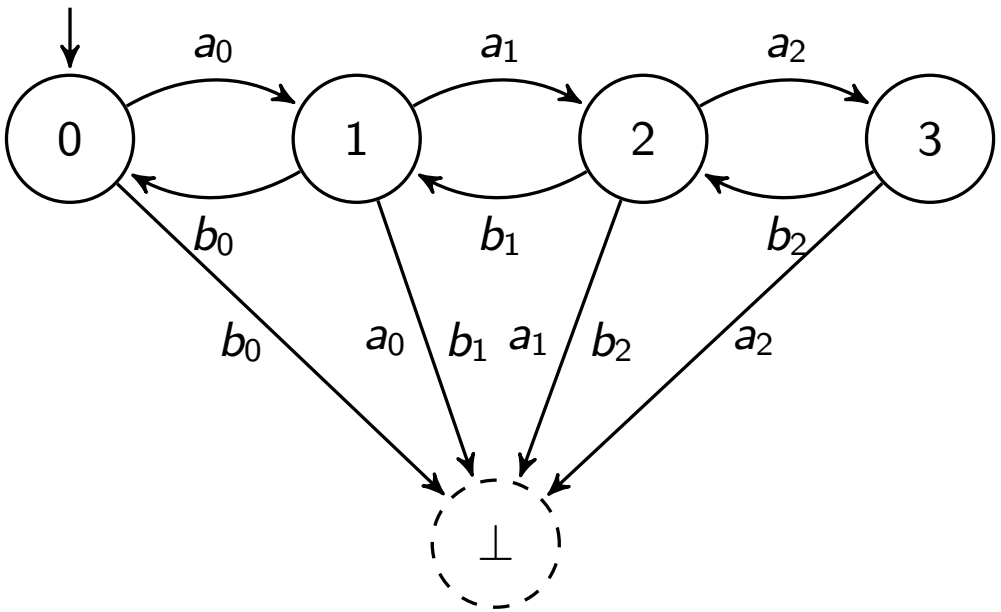
A GFG coBüchi automaton for K_n :



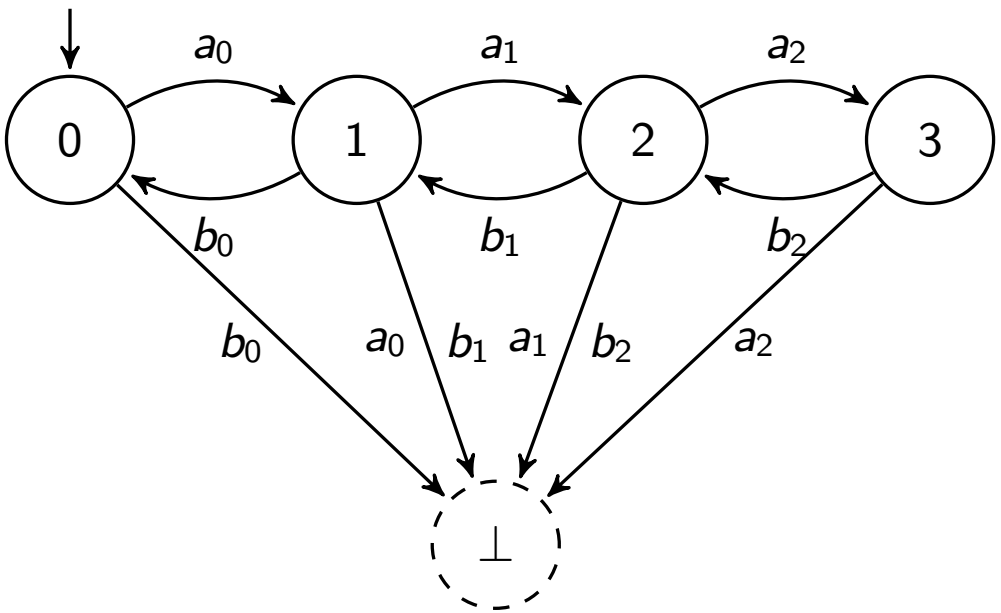
A GFG coBüchi automaton for K_n :



A GFG coBüchi automaton for K_n :



A GFG coBüchi automaton for K_n :



From any letter, jump from $\bot$ to any other state.

Theorem

K_n is LTL-definable.

Is this good news ?

Theorem

K_n is LTL-definable.

Is this good news ?

