LTLf Synthesis

Rémy LE BOHEC
under the supervision of Philipp Schlehuber-Caissier

EPITA Research Laboratory

July 2024 Seminar

Table of Contents

Table of Contents

What is LTL?

LTL is a formalism used in model checking to describe and reason over sequences of boolean variables.

What is LTL?

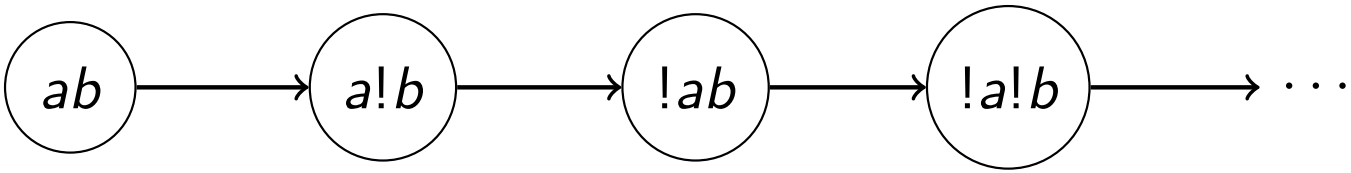
LTL is a formalism used in model checking to describe and reason over sequences of boolean variables.

LTL: the language

$$\varphi ::= p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \implies \varphi_2$$
$$\mid X\varphi \mid F\varphi \mid G\varphi \mid \varphi_1 U \varphi_2 \mid \varphi_1 W \varphi_2 \mid \varphi_1 R \varphi_2 \mid \varphi_1 M \varphi_2$$

What is LTL?: traces

A trace (also known as a path or a run) is a sequence of boolean variables.



What is LTL?: Atomic propositions

$$\varphi ::= p$$

p is an atomic proposition, a single boolean variable.

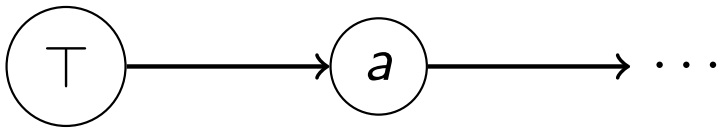
Atomic propositions can also be combined using boolean logic operators (not, and, etc).

What is LTL?: Next

$$\varphi ::= X\psi$$

X is the next operator. It means that the formula must hold starting at the next state.

For the formula $\varphi = Xa$:



What is LTL?: While and Until

$$\varphi ::= \psi_1 W \psi_2 \mid \psi_1 U \psi_2$$

U is the until operator. ψ_1 must hold until ψ_2 becomes (and will become true).
W (while) is the same except ψ_2 may never be true.

For the formula $\varphi = a\ U\ b$:

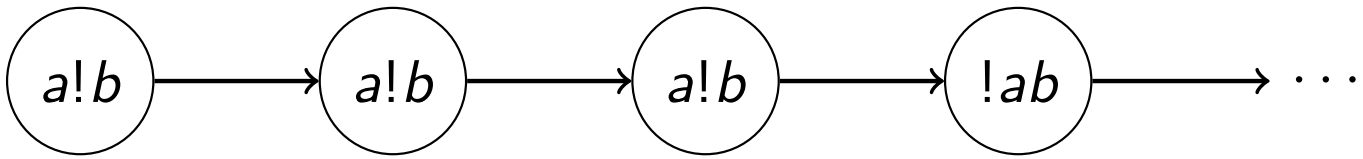


Table of Contents

What is LTLf?

LTL is used to reason about infi-
nite sequences of states which is convenient for systems who run indefinitely.

LTLf is a modification of LTL intended for finite traces, that is traces with a finite number of states.

LTLf examples

While LTLf is similar to LTL, it posesses some slight differences.

LTLf adds variants which wouldn't make sense in a LTL context, such as weak next (X).

LTLf examples

While LTLf is similar to LTL, it posesses some slight differences.

LTLf adds variants which wouldn't make sense in a LTL context, such as weak next (X).

Weak next of φ means that if a future state exists, then φ must hold in it.

Strong next ($X[!]$) means that the future state must exist and φ must hold in it.

