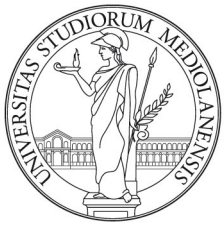


**UNIVERSITÀ DEGLI STUDI DI MILANO**

FACOLTÀ DI SCIENZE E TECNOLOGIE

Corso di Laurea Magistrale in Matematica

# Decision Techniques for Parameterized Verification: the case of Token-Passing Rings

Relatore:
Prof. Silvio GHILARDI

Tesi di laurea di:
Enrico LIPPARINI
Matr. 921371

Anno Accademico 2018/2019

# Contents

# Introduction

Model checking is a powerful method to formal verify properties of computer systems of various kinds, having more and more applications in proving correctness of both hardware and software systems. Indeed, it plays a fundamental role in many fields where malfunctioning is not an option: well-known examples involve air traffic control systems, medical technologies, telecommunications networks, and e-commerce.

Along with the increasing use of computer systems in key economic sectors during the second half of the 1900s, ensuring their correctness became a hot topic in computer science. Nonetheless, not much progress was done until the beginning of the 80's: in fact, most strategies involved hand constructed proof (mostly using Floyd-Hoare logic). There were still no appropriate frameworks to express concepts like mutual exclusion or deadlocks. In the late 1970's Pnueli found the missing link: temporal logics. In fact, given a concurrent system, by describing the behaviour of its individual statements as a set of axioms, the problem can be reduced to prove properties in the constructed theory. The very last steps to get to model checking where made by Clarke, Emerson, and Sifakis. In addition to their fundamental works on improving temporal logics expressiveness, their great merit was to put forward the idea that instead of verifying through a synthesis method that a formula is valid in every model, what really matters is to explicitly show a finite model in which the formula is not valid.

In general, model checking can be applied to a wide range of systems, like sequential, distributed, concurrent, reactive, and more. However, it has proven to be particularly successful for finite-state automata, expecially by using an explicit approach. While explicit model checking is not possible while treating with infinite state systems, new techniques like showing the existence of cutoffs are proving themselves able to overcome this problem.

A simple yet very relevant case is the token-passing rings one, in which several concurrent processes disposed in circle exchange information through the clockwise passage of a token.

The thesis is divided in two parts. In the first part we analyze the problem of finding cut-off theorems in the case of token-passing rings seen as labeled transition systems. Main results are given by the works of Emerson and Namjoshi [1995, 2003].

In the second part we aim to transpose the problem in the SMT (Satisfiability Modulo Theories) formalism using the array-based systems structure, introduced by Ghilardi and Ranise [2010]. This new approach presents remarkable advantages. In fact, by establishing a decidability procedure obtained through the application of formal rules, the theory of token-passing rings can be potentially encoded in any SMT-solver. The main result achieved in the second part, obtained in two different ways, is the proof of the decidability of the

satisfiability problem for the fragment of the theory of arrays over finite rings containing all the formulas of the form $\exists \underline{y} \forall \underline{x} A$. The first way consists in encoding the problem into monadic second order logic of one-successor function S1S. This approach, while being mathematically very solid, is rather weak from the complexity viewpoint. In the second proof we sketch a step-by-step procedure, which better adapts to the modular structure of an SMT solver, by being more direct.

The thesis is structured as follows:

The first chapter contains the definition of Labeled Transition System and the description of its basic operations. Moreover, a proof of the associativity for composition is presented.

Temporal logics, a founding piece in the field of model checking, are introduced in the second chapter. After a brief description of the most commonly used temporal logics, as CTL, CTL* and LTL, a fair amount of space is reserved for presenting indexed temporal logics, a powerful framework while dealing with parameterized verification.

The third chapter contains the construction of the token-passing rings as labeled transition systems plus the proof of two crucial theorems necessary to obtain cutoff results. Stuttering bisimulation is a key concept for comparing similar labeled transition systems (e.g. two token-passing rings having different dimensions) between them: in fact, the valid formulas in stuttering equivalent systems are closely related, as proved in Thm 3.1. The other essential result is the Reduction theorem, which provides the construction of a stuttering equivalence between quotients of token-passing rings having different dimension.

In the fourth chapter the parameterized model checking problem is presented and several cutoffs theorems are proved. Furthermore, a brief overview of the state-of-the-art of parameterized verification is given, summarizing the most relevant decidability and undecidability results for richer topologies than the unidirectional rings one. Finally, we show how token-passing rings find applications in several important *real-world* scenarios.

In the fifth chapter we present some logic preliminaries and introduce the central notions of array-based system and backward reachability. First, we define the theory of arrays: a three-sorted theory, defined over a (mono-sorted) index theory and a (multi-sorted) element theory. Then we formally describe token-passing rings as array-based systems: the element theory is a theory having two enumerated sorts, and the index theory is the theory of finite rings. Being not locally finite, this index theory leads to a particularly interesting case. In fact, the correctness of the backward reachability algorithm requires the decidability of the fragment $\exists^*\forall^*$ of the theory of arrays. While this result can be easily obtained having a locally finite index theory, for our case we need to explore new paths. The next chapters are dedicated to proving the decidability of the fragment $\exists^*\forall^*$ in two different ways.

The first proof is based on encoding the theory into the monadic second

order logic (MSOL) of one-successor function S1S. By doing so, we reduce to the decidability of S1S, which is a well-known result. In particular, this decision procedure involves associating to a S1S formula a Büchi automaton and then checking whether its language is not empty. However, finding this Büchi automaton can be potentially very bad computationally speaking, especially when having to construct complements. This leads us to try for a more direct proof, treated in the last two chapters.

In the seventh chapter we deal with the theory of finite rings without arrays. First, we provide a decision procedure for the satisfiability problem of finite sets of literals. Then we show a procedure to construct an equisatisfiable quantifier free formula from a formula with quantifiers, which leads to the decidability in the general case for the $\exists^*\forall^*$-fragment.

In chapter eight we propose a decidability proof for the $\exists^*\forall^*$-fragment of the theory of arrays over finite rings, which is based on the application of formal transformation rules. First, we show a proof for the case with no universal quantifiers, similarly to what did in chapter seven. Then we see how to deal with a single universal quantifier, first in the case without constants and then in the one with constants. Finally, we sketch a proof for the general case (this proof must be seen as a work-in-progress). The underlying idea of is to use the *mosaic method*. The mosaic method consists in showing that the existence of a model is tied with the existence of mosaics (that can be thought as small potential pieces of model) well linked between each other. Indeed, we manipulate the starting formula to show either a set of mosaics from which a real model can be constructed, or that the formula is unsatisfiable.

# Ringraziamenti

Chi mi conosce sa che in queste occasioni non sono capace di trovare le parole giuste, che qualsiasi loro combinazione mi sembra appiattire ciò che vorrei comunicare. Sono quindi certo che nessuno ne avrà male se questa sezione sarà la più scarna della tesi (per me, senza dubbio, è stata anche la più complicata!). Un primissimo grazie va ai miei genitori: la mia infinita e molesta curiosità è sicuramente e in gran parte merito e colpa loro. A Silvio Ghilardi va un ringraziamento speciale: oltre ad aver fatto nascere in me la passione per il ragionamento automatico, la sua enorme pazienza e disponibilità sono stati elementi imprescindibili per la buona riuscita di questo lavoro. Infine, a chi più di tutti mi ha supportato e sopportato in questi impegnativi mesi di stesura della tesi, va l'ultimo enorme ringraziamento piccolissimo.

# Part I
# Token-Passing Rings as Labeled Transition Systems

## 1 Labeled Transition Systems (LTS)

### 1.1 Definition

**Definition 1.1** (LTS)**.** *A Labeled Transition System (LTS) $A$ is a 6-tuple $(Q, \Sigma, \delta, \lambda, L, I)$ where*

- *$Q$ is a non-empty set of states*
- *$\Sigma$ is an alphabet of transitions (called also actions) which contains a special symbol $\tau$ (called the silent action)*
- *$\delta \subseteq Q \times \Sigma \times Q$ is a transition relation such that $\forall q \in Q \ (q, \tau, q) \in \delta$*
- *$L$ is a non-empty set of labels of the form $2^{AP}$, where $AP$ is a set of atomic propositions*
- *$\lambda : Q \mapsto \mathcal{P}(L)$ is a labelling function*
- *$I \subseteq Q$ is a set of initial states.*

We write $s \xrightarrow{a}_A t$ instead of $(s, a, t) \in \delta$, and $s \rightarrow_A t$ instead of $(\exists a \in A : (s, a, t) \in \delta)$. Moreover, $A$ can be dropped when it is clear which LTS we are referring to.