- Problem : The LTL formula we have for K_n has size exponential in n.

Theorem

K_n is LTL-definable.

Is this good news ?

- Problem : The LTL formula we have for K_n has size exponential in n.
- **Conjecture :** Every LTL formula for K_n has size exponential in n.

Theorem

K_n is LTL-definable.

Is this good news ?

- Problem : The LTL formula we have for K_n has size exponential in n.
- **Conjecture :** Every LTL formula for K_n has size exponential in n.
- **Idea :** Replace LTL with a logic better suited for GFG construction

Plan

Safety logic $S\nu TL$

- Syntax : positive boolean operators, **X** operator, greatest fixed point operator ν.
- Semantic : usual μ-calculus semantic.

Safety logic $S\nu TL$

- Syntax : positive boolean operators, **X** operator, greatest fixed point operator ν.
- Semantic : usual μ-calculus semantic.

Theorem (Folklore ?)

Languages defined by $S\nu TL$ are exactly safety languages.

Eventually Safe logic $E\nu TL$:

- Syntax : positive boolean operators, **X** operator, greatest fixed point operator ν.
- Semantic : prefix-independent closure of the usual linear μ-calculus semantic.

Eventually Safe logic $E\nu TL$:

- Syntax : positive boolean operators, **X** operator, greatest fixed point operator ν.
- Semantic : prefix-independent closure of the usual linear μ-calculus semantic.

Theorem

Languages defined by $E\nu TL$ are exactly languages of the form $\Sigma^* L_{safe}$ where L_{safe} is a suffix-closed safety language, or equivalently prefix-independent coBüchi languages.

Eventually Safe logic $E\nu TL$:

- Syntax : positive boolean operators, **X** operator, greatest fixed point operator ν.
- Semantic : prefix-independent closure of the usual linear μ-calculus semantic.

Theorem

Languages defined by $E\nu TL$ are exactly languages of the form $\Sigma^* L_{safe}$ where L_{safe} is a suffix-closed safety language, or equivalently prefix-independent coBüchi languages.

Why $E\nu TL$?

Eventually Safe logic $E\nu TL$:

- Syntax : positive boolean operators, **X** operator, greatest fixed point operator ν.
- Semantic : prefix-independent closure of the usual linear μ-calculus semantic.

Theorem

Languages defined by $E\nu TL$ are exactly languages of the form $\Sigma^* L_{safe}$ where L_{safe} is a suffix-closed safety language, or equivalently prefix-independent coBüchi languages.

Why $E\nu TL$?

- Encodes some standard verification properties (e.g. fairness).

Eventually Safe logic $E\nu TL$:

- Syntax : positive boolean operators, **X** operator, greatest fixed point operator ν.
- Semantic : prefix-independent closure of the usual linear μ-calculus semantic.

Theorem

Languages defined by $E\nu TL$ are exactly languages of the form $\Sigma^* L_{safe}$ where L_{safe} is a suffix-closed safety language, or equivalently prefix-independent coBüchi languages.

Why $E\nu TL$?

- Encodes some standard verification properties (e.g. fairness).
- Synthesis can be done modularly : can be combined with other algorithms.

Eventually Safe logic $E\nu TL$:

- Syntax : positive boolean operators, **X** operator, greatest fixed point operator ν.
- Semantic : prefix-independent closure of the usual linear μ-calculus semantic.

Theorem

Languages defined by $E\nu TL$ are exactly languages of the form $\Sigma^* L_{safe}$ where L_{safe} is a suffix-closed safety language, or equivalently prefix-independent coBüchi languages.

Why $E\nu TL$?

- Encodes some standard verification properties (e.g. fairness).
- Synthesis can be done modularly : can be combined with other algorithms.
- coBüchi GFG automata are well understood.

Eventually Safe logic $E\nu TL$:

- Syntax : positive boolean operators, **X** operator, greatest fixed point operator ν.
- Semantic : prefix-independent closure of the usual linear μ-calculus semantic.

Theorem

Languages defined by $E\nu TL$ are exactly languages of the form $\Sigma^* L_{safe}$ where L_{safe} is a suffix-closed safety language, or equivalently prefix-independent coBüchi languages.

Why $E\nu TL$?

- Encodes some standard verification properties (e.g. fairness).
- Synthesis can be done modularly : can be combined with other algorithms.
- coBüchi GFG automata are well understood.
- The languages K_n can be defined in $E\nu TL$ with formulas of size $O(n)$.