LTLf examples

As a consequence, consider the following formula:

LTLf examples

As a consequence, consider the following formula:

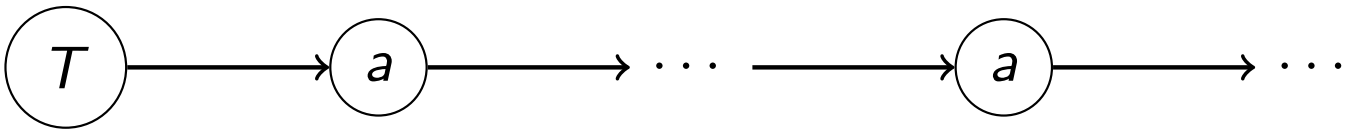
$$\varphi = G(X!a)$$

LTLf examples

As a consequence, consider the following formula:

$$\varphi = G(X!a)$$

In LTL, a valid trace could have been:

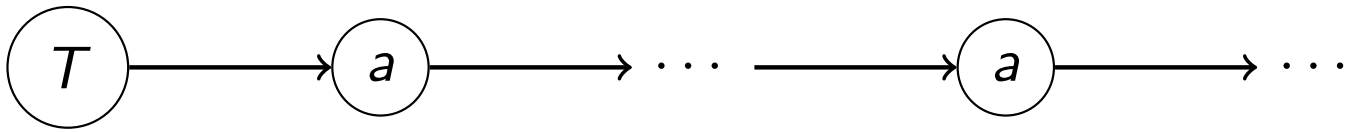


LTLf examples

As a consequence, consider the following formula:

$$\varphi = G(X!a)$$

In LTL, a valid trace could have been:



However, in LTLf, no trace is valid! (Because of the strong next).

Table of Contents

Why LTLf?

First, LTLf is more appropriate to use than LTL when studying systems based on finite tasks (processes, workflows) compared to systems which should run indefinitely such as operating systems.

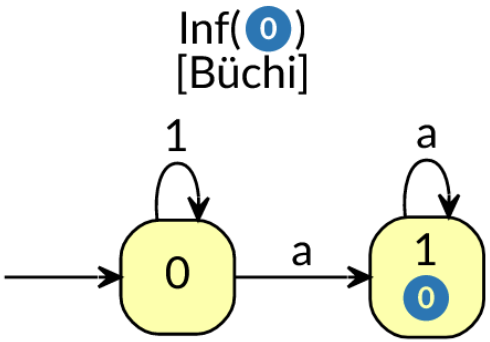
Why LTLf?

First, LTLf is more appropriate to use than LTL when studying systems based on finite tasks (processes, workflows) compared to systems which should run indefinitely such as operating systems.

Then, LTLf can be translated into simpler automata than LTL.
We do not need Büchi acceptance for infinite words since we work with finite traces.

A dedicated translation?

Typically, LTL is translated into deterministic TELA (transition based Emerson-Lei automata) or nondeterministic generalized Büchi automata.



For LTLf, since all words are finite, we can directly translate into deterministic finite automata.

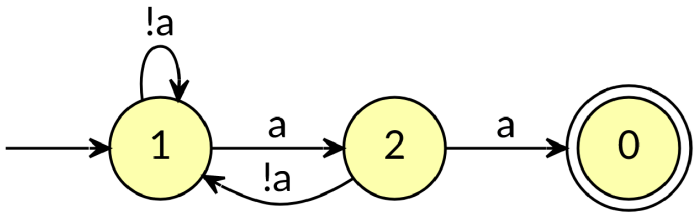


Table of Contents

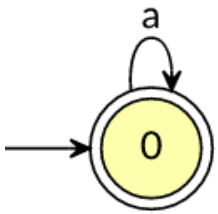
Formula rewriting

Given a LTLf formula φ, we want to rewrite the formula into a current condition and a future formula that we can process later.

Let $\varphi = G(a)$.
φ rewrites to $\varepsilon \mid a \;\&\; X(G(a))$: we want to accept or have a on the current state and $G(a)$ in the next state if it exists.

We already are translating $G(a)$, meaning we can loop on it.

The resulting automaton looks like this:



Linear forms

Linear forms are vectors of (bdd, formula) representing all transitions from a state.