**Definition 1.2** (Path)**.** *A path $p$ on an LTS $A$ is a pair $(\sigma, \alpha)$ where: $\sigma$ is a sequence of states $(s_0, s_1, ...)$, $\alpha$ is a sequence of actions $(a_0, a_1, ...)$, $|\sigma| = |\alpha|+1$, and, $\forall i \leqslant |\sigma|, \ \ s_i \xrightarrow{a_i}_A s_{i+1}$.*

The lenght of a path $p = (\sigma, \alpha)$ is defined as $|p| := |\sigma|$.

**Definition 1.3** (Fullpath)**.** *A path $p$ is said to be a fullpath iff either it is infinite ($|p| = \infty$), or its last state $s_n$ is terminal (that is, for all $a \in \Sigma$ s.t $a \neq \tau$, and for all $q \in Q$, $(s_n, a, q) \notin \delta$).*

**Definition 1.4** (Execution)**.** *A path $p$ is an execution (or computation) iff it is a fullpath and its first state $s_0$ is an initial state ($s_0 \in I$).*

**Definition 1.5** (Isomorphism of LTS)**.** *Two LTS $A$ and $B$ are said isomorphic iff all the following conditions are satisfied at the same time:*

1. *There is a bijection $f_Q : Q_A \to Q_B$ s.t. $f_Q(I_A) = I_B$*
2. *There is a bijection $f_\Sigma : \Sigma_A \to \Sigma_B$ s.t. $f_\Sigma(\tau_A) = \tau_B$*
3. *For all $q, q' \in Q_A$ and for all $a \in \Sigma_A$: $q \xrightarrow{a}_A q'$ iff $f_Q(q) \xrightarrow{f_\Sigma(a)}_B f_Q(q')$*
4. *There is a bijection $f_L : L_A \to L_B$ s.t. $f_L(\lambda_A(q)) = \lambda_B(f_Q(q))$ for all $q \in Q$.*

## 1.2 Composition

Given two LTS $A = (Q_A, \Sigma_A, \delta_A, \lambda_A, L_A, I_A)$ and $B = (Q_B, \Sigma_B, \delta_B, \lambda_B, L_B, I_B)$, and a bijective partial function $\Gamma : \Sigma_A \backslash \{\tau_A\} \mapsto \Sigma_B \backslash \{\tau_B\}$, it is possible to define the composed LTS $A||_\Gamma B$ (also noted $AB$ when $\Gamma$ is clear, with an abuse of notation).
In order to do that let's first introduce some notation:

- $\Sigma^\Gamma_{I_A} := \Sigma_A \backslash dom(\Gamma) \backslash \{\tau_A\}$ (actions *internal* on A)
- $\Sigma^\Gamma_{I_B} := \Sigma_B \backslash Im(\Gamma) \backslash \{\tau_B\}$ (actions *internal* on B)
- $\Sigma^\Gamma_S := \{(a, b) \in \Sigma_A \times \Sigma_B : \Gamma(a) = b\}$ i.e. $\Sigma_\Gamma$ is the graph of $\Gamma$ (*synchronized* actions)

Note that all the sets are defined with respect to $\Gamma$.

**Definition 1.6.** *Given two LTS $A$, $B$, and a bijective partial function $\Gamma : \Sigma_A \backslash \{\tau\} \mapsto \Sigma_B \backslash \{\tau\}$, the composed LTS $A||_\Gamma B = (Q_{AB}, \Sigma_{AB}, \delta_{AB}, \lambda_{AB}, L_{AB}, I_{AB})$ is defined as:*

- $Q_{AB} = Q_A \times Q_B$
- $\Sigma_{AB} = \Sigma^\Gamma_{I_A} \bigsqcup \Sigma^\Gamma_{I_B} \bigsqcup \Sigma^\Gamma_S \bigsqcup \{\tau_{AB}\}$
- *$\delta_{AB}$ is defined by: $(s, t) \xrightarrow{c}_{AB} (u, v)$ holds iff one of the following holds:*
  - *(a)* $c \in \Sigma^\Gamma_{I_A} \ \wedge \ s \xrightarrow{c}_A u \ \wedge \ t = v$
  - *(b)* $c \in \Sigma^\Gamma_{I_B} \ \wedge \ s = u \ \wedge \ t \xrightarrow{c}_B v$
  - *(c)* $c = (a, b) \in \ \Sigma^\Gamma_S \ \wedge \ s \xrightarrow{a}_A u \ \wedge \ t \xrightarrow{b}_B v$
  - *(d)* $c = \tau_{AB} \ \wedge \ s \xrightarrow{\tau_A}_A u \ \wedge \ t \xrightarrow{\tau_B}_B v$
- $L_{AB} = L_A \bigsqcup L_B$
- $\lambda_{AB}(s, t) = \lambda_A(s) \bigsqcup \lambda_B(t)$
- $I_{AB} = I_A \times I_B$

### 1.2.1 Associativity

**Proposition 1.** *Let $A$, $B$ and $C$ be three LTS, and let $A||_{\Gamma_1}B$ and $B||_{\Gamma_2}C$ be two compositions. If $Im(\Gamma_1) \cap dom(\Gamma_2) = \varnothing$ then $(A||_{\Gamma_1}B)||_{\Gamma_2}C$ is isomorphic to $A||_{\Gamma_1}(B||_{\Gamma_2}C)$.*

*Proof.* Let's use the abbreviation $(AB)C$ for $(A||_{\Gamma_1}B)||_{\Gamma_2}C$ and $A(BC)$ for $A||_{\Gamma_1}(B||_{\Gamma_2}C)$.

- *The compositions are well-defined:* It is not obvious that we can write $(A||_{\Gamma_1}B)||_{\Gamma_2}C$, since the domain of $\Gamma_2$ is a subset of $\Sigma_B\backslash\{\tau_B\}$ and not a subset of $\Sigma_{AB}\backslash\{\tau_{AB}\}$ as it should be.
  Remember that $\Sigma_{AB}\backslash\{\tau_{AB}\} = \Sigma_{I_A}^{\Gamma_1} \bigsqcup \Sigma_{I_B}^{\Gamma_1} \bigsqcup \Sigma_S^{\Gamma_1}$, where $\Sigma_{I_B}^{\Gamma_1} = \Sigma_B\backslash Im(\Gamma_1)\backslash\{\tau_B\}$.
  By hypothesis, $Im(\Gamma_1)\cap dom(\Gamma_2) = \varnothing$, hence $dom(\Gamma_2) \subseteq \Sigma_{I_B}^{\Gamma_1} \subseteq \Sigma_{AB}\backslash\{\tau_{AB}\}$.
  This means it is possible to see $\Gamma_2$ as a partial function with domain in $\Sigma_{AB}\backslash\{\tau_{AB}\}$, therefore the first composition is well defined.
  A similar reasoning can be done for the other composition, showing that $\Gamma_1$ is well defined as a partial function with image in $B||_{\Gamma_2}C$.

- A bijection for the sets of states ($Q$ and $I$) arise naturally by the associativity of the cartesian product. In fact, as bijection we can take the identity. Thus condition *1.* of the definition of isomorphism is satisfied.

- A bijection for the sets of labels ($L$) arise naturally by the associativity of the disjoint union: again, we can take the identity. Trivially, condition *4.* is satisfied.