Eventually Safe logic $E\nu TL$:

- Syntax : positive boolean operators, **X** operator, greatest fixed point operator ν.
- Semantic : prefix-independent closure of the usual linear μ-calculus semantic.

Theorem

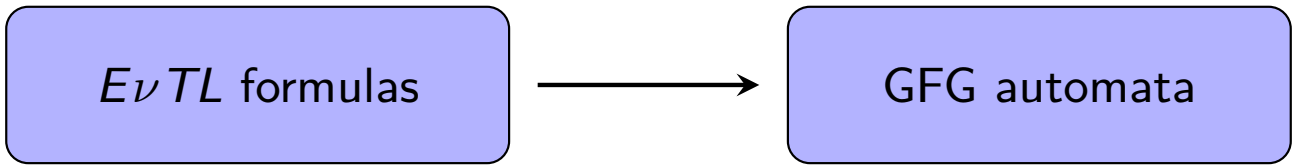
Languages defined by $E\nu TL$ are exactly languages of the form $\Sigma^* L_{safe}$ where L_{safe} is a suffix-closed safety language, or equivalently prefix-independent coBüchi languages.

Why $E\nu TL$?

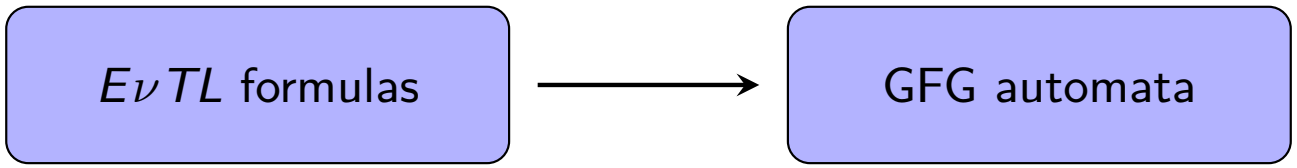
- Encodes some standard verification properties (e.g. fairness).
- Synthesis can be done modularly : can be combined with other algorithms.
- coBüchi GFG automata are well understood.

- The languages K_n can be defined in $E\nu TL$ with formulas of size $O(n)$.
- Can GFG synthesis be done in this context then ?

We can build GFG automata algorithmically from $E\nu TL$ formulas :



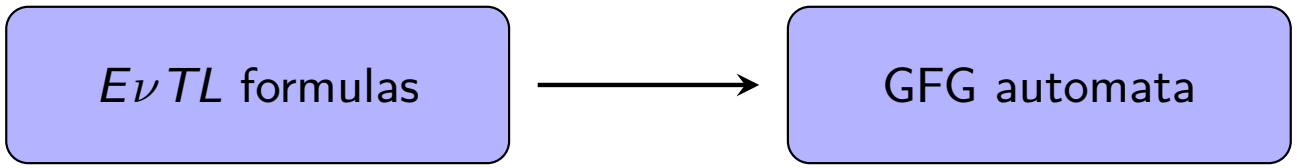
We can build GFG automata algorithmically from $E\nu TL$ formulas :



Idea of the construction :

- See the formula as a $S\nu TL$ formula, and build a safety alternating automaton from it.

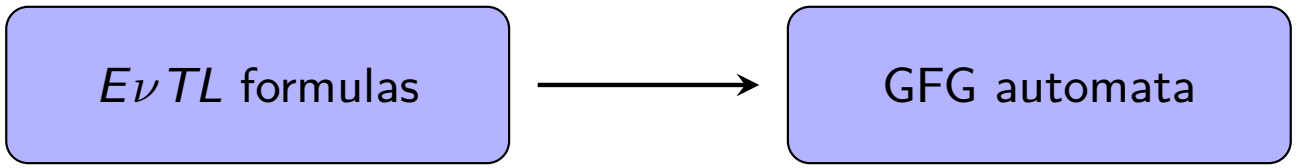
We can build GFG automata algorithmically from $E\nu TL$ formulas :



Idea of the construction :

- See the formula as a $S\nu TL$ formula, and build a safety alternating automaton from it.
- Build an equivalent deterministic safety automaton.