Linear forms

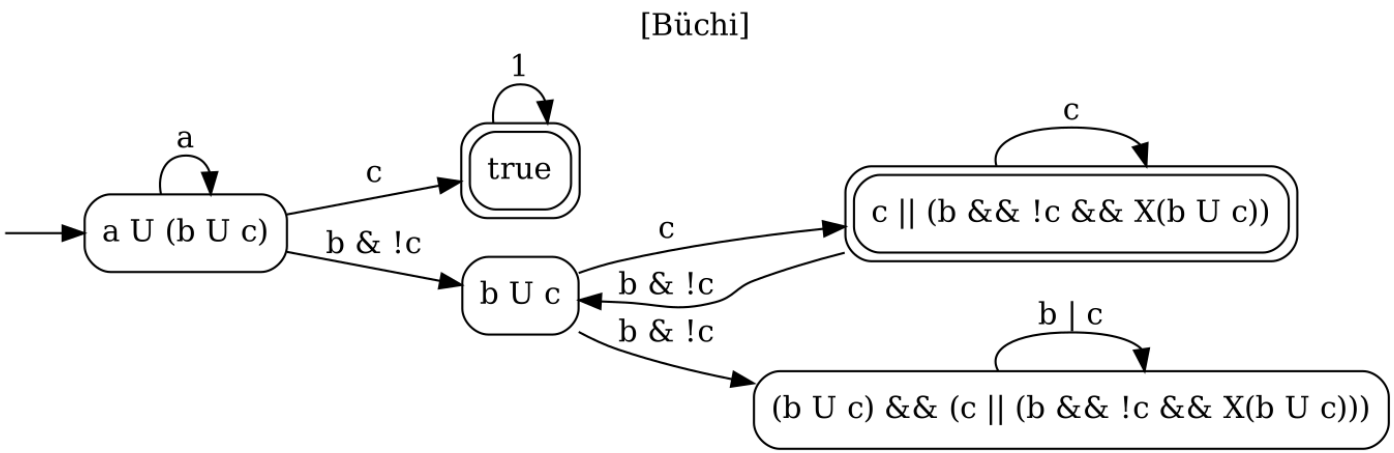
Linear forms are vectors of (bdd, formula) representing all transitions from a state.

A formula is rewritten into a linear form, and the process is reapplied to new formulas that appear.

Linear forms

Linear forms are vectors of (bdd, formula) representing all transitions from a state.

A formula is rewritten into a linear form, and the process is reapplied to new formulas that appear.



State 0 linear form: $[(a, a\ U\ (b\ U\ c)), (c, \top), (b\&!c, b\ U\ c)]$

Formula graph

To assist in the translation, we use a formula graph to:

Formula graph

To assist in the translation, we use a formula graph to:

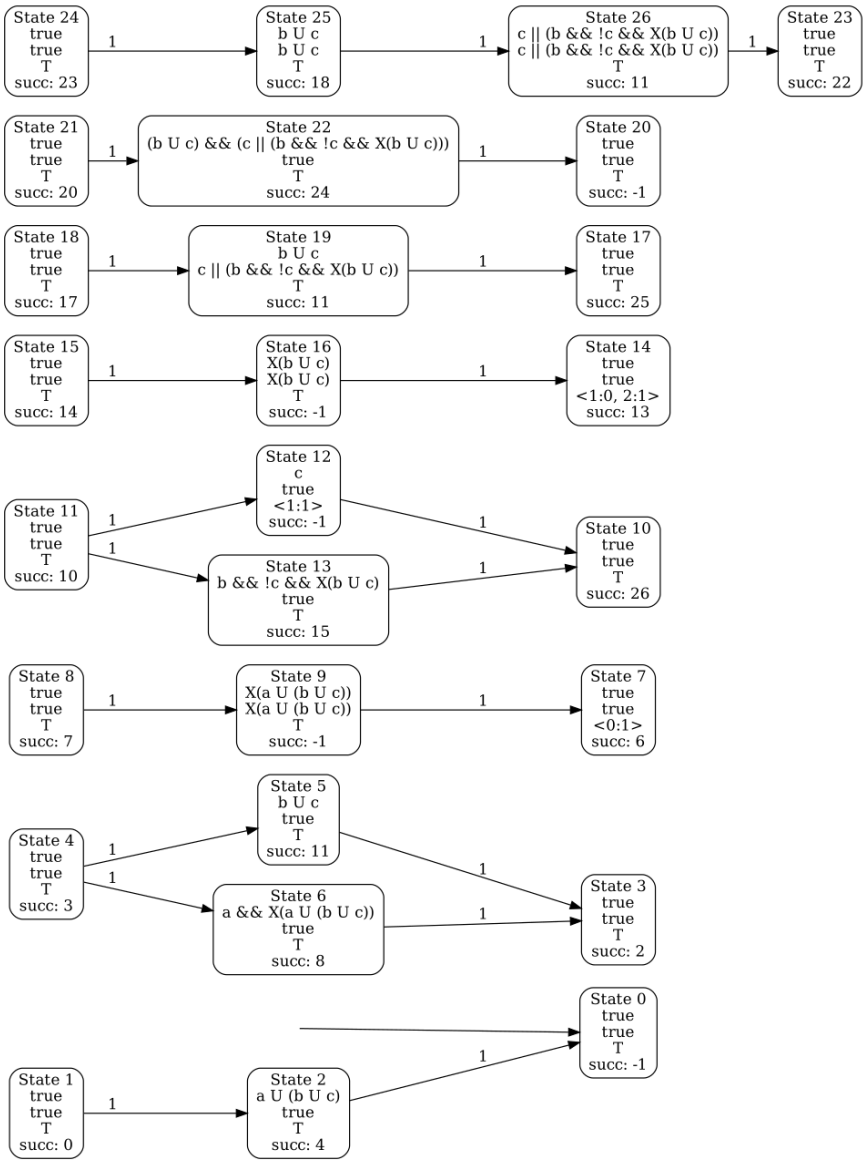
- Store the formulas we have already translated