- In order to show that there is a trivial bijection for the sets of actions ($\Sigma$), therefore proving condition *2.*, some annoying calculations are needed, but it is just about applying the definitions given.

$$\Sigma_{(AB)C} = \Sigma_{I_{AB}}^{\Gamma_2} \bigsqcup \Sigma_{I_C}^{\Gamma_2} \bigsqcup \Sigma_S^{\Gamma_2} \bigsqcup \{\tau_{(AB)C}\} \tag{1}$$

$$\begin{aligned}
\Sigma_{I_{AB}}^{\Gamma_2} &= \Sigma_{AB} \setminus dom(\Gamma_2) \setminus \{\tau_{AB}\} \\
&= (\Sigma_{I_A}^{\Gamma_1} \bigsqcup \Sigma_{I_B}^{\Gamma_1} \bigsqcup \Sigma_S^{\Gamma_1} \bigsqcup \{\tau_{AB}\}) \setminus dom(\Gamma_2) \setminus \{\tau_{AB}\}) \\
&= \Sigma_{I_A}^{\Gamma_1} \bigsqcup (\Sigma_{I_B}^{\Gamma_1} \setminus dom(\Gamma_2)) \bigsqcup \Sigma_S^{\Gamma_1} \\
&= \Sigma_{I_A}^{\Gamma_1} \bigsqcup [(\Sigma_B \setminus Im(\Gamma_1) \setminus \{\tau_B\}) \setminus dom(\Gamma_2)] \bigsqcup \Sigma_S^{\Gamma_1} \\
&= \Sigma_{I_A}^{\Gamma_1} \bigsqcup (\Sigma_B \setminus Im(\Gamma_1) \setminus dom(\Gamma_2) \setminus \{\tau_B\}) \bigsqcup \Sigma_S^{\Gamma_1}
\end{aligned} \tag{2}$$

Substituting the last member of (2) in the right side of (1) we get:

$$\begin{aligned}
\Sigma_{(AB)C} = \Sigma_{I_A}^{\Gamma_1} &\bigsqcup (\Sigma_B \setminus Im(\Gamma_1) \setminus dom(\Gamma_2) \setminus \{\tau_B\}) \\
&\bigsqcup \Sigma_S^{\Gamma_1} \bigsqcup \Sigma_{I_C}^{\Gamma_2} \bigsqcup \Sigma_S^{\Gamma_2} \bigsqcup \{\tau_{(AB)C}\}
\end{aligned} \tag{3}$$

Let's do the same for $(AB)C$:

$$\Sigma_{A(BC)} = \Sigma_{I_A}^{\Gamma_1} \bigsqcup \Sigma_{I_{BC}}^{\Gamma_1} \bigsqcup \Sigma_S^{\Gamma_1} \bigsqcup \{\tau_{A(BC)}\} \tag{4}$$

$$\begin{aligned}
\Sigma_{I_{BC}}^{\Gamma_1} &= \Sigma_{BC} \setminus Im(\Gamma_1) \setminus \{\tau_{BC}\} \\
&= (\Sigma_{I_B}^{\Gamma_2} \bigsqcup \Sigma_{I_C}^{\Gamma_2} \bigsqcup \Sigma_S^{\Gamma_2} \bigsqcup \{\tau_{BC}\}) \setminus Im(\Gamma_1) \setminus \{\tau_{BC}\}) \\
&= (\Sigma_{I_B}^{\Gamma_2} \setminus Im(\Gamma_1)) \bigsqcup \Sigma_{I_C}^{\Gamma_2} \bigsqcup \Sigma_S^{\Gamma_2} \\
&= [(\Sigma_B \setminus dom(\Gamma_2) \setminus \{\tau_B\}) \setminus Im(\Gamma_1)] \bigsqcup \Sigma_{I_C}^{\Gamma_2} \bigsqcup \Sigma_S^{\Gamma_2} \\
&= (\Sigma_B \setminus Im(\Gamma_1) \setminus dom(\Gamma_2) \setminus \{\tau_B\}) \bigsqcup \Sigma_{I_C}^{\Gamma_2} \bigsqcup \Sigma_S^{\Gamma_2}
\end{aligned} \tag{5}$$

Substituting the last member of (5) in the right side of (4) we get:

$$\begin{aligned}
\Sigma_{A(BC)} = \Sigma_{I_A}^{\Gamma_1} \bigsqcup (\Sigma_B \setminus Im(\Gamma_1) \setminus dom(\Gamma_2) \setminus \{\tau_B\}) \\
\bigsqcup \Sigma_{I_C}^{\Gamma_2} \bigsqcup \Sigma_S^{\Gamma_2} \bigsqcup \Sigma_S^{\Gamma_1} \bigsqcup \{\tau_{A(BC)}\}
\end{aligned} \tag{6}$$

Identifying $\tau_{A(BC)}$ with $\tau_{(AB)C}$, and up to reordering, the RHS of (3) is equal to the RHS of (6): $\Sigma_{(AB)C} = \Sigma_{A(BC)}$ (thus is legit to write $\Sigma_{ABC}$). It remains to show that these bijetions satisfy condition *3.*

- Again, the proof is a bit long but there is no difficult concepts beyond it, it is all about applying the definitions.
  Suppose $d \in \Sigma_{ABC}$: we have to prove that $(s,t,u) \xrightarrow{d}_{(AB)C} (v,w,z)$ iff $(s,t,u) \xrightarrow{d}_{A(BC)} (v,w,z)$, for all $(s,t,u) \in Q_{ABC}$ and $(v,w,z) \in Q_{ABC}$. Let's procede by case:

  - $d \in \Sigma_{I_A}^{\Gamma_1}$: $(s,t,u) \xrightarrow{d}_{(AB)C} (v,w,z)$
    iff $(s,t) \xrightarrow{d}_{AB} (v,w) \ \wedge \ u = z$
    iff $(s \xrightarrow{d}_{A} v \ \wedge \ t = w) \ \wedge \ u = z$
    iff $s \xrightarrow{d}_{A} v \ \wedge \ (t,u) = (w,z)$
    iff $(s,t,u) \xrightarrow{d}_{A(BC)} (v,w,z)$

  - $d \in \Sigma_{I_C}^{\Gamma_2}$: $(s,t,u) \xrightarrow{d}_{(AB)C} (v,w,z)$
    iff $(s,t) = (v,w) \ \wedge \ u \xrightarrow{d}_{C} \mathrm{z}$
    iff $s = v \ \wedge \ (t = w \ \wedge \ u \xrightarrow{d}_{C} \mathrm{z})$
    iff $s = v \ \wedge \ (t,u) \xrightarrow{d}_{BC} (\mathrm{w},\mathrm{z})$
    iff $(s,t,u) \xrightarrow{d}_{A(BC)} (v,w,z)$

  - $d \in (\Sigma_B \setminus Im(\Gamma_1) \setminus dom(\Gamma_2) \setminus \{\tau_B\})$: $(s,t,u) \xrightarrow{d}_{(AB)C} (v,w,z)$
    iff $(s,t) \xrightarrow{d}_{AB} (v,w)) \ \wedge \ u = z$

iff $(s = v \ \wedge \ t \xrightarrow{d}_B w) \ \wedge \ u = z$
iff $s = v \ \wedge \ (t \xrightarrow{d}_{B,C} w \ \wedge \ u = z)$
iff $s = v \ \wedge \ ((t,u) \xrightarrow{d}_B (w,z))$
iff $(s,t,u) \xrightarrow{d}_{A(BC)} (v,w,z)$

– $d = (a,b) \in \Sigma_S^{\Gamma_1}$: $(s,t,u) \xrightarrow{(a,b)}_{(AB)C} (v,w,z)$
iff $(s,t) \xrightarrow{(a,b)}_{AB} (v,w) \ \wedge \ u = z$
iff $s \xrightarrow{a}_A v \ \wedge \ t \xrightarrow{b}_B w \ \wedge \ u = z$
iff $s \xrightarrow{a}_A v \ \wedge \ ((t,u) \xrightarrow{b}_{BC} (w,z))$
iff $(s,t,u) \xrightarrow{(a,b)}_{A(BC)} (v,w,z)$

– $d = (b,c) \in \Sigma_S^{\Gamma_2}$: $(s,t,u) \xrightarrow{(b,c)}_{(AB)C} (v,w,z)$
iff $(s,t) \xrightarrow{b}_{AB} (v,w) \ \wedge \ u \xrightarrow{c}_C z$
iff $s = v \ \wedge \ t \xrightarrow{b}_B w \ \wedge \ u \xrightarrow{c}_C z$
iff $s = v \ \wedge \ ((t,u) \xrightarrow{(b,c)}_{BC} (w,z))$
iff $(s,t,u) \xrightarrow{(b,c)}_{A(BC)} (v,w,z)$

– $d = \tau_{ABC}$: $(s,t,u) \xrightarrow{\tau_{ABC}}_{(AB)C} (v,w,z)$
iff $(s,t) \xrightarrow{\tau_{AB}}_{AB} (v,w) \ \wedge \ u \xrightarrow{\tau_C}_C z$
iff $s \xrightarrow{\tau_A}_A v \ \wedge \ t \xrightarrow{\tau_B}_B w \ \wedge \ u \xrightarrow{\tau_C}_C z$
iff $s \xrightarrow{\tau_A}_A v \ \wedge \ ((t,u) \xrightarrow{\tau_{BC}}_{BC} (w,z))$
iff $(s,t,u) \xrightarrow{\tau_{ABC}}_{A(BC)} (v,w,z)$

□

## 1.3 Projection and silent action

**Definition 1.7.** *Given a LTS of the form $C = A||_\Gamma B$, its projection on $A$ is the LTS $C|_A = (Q, \Sigma, \delta, \lambda, L, I)$ where*