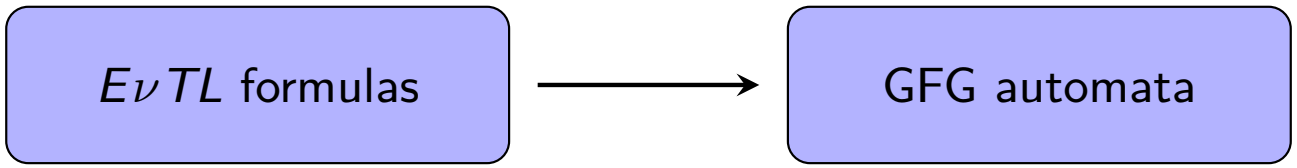
We can build GFG automata algorithmically from $E\nu TL$ formulas :



Idea of the construction :

- See the formula as a $S\nu TL$ formula, and build a safety alternating automaton from it.
- Build an equivalent deterministic safety automaton.
- Minimize the result, and build a coBüchi automaton by adding a $\bot$ state, and following the deterministic transitions when possible, going to $\bot$ otherwise, and jumping to any state from $\bot$.

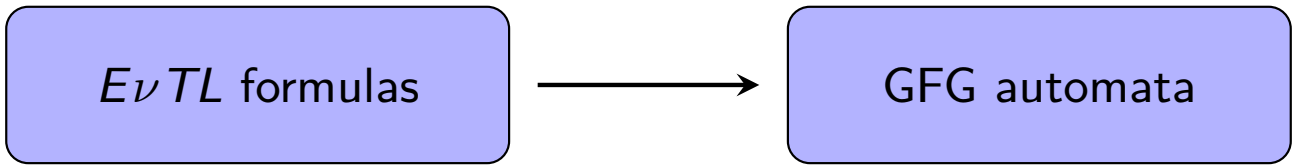
We can build GFG automata algorithmically from $E\nu TL$ formulas :



Idea of the construction :

- See the formula as a $S\nu TL$ formula, and build a safety alternating automaton from it.
- Build an equivalent deterministic safety automaton.
- Minimize the result, and build a coBüchi automaton by adding a $\bot$ state, and following the deterministic transitions when possible, going to $\bot$ otherwise, and jumping to any state from $\bot$.
- GFG-ness follows from the same kind of idea as before : jump to the path that has been uncut for the longest time.

We can build GFG automata algorithmically from $E\nu TL$ formulas :

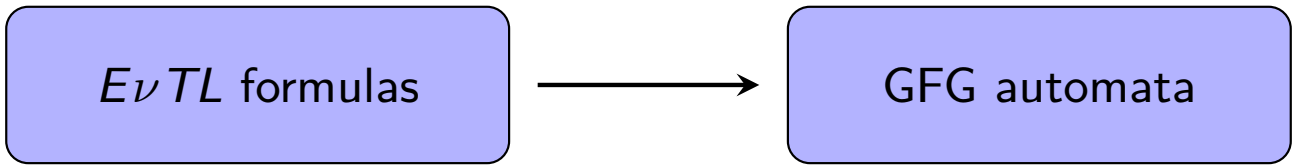


Idea of the construction :

- See the formula as a $S\nu TL$ formula, and build a safety alternating automaton from it.
- Build an equivalent deterministic safety automaton.
- Minimize the result, and build a coBüchi automaton by adding a $\bot$ state, and following the deterministic transitions when possible, going to $\bot$ otherwise, and jumping to any state from $\bot$.
- GFG-ness follows from the same kind of idea as before : jump to the path that has been uncut for the longest time.

In general : the translation problem from $E\nu TL$ to GFG automata is doubly exponential.

We can build GFG automata algorithmically from $E\nu TL$ formulas :



Idea of the construction :

- See the formula as a $S\nu TL$ formula, and build a safety alternating automaton from it.
- Build an equivalent deterministic safety automaton.
- Minimize the result, and build a coBüchi automaton by adding a $\perp$ state, and following the deterministic transitions when possible, going to $\perp$ otherwise, and jumping to any state from $\perp$.
- GFG-ness follows from the same kind of idea as before : jump to the path that has been uncut for the longest time.

In general : the translation problem from $E\nu TL$ to GFG automata is doubly exponential.

Theorem

The resulting GFG automaton has a minimal amount of states.

Plan

Future work

Future work

- Prove that K_n has no short LTL definition. Would imply succinctness of μ-calculus over LTL !

Future work

- Prove that K_n has no short LTL definition. Would imply succinctness of μ-calculus over LTL !
- Find fragments of LTL or μ-calculus that are Good-for-Good-for-Games (i.e. no doubly exponential blowup).