Formula graph

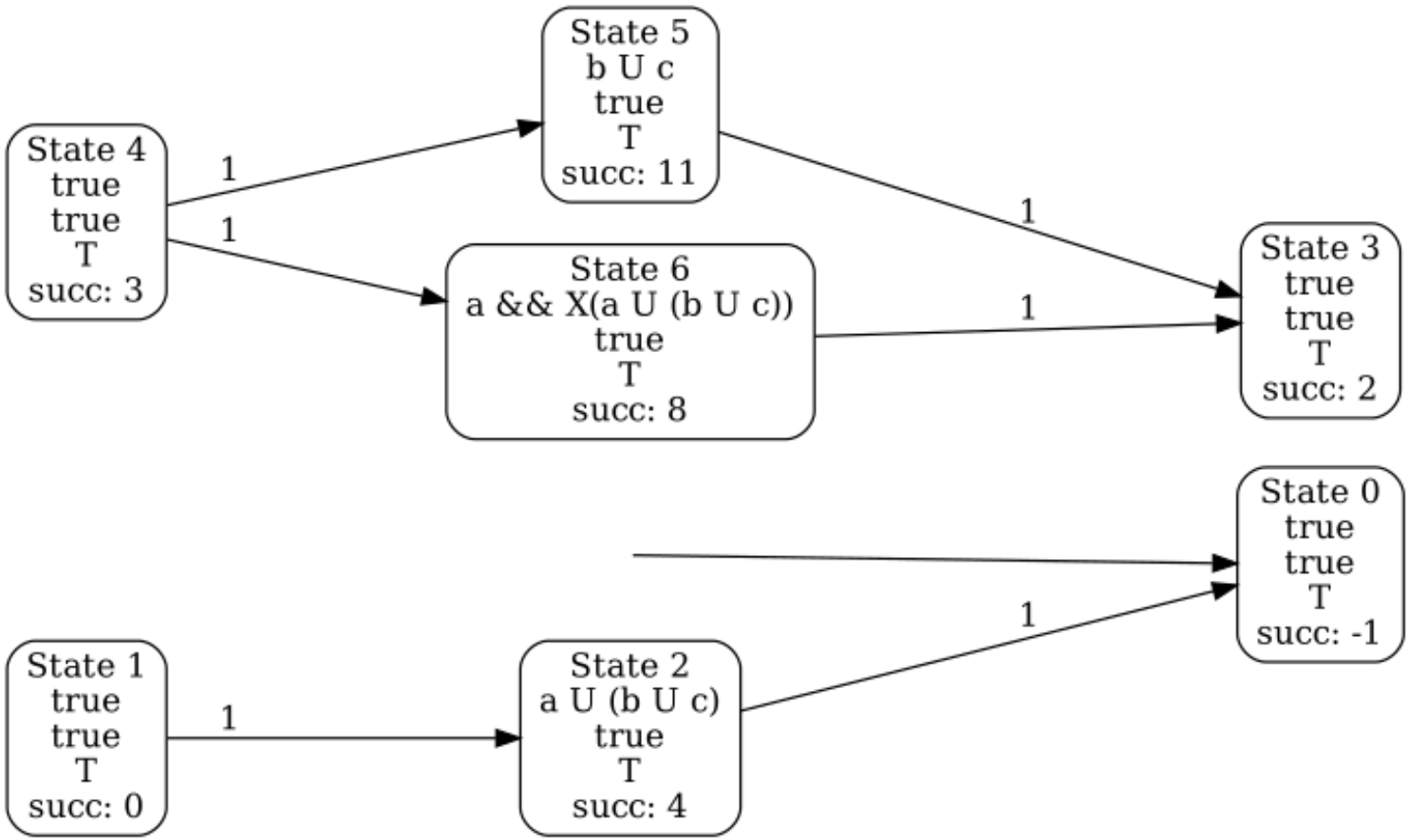
To assist in the translation, we use a formula graph to:

- Store the formulas we have already translated
- Remember which states to go back to once a formula has been visited

Formula graph: an example



Formula graph: a visible example



Determinize and Merge

In order to produce deterministic finite automata while translating the formula, I wrote two functions: determinize and merge_transitions.