- $Q = Q_C$
- $\Sigma = \Sigma_A$
- *$\delta$ is defined by: $s \xrightarrow{a} t$ holds iff one of the following holds:*
  - *(a)* $a \in \Sigma^\Gamma_{I_A} \ \wedge \ s \xrightarrow{a}_C t$
  - *(b)* $a \in dom(\Gamma) \ \wedge \ s \xrightarrow{(a,\Gamma(a))}_C t$
  - *(c)* $a = \tau_A \ \wedge \ (\exists c \in (\Sigma^\Gamma_{I_B} \cup \{\tau_{AB}\}) : s \xrightarrow{c}_C t)$
- $L = L_A$
- $\lambda(s) = (\lambda_C(s))|_{L_A}$
- $I = I_C$

*The only relevant changes happening are that the internal actions of $B$ are being compressed into the silent action $\tau$, and the labels not in $A$ vanish. Synchronized actions are not compressed into $\tau$.*

Note: dealing with an associative composition of LTS of the form $C = A_1 \ ||_{\Gamma_1} \ A_2 \ ||_{\Gamma_2} \ ... \ ||_{\Gamma_{n-1}} \ A_n$, given a set of indices $I = \{i_1, \ ..., \ i_m\} \subseteq [n]$ it is possible to project $C$ on the processes indexed by elements in $I$. Indeed, this sort of "group projection" is nothing but the result of the application step by step of the single projections on $A_{i_1}$, $A_{i_2}$, ... , $A_{i_m}$. Briefly, by $C|_I$ we indicate $(((C|_{A_{i_1}})|_{A_{i_2}})...)|_{A_{i_m}}$. Doing so without worries is possible only thanks to the associativity of the composition.

This kind of operation will play an important role while studying token passing rings. Indeed, in token passing rings it is really useful to project $Ring_n$ on a subset $I$ of the set of the indices: in such case, we project $Ring_n$ onto the processes indexed by the elements of $I$ (see chapter 3 for the details).

# 2 Temporal Logics

## 2.1 CTL*

**Syntax.** The temporal logic $CTL^*$ has two types of formulas: state formulas and path formulas. These formulas are defined over a set of atomic propositions $AP$ as follows:

S1) If $P$ is an atomic proposition, then $P$ is a state formula.

S2) If $f$ is a state formula, then $\neg f$ is a state formula.

S3) If $f$ and $g$ are state formulas, then $f \wedge g$ is a state formula.

S4) If $f$ is a path formula, then $Ef$ is a state formula ($E$ stands for "exists a future such that").

S5) If $f$ is a path formula, then $Af$ is a state formula ($A$ stands for "for all the futures").

P1) If $f$ is a state formula, then $f$ is also a path formula.

P2) If $f$ is a path formula, then $\neg f$ is a path formula.

P3) If $f$ and $g$ are path formulas, then $f \wedge g$ is a path formula.

P4) If $f$ and $g$ are path formulas, then $f\ U_w\ g$ is a path formula ($U_w$ stands for "weak until").

P5) If $f$ is a path formula, then $X_s f$ is a path formula ($X_s$ stands for "strong next time").

It can be useful to introduce some derived operators as abbreviations, in order to improve the readability of a formula. We can define the usual logic connectors:

- $f \vee g \ \equiv\ \neg(\neg f \wedge \neg g)$
- $f \rightarrow g \ \equiv\ \neg f \vee g$
- $f \leftrightarrow g \ \equiv\ f \rightarrow g \wedge g \rightarrow f$

and some new symbols:

- $Gf \ \equiv\ f\ U_w\ false$ ($G$ stands for "globally", or "always")
- $Ff \ \equiv\ \neg(G\neg f)$ ($F$ stands for "eventually in the future")
- $f\ U_s\ g \ \equiv\ (f\ U_w\ g) \wedge Fg$ ($U_s$ stands for "strong until")
- $X_w f \ \equiv\ \neg(X_s \neg f)$ ($X_w$ stands for "weak next time")

**Semantics.** Now we procede to define the semantics of the formulas constructed above, with respect to an LTS $A = (Q, \Sigma, \delta, \lambda, 2^{AP}, I)$.

For a state formula $f$, we write $A, s \models f$ (or just $s \models f$ when $A$ is clear), to mean that the state formula $f$ is *true* in $A$ at the state $s$.

For a path formula $f$, we write $A, p \models f$ (or just $p \models f$ when $A$ is clear) to mean that the path formula $f$ is *true* in $A$ w.r.t the fullpath $p$.

The notion of *truth* is inductively defined as follows:

S1): $s \models P$ iff $P \in \lambda(s)$

S2): $s \models \neg f$ iff not $s \models f$

S3): $s \models f \wedge g$ iff $s \models f$ and $s \models g$

S4): $s \models Ef$ iff there exists a fullpath $p$ starting at $s$ such that $p \models f$

S5): $s \models Af$ iff for every fullpath $p$ starting at $s$, it holds that $p \models f$

P1): $p \models f$ iff $s_0 \models f$, where $s_0$ is the first state of $p$

P2): $p \models \neg f$ iff not $p \models f$

P3): $p \models f \wedge g$ iff $p \models f$ and $p \models g$

P4): $p \models f \; U_w \; g$ iff for all $i < |p|$, either exists $j \leqslant i$ such that $s_j \models g$, or $s_i \models f$

P5): $p \models X_s f$ iff the second state of the path $s_1$ exists and $s_1 \models f$

*Note*: it is not necessary to include $A$ among the primitive operators. Indeed, it could be defined as an abbreviation: $Af \equiv \neg E \neg f$, observing that this abbreviation is consistent with the notions of truth S4 and S5 defined above. However, including it among the primitive operators brings the remarkable advantage that it is way easier to define $CTL$ as a sublanguage of $CTL^*$ (see subsection below).

Now, one should note that the actions labeling does not play any role in the semantics of $CTL^*$ formulas. In fact, the description of $CTL^*$ formulas presented above suits as well unlabeled transition systems. It is therefore possible to enrich our logic by adding, for every action $a$, a new operator $X_s^a$:

P5a) If $f$ is a path formula and $a$ is an action, then $X_s^a f$ is a path formula.

This new temporal logic is called $CTL^*_\Sigma$, and its semantics is an extension of the $CTL^*$ one with the addition of:

P5a): $p \models X_s^a f$ iff there exists $i < |p|$ such that for every $j < i$, the $j$-th action is $a_j = \tau$, and the state $s_{i+1}$ exists and $s_{i+1} \models f$

## 2.2 Computational Tree Logic

$CTL^*$ takes its name from $CTL$ (*Computational Tree Logic*), of which it is an extension. In fact, $CTL$ can be defined as a fragment of $CTL^*$ by replacing $P1 - P5$ with:

$\tilde{P}4$) If $f$ and $g$ are state formulas, then $f\ U_w\ g$ is a path formula

$\tilde{P}5$) If $f$ is a state formula, then $X_s f$ is a path formula

The language $CTL$ can be defined as the set of formulas generated by $S1 - S5$, $\tilde{P}4$, $\tilde{P}5$.

**Proposition 2.** *$CTL$ is a sublanguage of $CTL^*$.*

*Proof.* First, note that every state formula of $CTL$ is in $CTL^*$, since the rules $S1 - S5$ are shared by both the languages.

Every $CTL$-formula generated by $\tilde{P}4$ is in $CTL^*$: if $f$ is a $CTL$-state formula, it is also a $CTL^*$-state formula. By $P1$, it is a $CTL^*$-path formula, and by $P4$, $f\ U_w\ f$ is a $CTL^*$-path formula.

The same argument can be made for formulas generated by $\tilde{P}5$. □

## 2.3 Linear Temporal Logic

An important fragment of $CTL^*$ is $LTL$ (*Linear Temporal Logic*). It consists of all the $CTL^*$ formulas of the form $Af$, where $A$ is the universal path quantifier and $f$ is a $CTL^*$ formula not containing any path quantifier $E$ or $A$.

## 2.4 Indexed Temporal Logics

The idea of indexed temporal logics was introduced by Browne, Clarke, and Grumberg in 1989 [3], with the aim to provide a convenient temporal logic for concurrent systems. In fact, adding the possibility to quantify over indices is a good strategy to improve the expressiveness of the temporal logic we are working on (be it $CTL^*$, $LTL$, or others).

For example, without index quantifiers, in order to express the property "there is a process in a critical section" for a system with $n$ processes, one should write $\bigvee_{i \in [n]} critical_i$. Since this formula is parameterized by $n$, one should write different formulas for similar systems with a different number of processes. By allowing to quantify over indices, the same property can be expressed for similar systems by the formula $\exists i : critical_i$. Of course, the semantics will be different from system to system, but having the same syntax will be very useful.