Determinize and Merge

In order to produce deterministic finite automata while translating the formula, I wrote two functions: determinize and merge_transitions.

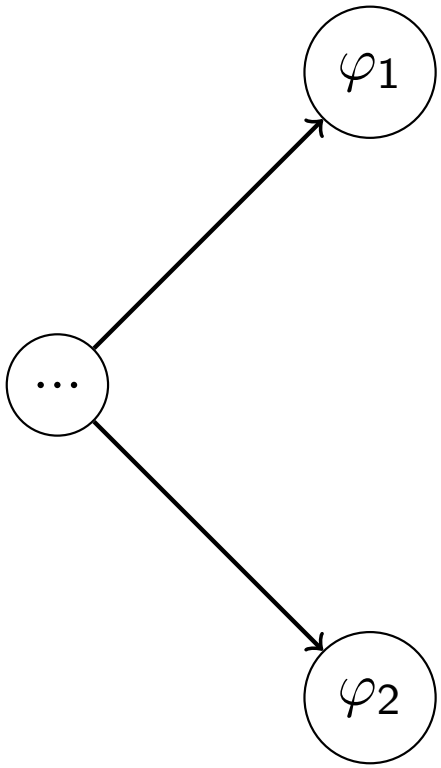
The determinize function aims to partition all transitions such that no conditions overlap.

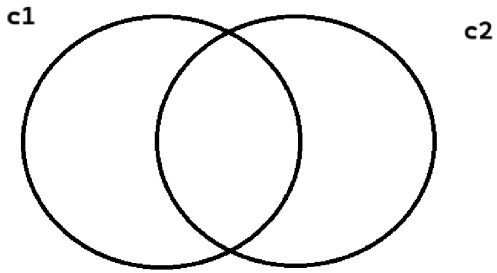
determinize

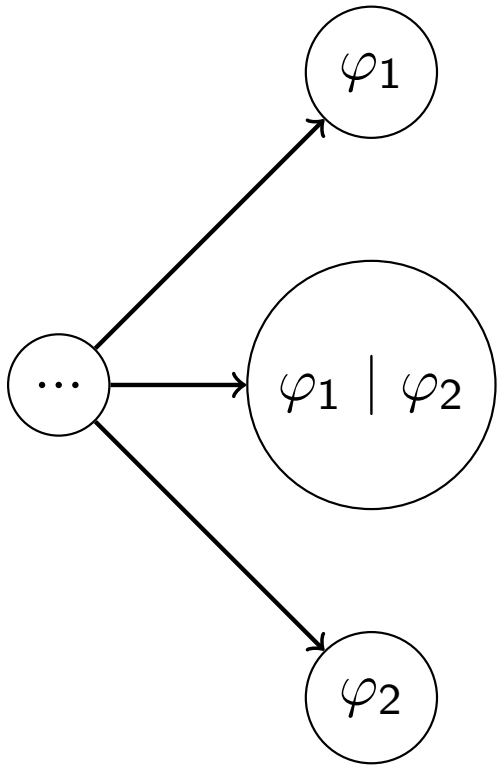
First, we create an implication graph whose leaves are all disjoint and can be combined to form the constraints in the linear form (bdd_partition).

Then, for each condition c of the partition, we make a disjunction of all formulas in the linear form which can be reached by c.

determinize: example







merge_transitions

The objective here is to add transitions that lead to the same formula.

We start with our linear form:
$[(c_1, \varphi_1), (c_2, \varphi_2), (c_3, \varphi_1), (c_4, \varphi_2)]$

We iterate over the linear form, starting at (c_1, φ_1) and grouping all transitions with the same formula:
$[(c_1|c_3, \varphi_1), (c_2, \varphi_2), (c_4, \varphi_2)]$

We advance and repeat until we reach the end of the linear form:
$[(c_1|c_3, \varphi_1), (c_2|c_4, \varphi_2)]$

Table of Contents

Model checking

Model checking is a verification technique that explores all reachable system states to ensure that the induce runs to not violate the specification.

Model checking

Model checking is a verification technique that explores all reachable system states to ensure that the induce runs to not violate the specification.

In our case, given a LTLf specification, we want to ensure that a system behaves properly.