We give below a definition of indexed $CTL^*$ (also called $ICTL^*$), from which it is possible to obtain $ILTL$ or $ICTL$ as fragments.

In the following, by $I$ we indicate an infinite set of index variables. The semantics will be defined over a colored graph, consisting of a set of vertices $V$, a set of edges $\mathcal{E} \subseteq V \times V$, and a function $\texttt{type} : V \to \texttt{Type}$[1]. If $|\texttt{type}(V)| = d$, we say that the graph is a $d$-colored graph, and could use $[d]$ instead of $\texttt{Type}$. By $i \in \mathcal{E}(j)$ we mean $(i, j) \in \mathcal{E}$.
A *range expression* $r$ is an expression that can have one of the following forms:

a) $neq(i_1, ..., i_k)$

b) $i \in \mathcal{E}(j)$

c) $\texttt{type}(i) = t$

d) $\texttt{true}$

The formulas are defined over a set of atomic propositions $AP$.

**Syntax**:

I-S1) If $P$ is an atomic proposition and $i \in I$, then $P_i$ is a state formula.

S2) If $f$ is a state formula, then $\neg f$ is a state formula.

S3) If $f$ and $g$ are state formulas, then $f \wedge g$ is a state formula.

S4) If $f$ is a path formula, then $Ef$ is a state formula.

S5) If $f$ is a path formula, then $Af$ is a state formula.

[1] Here, by $\texttt{Type}$ we indicate the set of colors in the common language of graphs.

I-S6) If $f$ is a state formula and $r$ is a range expression, then $\forall i : r : f$ is a state formula.

I-S7) If $f$ is a state formula and $r$ is a range expression, then $\exists i : r : f$ is a state formula.

P1) If $f$ is a state formula, then $f$ is also a path formula.

P2) If $f$ is a path formula, then $\neg f$ is a path formula.

P3) If $f$ and $g$ are path formulas, then $f \wedge g$ is a path formula.

P4) If $f$ and $g$ are path formulas, then $f\ U_w\ g$ is a path formula.

P5) If $f$ is a path formula, then $X_s f$ is a path formula.

I-P6) If $f$ is a path formula and $r$ is a range expression, then $\forall i : r : f$ is a path formula.

I-P7) If $f$ is a path formula and $r$ is a range expression, then $\exists i : r : f$ is a path formula.

When $r \equiv \texttt{true}$, instead of $(\forall i : r : f)$ we will write $(\forall i : f)$, and instead of $(\exists i : r : f)$ we will write $(\exists i : f)$

**Semantics.** Now we define the semantics of the formulas constructed above, interpreted over a composed LTS $A = T_1 \,||\, ... \,||T_k$, having associated a graph $G = (V, \mathcal{E}, \texttt{type})$ defined as:

- $V = \{1, ..., k\}$
- $\texttt{type}(i) = \texttt{type}(j)$ iff $T_i$ is isomorphic to $T_j$
- $(i, j) \in \mathcal{E}$ iff $\Sigma_A$ contains a synchronized action between $T_i$ and $T_j$.

and over a *valuation* $e : I \to V$.
A valuation $e$ is said to satisfy the range expression $r$ iff:

1. $r \equiv neq(i_1, ...i_m)$ and $e(i_1), ..., e(i_m)$ are pairwise distinct
2. $r \equiv \texttt{type}(i) = t$ and $\texttt{type}(e(i)) = t$
3. $r \equiv i \in \mathcal{E}(j)$ and $(e(i), e(j)) \in \mathcal{E}$.
4. $r \equiv \texttt{true}$

Moreover, a valuation $e$ is said an *i-variant* of a valuation $e'$ iff $e(j) = e'(j)$ for all $j \in I \backslash \{i\}$.

For an $A$-state $s = (s_1, ..., s_k)$ or a path $p$, the notions of *truth* $A, s, e \models f$ and $A, p, e \models f$ are defined inductively by ($A$ can be dropped when it is clear):

I-S1): $s, e \models P_i$ iff $P \in \lambda_{T_{e(i)}}(s_{e(i)})$

S2): $s, e \models \neg f$ iff not $s, e \models f$

S3): $s, e \models f \wedge g$ iff $s, e \models f$ and $s, e \models g$

S4): $s, e \models Ef$ iff there exists a fullpath $p$ starting at $s$ such that $p, e \models f$

S5): $s, e \models Af$ iff for every fullpath $p$ starting at $s$, it holds that $p, e \models f$

I-S6): $s, e \models \forall i : r : f$ iff for every *i-variant* $e'$ that satisfies $r$, it holds $s, e' \models f$

I-S7): $s, e \models \exists i : r : f$ iff there exists an *i-variant* $e'$ that satisfies $r$ such that $s, e' \models f$ holds

P1): $p, e \models f$ iff $s_0, e \models f$, where $s_0$ is the first state of $p$

P2): $p, e \models \neg f$ iff not $p, e \models f$

P3): $p, e \models f \wedge g$ iff $p, e \models f$ and $p, e \models g$

P4): $p, e \models f \; U_w \; g$ iff for all $k < |p|$, either exists $k' \leqslant k$ such that $s_{k'}, e \models g$, or $s_k, e \models f$

P5): $p, e \models X_s f$ iff the second state of the path $s_1$ exists and $s_1, e \models f$

I-S6): $p, e \models \forall i : r : f$ iff for every *i-variant* $e'$ that satisfies $r$, it holds $p, e' \models f$

I-S7): $p, e \models \exists i : r : f$ iff there exists an *i-variant* $e'$ that satisfies $r$ such that $p, e' \models f$ holds

Let's define $A, s \models f$ if for all valuations $e : I \to V$ it holds that $A, s, e \models f$, and $A, p \models f$ if for all valuations $e : I \to V$ it holds that $A, p, e \models f$. Finally, define:

$$A \models f \text{ iff for every initial state } s \text{ of } A, \text{ it holds that } A, s \models f$$

Let's now introduce some terminology. An index variable $i$ and an atom $P_i$ are said to be *bound* if they are in the scope of an index quantifier $\forall i : r$ or $\exists i : r$. A formula is a *sentence* if every atom in it is bound. With $f(i_1, ...i_k)$ we indicate a formula in which $i_1, ..., i_k$ are not bound (they are also said to be *free*).

Moreover, for $c_1, ..., c_k \in V$, we write $A, s \models f(c_1, ..., c_k)$ if for every valuation $e$ that maps $i_j$ to $c_j$ (for $j = 1, ..., k$) it holds that $A, s, e \models f(i_1, ..., i_k)$.

As done with $CTL^*$, it is possible to introduce the operator $X_s^a$ by adding $P5a$) as in the chapter 2.1, with an analogue interpretation.

# 3 Token passing rings as LTS

## 3.1 Definition of token passing ring

A token passing ring (TPR) consists of $n$ isomorphic LTS $K_i$ arranged in an unidirectional ring $Ring_n$ (to be properly defined). Each $K_i$ is constructed from a process template $P$, i.e. each $K_i$ is an isomorphic copy of an LTS $P$, and every state, every transition and every label of $K_i$ is indexed by $i$.

More specifically, we require $P$ to be a *token-passing* process template, i.e. it satisfies the following properties:

- The set of the states $Q_P$ is partitioned into two non empty sets $T$ and $N$. The states in $T$ are said to have the token, while the states in $N$ are said not to have the token.
- Every initial state does not have the token. That is, $I_P \subseteq N$.
- $P$ has three types of actions: the "sending the token" action $snd$, the "receiving the token" action $rcv$, and some *internal* actions $\Sigma^I$. That is, $\Sigma_P = \{snd, rcv\} \cup \Sigma^I$.
- Every transition of the form $q \xrightarrow{snd}_P q'$ requires that $q$ has the token and $q'$ has not.
- Every transition of the form $q \xrightarrow{rcv}_P q'$ requires that $q$ does not have the token and $q'$ does.
- Every transition of the form $q \xrightarrow{a}_P q'$ for $a \in \Sigma^I$ requires that either both $q$ and $q'$ have the token, or neither of them does.

Given a process template $P$ that satisfies these conditions, we want to define the token passing ring as a composition of the form $Ring_n := K_0||K_1||...||K_{n-1}||D_n$, where each $K_i$ is a process (constructed from the same process template $P$ and indexed by $i$), $D_n$ is the initial token distribution process that at the very beginning synchronizes with a non-deterministically chosen $K_i$ to pass it the token (then does nothing), and each composition $K_i||K_{i+1}$ synchronizes the $K_i$-action $snd_i$ with the $K_{i+1}$-action $rcv_{i+1}$.