Model checking example

Consider a traffic light.

Figure: Are you a robot?

To prevent deadlocks, we want red light to eventually be followed up by a green light.

Therefore, the specification is $G(r \implies g)$.

Model checking here would be checking the reachable execution states of the light to ensure the green light eventually turns on.

Table of Contents

Model checking on the fly

Usually, model checking is done by traversing the FA corresponding to the controller alongside the FA corresponding the the negation of the specification ($\neg\varphi$).

Model checking on the fly

Usually, model checking is done by traversing the FA corresponding to the controller alongside the FA corresponding the the negation of the specification ($\neg\varphi$).

However, we decided to look into doing so on the fly: that is, building the linear forms associated with the specification as we go to potentially avoid building the whole automata.

WIP algorithm

Given a LTLf formula φ and a symbolic automaton ctrl representing a controller, we want to perform a depth first traversal of the automata associated with $\neg\varphi$ and ctrl to look for accepting paths.

WIP algorithm

Given a LTLf formula φ and a symbolic automaton ctrl representing a controller, we want to perform a depth first traversal of the automata associated with $\neg\varphi$ and ctrl to look for accepting paths.

We explore states (φ, bdd) by traversing the linear form associated with φ and retrieving the successor (if it exists) of the symbolic automata.

WIP algorithm

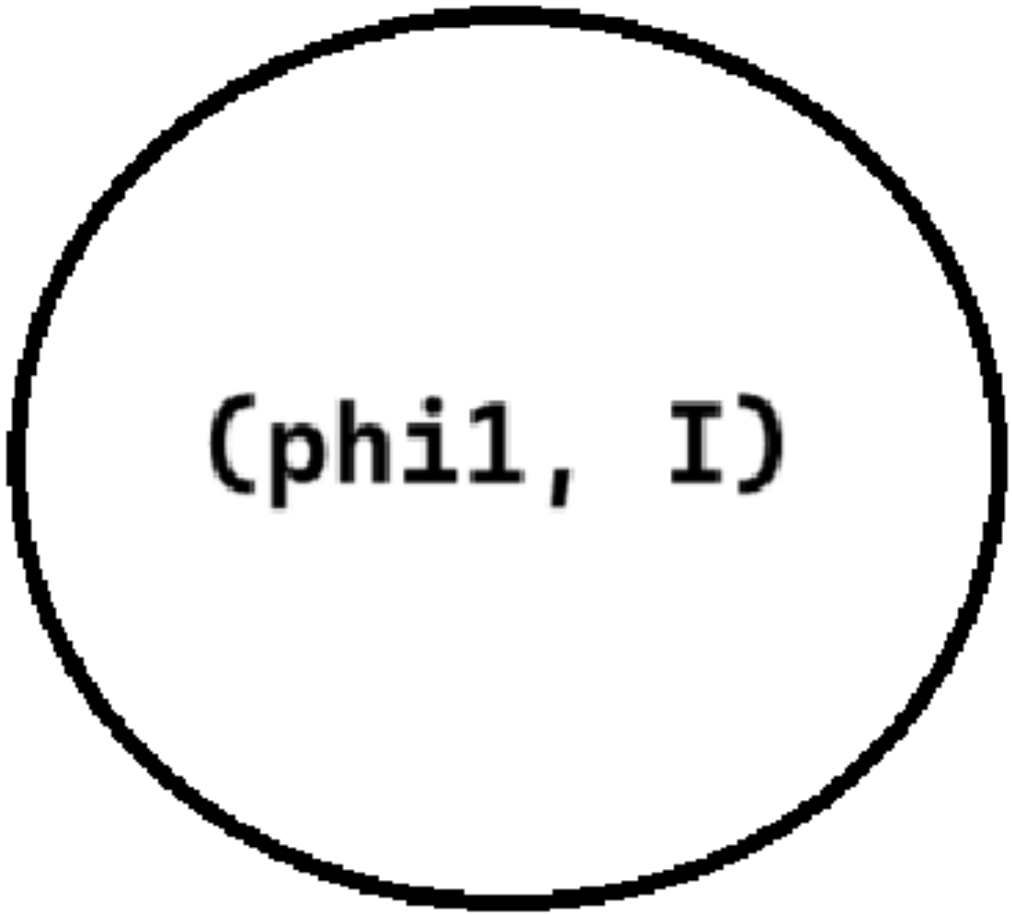
For terminating successors of the ctrl, we check for transitions leading to terminating states in the linear form and reject if needed.