In order to formally define $Ring_n$, we have to be a little pedantic. The idea is the following: compose the first $n-1$ processes in order to obtain a path graph, then compose the last process attaching it to the two extremities (obtaining a ring graph), and finally compose the initial token distribution process with the ring. In detail:

- Let's first define $R' := K_0||_{\Gamma_0}K_1||_{\Gamma_1}...||_{\Gamma_{n-3}}K_{n-2}$, where $\Gamma_i = \{(snd_i, rcv_{i+1})\}$ for $0 \leqslant i \leqslant n-3$. Since $Im(\Gamma_i) \cap dom(\Gamma_{i+1}) = \varnothing$ for all $i < n-3$, associativity holds, hence the definition is well posed. Informally, we could describe $R'$ as the path graph composed by the first $n-1$ processes.

- Now we have to *attach* the $n$-th process to the two extremities of the *string* to form a ring, so let's define $R'' := R'||_{\Gamma_{n-2}} K_{n-1}$, where $\Gamma_{n-2} = \{(snd_{n-2}, rcv_{n-1}), (snd_{n-1}, rcv_0)\}$.

- At this point we have the ring we were looking for, except for the fact that no process holds the token, which has to be assigned non deterministically by a distribution process $D_n$ at the very beginning.

- So let's define formally $D_n = (Q_D, \Sigma_D, \delta_D, \lambda_D, L_D, I_D)$, where:

  - $Q_D = \{init,\ end\}$.
  - $\Sigma_D = \{init\_snd_0,\ init\_snd_1,\ ..., init\_snd_{n-1},\ \tau\}$.
  - $I_D = \{init\}$.
  - $\delta_D = \{(q, \tau, q) : q \in Q_D\}\ \cup\ (\{init\} \times \{init\_snd_i\ :\ i \in I\} \times \{end\})$. That is, from every initial state, the process can either do nothing, or execute a token sending action and then terminate.

  – $L_D = \{0, 1\}$ (this can be seen as a *flag*)

  – $\lambda_D(q) = \begin{cases} \{0\}, & \text{if } q = init \\ \{1\}, & \text{if } q = end \end{cases}$

- At last, let's define $Ring_n := R''||_{\Gamma_D} D_n$, where $\Gamma_D = \{(rcv_i, init\_snd_i) : 0 \leqslant i \leqslant n-1\}$. At this point, we have constructed $Ring_n$ with the articulated form $Ring_n = ((K_0||_{\Gamma_0} K_1||_{\Gamma_1} ...||_{\Gamma_{n-3}} K_{n-2})||_{\Gamma_{n-2}} K_{n-1})||_{\Gamma_D} D_n$, which is completely formal.

For brevity, let's write $M_n$ instead of $Ring_n$. Given an index set $J \subseteq [n]$ let's write $M_{n|J}$ to indicate the projection over the processes indexed by $J$ (note that $D_n$ is not included among these processes).

It could be useful to write down a brief and informal description of $M_n$ and of $M_{n|J}$:

An **initial state of** $M_n$ is a (n+1)-uple of the form $(q_0, ..., q_{n-1}, init)$ where every $q_i$ is an initial local state of the process $K_i$ ($q_i \in I_{K_i}$), and $init$ is the initial state of $D_n$. Note that it is not said that the first action of $M_n$ has to be the token assignment. In fact, it could be any legal internal action of every process. It is possible to require the first action to be the token assignment by imposing conditions on the admissible executions, but since the theoretic results are not affected by the possibility of a non-token assignment first action, it is preferable to treat with the more general case.

Let's now consider the structure of $M_{n|J}$. The states of $M_{n|J}$ are the same of $M_n$. What really changes are the actions and the labels. Indeed, an **action of** $M_{n|J}$ can be:

- An internal action of a process $K_j$ ($j \in J$).

- A token passing action between two processes $K_j$ and $K_{j+1}$ (with $j, j+1 \in J$). This type of actions has the form $(snd_j, rcv_{j+1})$.

- A token passing action between two processes $K_j$ and $K_{j+1}$ where either $j$ or $j+1$ are in $J$, but not both. Looking at the definition of projection, this type of actions can have the form $snd_j$ (if $j \in J$) or $rcv_{j+1}$ (if $j+1 \in J$). However, it can be a little pedantic to distinguish between these two forms even when the only real change is the name, so we will always use the form $(snd_j, rcv_{j+1})$.
- A token passing action between two processes $K_j$ and $K_{j+1}$ such that neither $j$ nor $j+1$ is in $J$. This type of actions are $\tau$-actions.
- An initial token distribution from $D_n$ to $K_j$ ($j \in J$). Looking at the definition of projection, this type of actions has the form $rcv_j$, but we will accept also the form $(init\_snd_j, rcv_j)$ for simplicity.
- Initial token distribution from $D_n$ to $K_j$ ($j \notin J$). These actions are $\tau$-actions.

## 3.2 Fairness conditions

Working with token passing rings, we are interested in considering only *fair* executions, i.e. executions where every process receive and then send the token infinitely often.

There are two ways to do that. The first one is requiring that the structure of the process template $P$ is such that it ensures that the process will eventually pass the token. That can be done for example setting a maximum time after which the process forces itself to pass the token. If the process template has a finite number of states, it is possible to check with a model-checker that every execution of the token-passing ring is *fair*.

The second and more formal way is to use fairness constraints.

**Definition 3.1** (Fairness constraint)**.** *Given a LTS $A$, a fairness constraint is a finite set $F = \{S_1, ..., S_k\}$ whose elements are sets of $A$-states.*

*A path $(s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} ...)$ is said to be fair with respect to $F$ iff for every $j = 1, ..., k$ there are infinite $i \in \mathbb{N}$ such that $s_i \in S_j$.*

In particular, we can redefine the notion of *truth* for a $CTL^*$ formula $f$ in a state $s$ with respect to $F$ ($s \models_F f$):

S1) $s \models_F P$ iff $s \models P$

S2) $s \models_F \neg f$ iff $s \models \neg f$

S3) $s \models_F f \wedge g$ iff $s \models f \wedge g$

S4) $s \models_F Ef$ iff there exists a fair path $p$ starting at $s$ such that $p \models f$

P1) $p \models_F f$ iff $p \models f$

P2) $p \models_F \neg f$ iff $p \models \neg f$

P3) $p \models_F f \wedge g$ iff $p \models f \wedge g$

P4) $p \models_F X_s f$ iff the second state of the path $s_1$ exists and $s_1 \models_F f$

P5) $p \models_F f\ U_w\ g$ iff for all $i < |p|$, either exists $j \leqslant i$ such that $s_j \models_F g$, or $s_i \models_F f$

In general, note that the only meaningful change in the semantics is when an existential path quantifier is involved. If one wants to consider a richer temporal logic like $CTL^*_\Sigma$, $ICTL^*$ or $ICTL^*_\Sigma$, the redefinition of the semantics for the new formulas is intuitively similar to the one done above.

It is now possible to define a fairness constraint for token passing rings. Remember that, given a process template $P$, we indicate with $T$ the set of the $P$-states containing the token, and with $N$ the set of the $P$-states that do not contain the token. For every process $K_i$, the two sets are denoted by $T_i$ and $N_i$.

Let's write $\tilde{T}_i$ to indicate the set of the $M_n$-states such that the local state of the process $K_i$ is in $T_i$. Symmetrically, let's write $\tilde{N}_i$ to indicate the set of the $M_n$-states such that the local state of the process $K_i$ is in $N_i$.

It is clear that the fairness constraint that restricts the executions to the ones where every process receives the token infinitely often is $F = \{\tilde{T}_1, ..., \tilde{T}_n\}$.

In the following, we will always work with the notion of truth with respect to $F$. With an abuse of notation the $F$ in $\models_F$ will be dropped.

## 3.3 Stuttering bisimulation

The idea behind the definition of stuttering bisimulation is to have a tool which enables us to compare different instances of similar systems (ex.: two token passing rings of different sizes) which is indifferent to "stuttering" behaviours (like repetitions of states, or actions irrelevant for the purposes of our investigation). To do that we first introduce the concept of *stuttering sequence* on a LTS: informally, it is a path in which there are no meaningful changes except after the final transition (i.e. in every sequence there is only one relevant change, and it has to be at the end of it).