WIP algorithm

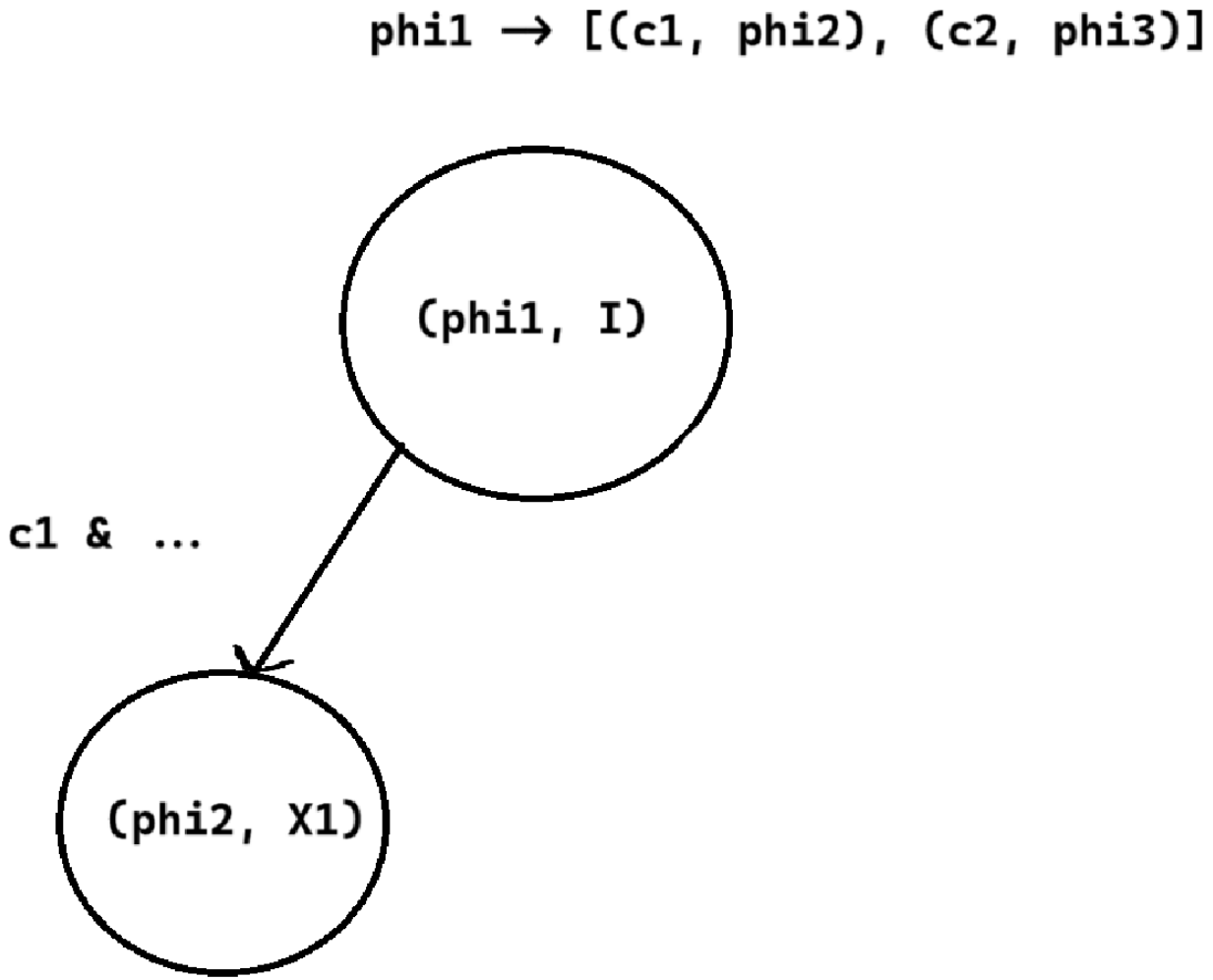
For terminating successors of the ctrl, we check for transitions leading to terminating states in the linear form and reject if needed.

For the non-terminating successors, we continue exploring while possible.