**Definition 3.2** (Stuttering sequence)**.** *A stuttering sequence on a LTS $A$ is a path $p = (\sigma, \alpha)$ starting at a state $s_0$ s.t.:*

1. *Every transition (except the final one, if the path is finite) is a $\tau$-action*
2. *Every state $s_i$ on the path (except possibly the final one) is s.t. $\lambda_A(s_0) = \lambda_A(s_i)$*

*Explicitly, a path $p = (\sigma, \alpha)$ is a stuttering sequence iff for all $i \in \mathbb{N}$ s.t. $0 \leqslant i < |\alpha| - 1$ we have $a_i = \tau$, $a_{|\alpha|-1} \neq \tau$, and $\lambda_A(s_0) = \lambda_A(s_i)$.*

The set of stuttering sequences starting at state $s$ and having $a$ as final transition is denoted by $Stutter(s, a)$. For the set of infinite stuttering sequences starting at state $s$, we write $Stutter(s, Inf)$.

**Definition 3.3** (Stuttering Bisimulation)**.** *Given two LTS $A$ and $B$, and a relation $R$ on $Q_A \bigsqcup Q_B$ having associated two bijective functions $h_L : L_A \mapsto L_B$ and $h_\Sigma : \Sigma_A \mapsto \Sigma_B$ (which map propositions and actions, respectively), we say that $R$ is a stuttering bisimulation iff:*

1. *Every initial state of $A$ is in relation with an initial state of $B$, and vice-versa*
2. *For every $s \in Q_A$ and $t \in Q_B$, $s\ R\ t$ implies that:*
   (a) *$h_L(\lambda_A(s)) = \lambda_B(t)$*
   (b) *For every non-silent action $a$ and for every path $p \in Stutter(s, a)$, there is a path $q \in Stutter(t, h_\Sigma(a))$ such that $p$ matches $q$ by $R$.*
   (c) *For every path $p \in Stutter(s, Inf)$, there is a path $q \in Stutter(t, Inf)$ such that $p$ matches $q$ by $R$.*

We say that two paths $p$ and $q$ *match* iff they can be partitioned into an equal number of non-empty finite segments $(p^{(1)}, ..., p^{(m)})$ and $(q^{(1)}, ..., q^{(m)})$ s.t. every state of $p^{(i)}$ is related to every state of $q^{(i)}$, for all $i = 1, ..., m$.
If there is a stuttering bisimulation between $A$ and $B$, they are said to be *stuttering equivalent* ($A \sim B$).

Stuttering equivalence is a powerful property. In fact, it gives a correspondence between valid formulas of two stuttering equivalent LTS.

First, note that, given two composed LTS of the form $A = T_1 \mid\mid ... \mid\mid T_k$ and $B = U_1 \mid\mid ... \mid\mid U_l$, where $T_i \sim U_j$ (for every $i \in [k]$ and for every $j \in [l]$), and a bijective partial function $h : [k] \to [l]$, it possible to obtain two induced functions $h_\Sigma : \Sigma_A \to \Sigma_B$ and $h_L : L_A \to L_B$ by applying $h$ to every index in every actions and in every labels. Moreover, for any formula $f \in ICTL^*_\Sigma \backslash X$, let $h(f)$ be the formula obtained by replacing every occurrence of each proposition $P_i$ by $P_{h(i)}$ and every occurrence of every action $a$ by $h_\Sigma(a)$. We have the following results:

**Theorem 3.1.** *Given two composed LTS of the form $A = T_1 \mid\mid ... \mid\mid T_k$ and $B = U_1 \mid\mid ... \mid\mid U_l$, where $T_i \sim U_j$ (for every $i \in [k]$ and for every $j \in [l]$), let $R$ be a stuttering bisimulation between them, with associated bijective functions $h^{AP}$ and $h^\Sigma$ constructed from a bijective partial function $h : [k] \to [l]$. Then, for every $s \in Q_A$ and $t \in Q_B$ such that $s \; R \; t$, for every $ICTL^*_\Sigma \backslash X$ state formula $f$ and for every valuation $e$ it holds that*

$$A, s, e \models f \text{ iff } B, t, e \models h(f) \qquad (*)$$

*and, for every matching paths $p$ starting at $s$ and $q$ starting at $t$, for every $ICTL^*_\Sigma \backslash X$ path formula $f'$ and for every valuation $e$ it holds that*

$$A, p, e \models f' \text{ iff } B, q, e \models h(f') \qquad (**)$$

*Proof.* Let's prove it by mutual induction on formulas of $ICTL^*_\Sigma \backslash$X.

Let's first show the base case: let $P \in AP$. Then:
$A, s, e \models P_i$
$\iff$ (by definition)
$P \in \lambda_{T_i}(s_{e(i)})$
$\iff$ (condition 2.a of the definition of stuttering bisimulation)
$P \in h_L(\lambda_{T_i}(s_{e(i)}))$
$\iff$ (definition)
$P \in \lambda_{U_{h(i)}}(t_{e(h(i))})$
$\iff$ (by definition)
$B, t, e \models P_{h(e(i))}$
$\iff$ (by definition)
$B, t, e \models h(P_i)$

Now we prove inductively that, by applying any rule that produce a state formula starting from state formulas satisfying (*), the produced state formula still satisfies (*).

The conjunction case and the negative case are trivial.

Suppose $f \equiv \forall i : r : g$, where $r$ is a range expression. Then:

$A, s, e \models \forall i : r : g$
$\iff$ (by definition)
$A, s, e' \models g$ for every $i$-variant $e'$ of $e$ that satisfies r
$\iff$ (inductive hypothesis for each $i$-variant e' of $e$ satisfying r)
$B, t, e' \models h(g)$ for every $i$-variant e' of $e$ that satisfies h(r)
$\iff$ (by definition)
$B, t, e \models \forall i : h(r) : h(g)$
$\iff$ (by definition)
$B, t, e \models h(f)$

The case where $f \equiv \exists i : r : g$ is analogous.

The proofs for path formulas formed as conjunction, negation, and index quantifiers on other path formulas follow directly from the proofs above and from the definition of the semantics for path formulas.

For the case where $f$ is a path formula of the form $X_s^a g$.:
$A, p, e \models X_s^a g$
$\iff$ (by definition)
$\exists i$ s.t., for every $j < i$, $a_j = \tau$, $a_i = a$ and $A, s_{i+1}, e \models g$
$\iff$ (p and q match)
$\exists l$ s.t., for every $k < l$, $b_k = \tau$, $b_l = h^{\Sigma}(a)$ and $A, t_{k+1}, e \models h(g)$
$\iff$ (by definition)
$B, q, e \models X_s^{h^{\Sigma}(a)} h(g)$
$\iff$ (by definition)
$B, q, e \models h(X_s^a g)$

For the case where $f$ is a path formula of the form $g_0 \ U \ g_1$:
$A, p, e \models g_0 \ U \ g_1$
$\iff$ (by definition)
$\exists i$ s.t. $A, s_i, e \models g_1 \wedge (\forall j$ s.t. $j < i, A, s_j, e \models g_0)$
$\iff$ (p and q match)
$\exists k$ s.t. $B, t_k, e \models h(g_1) \wedge (\forall l$ s.t.$l < k, B, t_l, e \models h(g_0))$
$\iff$ (by definition)
$B, q, e \models h(g_0) \ U \ h(g_1)$
$\iff$ (by definition)
$B, q, e \models h(g_0 \ U \ g_1)$

It remains to show that, by constructing a state formula from a path formula that satisfies (**), the state formula obtained statisfies (*), and vice-versa. Let's procede by cases.

A path formula formed from a state formula can be obtained only by the

rule P1), and it is the very same state formula. This case is trivial.

Suppose $f$ is a state formula of the form $Eg$, where $g$ is a path formula.
$A, s, e \models Eg$
$\iff$ (by definition)
$\exists\ p$ fair fullpath starting at $s$ s.t. $A, p, e \models g$
$\iff$ (claim, proved below)
$\exists\ q$ fair fullpath starting at $t$ s.t. $B, q, e \models h(g)$
$\iff$ (by definition)
$B, t, e \models Eh(g)$
$\iff$ (by definition)
$B, t, e \models h(f)$

*Proof of the claim.* Suppose there exists a fullpath $p$ starting at $s$ such that $A, p, e \models g$. The idea is to decompose this fullpath in a sequence of stuttering sequences $\{p_0, p_1, ..., p_n, ...\}$ (note that it can be a finite sequence). We then proceed by recursion: by hypothesis, $p_0 \in Stutter(s, a)$, and by point 2 (b or c) of the definition of stuttering bisimulation, there is a stuttering sequence $q_0 \in Stutter(t, h_\Sigma(a))$ such that $p_0$ and $q_0$ match. The matching of $p_0$ and $q_0$ also implies that their last states, let's call them $s_1$ and $t_1$ respectively, are related by $R$. We can apply again point 2 of the definition of stuttering bisimulation to $p_1$ and so on, obtaining a sequence of stuttering sequences $\{q_0, q_1, ..., q_n, ...\}$, such that each $q_i$ matches with $p_i$. It is straightforward to see that this means that the path $q$, obtained by concatenating the pahts $\{q_0, q_1, ..., q_n, ...\}$, matches with $p$. By inductive hypothesis, we got the double implication.