WIP algorithm: example



WIP algorithm: example



WIP algorithm: example

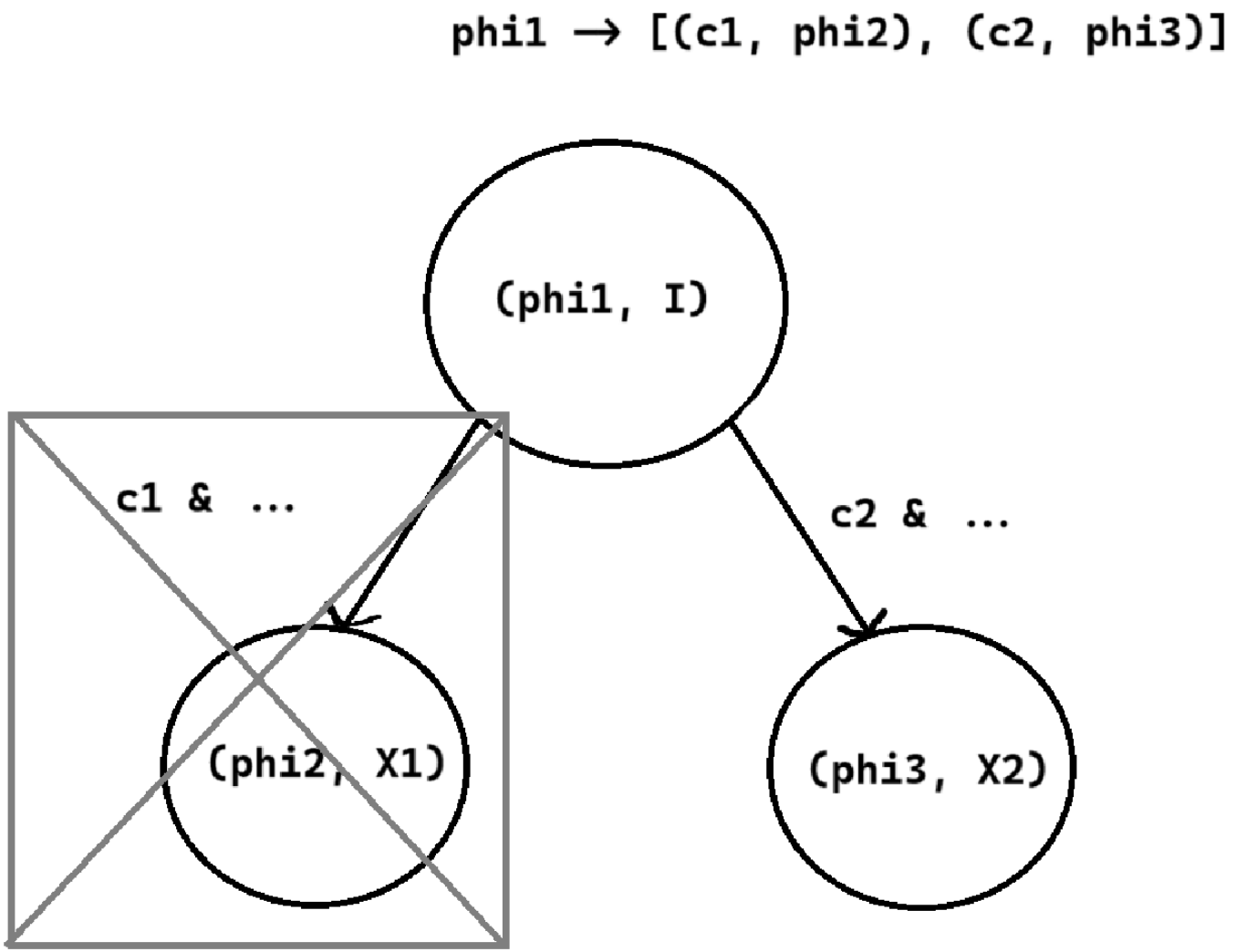


Table of Contents

Conclusion

My work aimed to improve the current process of synthesis and model checking in spot regarding LTLf.

Conclusion

My work aimed to improve the current process of synthesis and model checking in spot regarding LTLf.

While the model checking algorithm has yet to be fully implemented and tested, the LTLf to DFA part produces satisfactory results.

Conclusion

My work aimed to improve the current process of synthesis and model checking in spot regarding LTLf.

While the model checking algorithm has yet to be fully implemented and tested, the LTLf to DFA part produces satisfactory results.

Some improvements are possible which hopefully will be ready for next year's edition of SYNTCOMP: a competition for reactive synthesis tools and also to integrate within Lisa.

Bibliography

Pnueli, A. 1977. The temporal logic of programs. In FOCS, 46–57. IEEE.

De Giacomo, G., Favorito, M., Li, J., Vardi, M. Xiao, S., Zhu, S. 2022. LTLf Synthesis as AND-OR Graph Search: Knowledge Compilation at Work. IJCAI-ECAI.

De Giacomo, G. 2023. Linear-time Temporal Logics on Finite Traces. AAAI 2023 Spring Symposium.