The case where $f$ has the form $Ag$ follows by the remark that $Ag$ is equivalent to $\neg E \neg g$, so it is sufficient to apply the inductive hypothesis.

□

## 3.4 The Reduction Theorem

The Reduction Theorem is the key result to prove the existence of cut-offs, because it permits to construct a stuttering bisimulation between quotients of TPRs of different sizes. The theorem was proven by Emerson and Namjoshi in [6] using block bisimulation. Here we present a proof that uses stuttering bisimulation instead of block bisimulation.

**Definition 3.4.** *(Intervals on a ring)* $[i:j]_n$ *denotes the set of indices on the clockwise segment of a ring of size* $n$ *starting from* $i$ *and ending in* $j$*.* $[i:j)_n = [i:j]_n \backslash \{j\}$, $(i:j]_n = [i:j]_n \backslash \{i\}$ *and* $(i:j)_n = [i:j]_n \backslash \{i,j\}$*. If* $i = j$*, by convention* $[i,i] = [n]$ *and* $[i,i) = (i,i] = (i,i) = [n] \backslash \{i\}$*.*

**Theorem 3.2** (Reduction)**.** *For* $I \subseteq [n]$ *and* $J \subseteq [k]$ *sets of indices, let* $h : I \to J$ *be a bijection s.t.* $\forall i, j \in I$*:*

1. $i \leqslant j$ *iff* $h(i) \leqslant h(j)$ *(where* $\leqslant$ *is the usual relation on* $\mathbb{N}$*)* [2]
2. $(i:j)_n \neq \varnothing$ *iff* $(h(i):h(j))_k \neq \varnothing$

*Then* $M_n|_I$ *and* $M_k|_J$ *are stuttering equivalent.*

*Proof.* For brevity, let's call $A := M_n|_I$ and $B := M_k|_J$. In order to prove the theorem, we define a relation $R$ on $Q_A \bigsqcup Q_B$ and two bijective functions $h_\Sigma$ and $h_L$, and prove it is a stuttering bisimulation.

First note that all the actions in $A$ have index in $I$ and all the actions in $B$ have index in $J$, hence we have an inducted function $h_\Sigma : \Sigma_A \to \Sigma_B$ that is well-defined. The same argument holds for the labels, hence we have an inducted function $h_L : L_A \to L_B$ that is well-defined.

Let $R$ be defined as follows: for every $s \in Q_A$ and for every $t \in Q_B$, $s \; R \; t$ iff:

1. The local state of the initial token distribution processes $D_n$ and $D_k$ are the same, that is: either both are in the *end* state, or neither of them is in the *end* state.
2. The state of the processes indexed by $I$ is identical to the state of the processes indexed by $J$, up to $h$. That is $h_L(\lambda_A(s)) = \lambda_B(t)$.
3. The token in $s$ is at the process $l \in I$ iff the token in $t$ is at the process $h(l)$.
4. The token in $s$ is between process $l$ and $m$ iff the token in $t$ is between $h(l)$ and $h(m)$.

[2] Actually, given any order on $I$ consistent with the direction of the ring $Ring_n$ (that is, choosing an arbitrary minimum point and then ordering by following the clockwise verse), it is sufficient to require that the order on $J$ induced by $h$ is consistent with the clockwise verse of the ring $Ring_k$. The point 1. above is the same as this condition after having chosen 0 as arbitrary minimum point, but there is no actual need to do that, except for simplicity's sake. In practice, it is irrelevant which minimum point is chosen, and the proof holds as well with the condition as written in this note.

Now let's prove that $R$ is a stuttering bisimulation. Suppose that $s\ R\ t$, where $s$ is a state in $M_n|_I$ and $t$ is a state in $M_k|_J$.

It is trivial to see that any initial state of $A$ is related to any initial state of $B$ by $R$, hence point 1. of the definition of stuttering bisimilation is satisfied.

By definition of $R$ (point 2), we have that $h_L(\lambda_A(s)) = \lambda_B(t)$, so point 2.a of the definition of stuttering bisimulation is satisfied.

Now note that there can't be infinite stuttering sequences due to fairness conditions: the token has to be passed eventually, which means that, since the ring is finite, every process will receive the token eventually (in particular some process indexed by $I$ will receive it), so silent transitions cannot succeed forever. This means that $Stutter(s, Inf) = \varnothing$, hence point 2.c is checked.

It remains just to show point 2.b: for every path $p \in Stutter(s, a)$, we need to find a path $q \in Stutter(t, h_\Sigma(a))$ s.t. $p$ and $q$ match by $R$. So let's consider a path $p$. The last transition of $p$ has to be a non silent transition, i.e. it has to involve a process $i \in I$, i.e. it could be either an internal transition of process $i$, or the process $i$ sending the token, or the process $i$ receiving the token (either from the initial distributor process $D_n$ or from another process).

Before proceeding case by case, let's note that every state on the path $p$ except the final one is related to $t$ by $R$: in fact, the four conditions of the definition of $R$ hold up to silent transitions. This means that we are on the right path to construct two partitions of $p$ and $q$: we have a segment $p^{(1)} = (s, s_1, s_2, ..., s_{n-1})$ and a segment $q^{(1)} = (t)$ s.t. every state of $p^{(1)}$ is related to every state of $q^{(1)}$. It remains to find an another segment $q^{(2)}$ s.t. every state in it matches with $p^{(2)} = (s_n)$. Let's consider the different cases:

- Internal move: $a = a_i$. Since the state of the processes indexed by $i$ and $h(i)$ are identical (by point 2 of the def. of $R$), it is possible to apply the transition $a_{h(i)}$ to $t$. The state obtained is clearly related to $s_n$ by $R$.

- Sending the token: $a = (snd_i, rcv_{i+1})$. Since the state of the processes indexed by $i$ and $h(i)$ (and by $i + 1$ and $h(i) + 1$) are identical, it is possible to apply the action $(snd_{h(i)}, rcv_{h(i)+1})$ in $t$, and the state obtained is related to $s_n$ by $R$.

- Receiving the token from another process: $a = (snd_{i-1}, rcv_i)$ (it is necessary to analyze this case only if $i - 1 \notin I$, otherwise it has been already considered in the previous case). Since the action is not silent, the token has to be in $(k : i)$ for some $k \in I$. Since $s\ R\ t$, by point 4 of the definition, the other token is at $j \in (h(k) : h(i))$. If $j = h(i) - 1$ it means that the process $h(i) - 1$ can send the token and we are done, since it is possible to apply $(snd_{h(i)-1}, rcv_{h(i)})$.
  Otherwise, by fairness conditions, the token has to be passed, step by step, from $j$ to $h(i) - 1$ (this is possible since $k < i$ implies $h(k) < h(i)$). Doing so involves only silent transitions, thus we can put these states in the segment $q^{(1)}$, since they are all related to $s$ (and to $s_1$,...,$s_{n-1}$) by $R$. After these transitions, it is possible to apply $(snd_{h(i)-1}, rcv_{h(i)})$.

- Receiving the token from the initial distribution process: $a = (init\_snd_i, rcv_i)$. If $i \in I$, let the first transition of $t$ be the token assignment to $h(i)$, so a matching path is found. If $i \notin I$, there exists a pair $k, l \in I$ s.t. $i \in (k : l)$ and $(k : l)$ does not contain any index of $I$. By property of $h$, $(h(k), h(l))$ is not empty, let's say it contains $z$, and let the first transition of $t$ be the token assignment to $z$: a matching path is found.

□

# 4 Decidability results for token-passing rings

Working with rings composed by a variable number of processes, it can be very useful to verify properties that hold independently from the number of processes. In order to study this question in a formal framework, we need to introduce the notion of parameterized model-checking problem.

## 4.1 Parameterized model-checking problem

Let's consider a sequence of LTSs $\mathbf{A} = (A_1, A_2, ...)$, where each $A_n$ is a composite LTS of the form $A_n = B_{n_1} || ... ||B_{n_{k_n}}$ built from $d$ process templates, i.e. there exist $d$ LTS $P_1, \ ... \ , P_d$ such that for every $n \in \mathbb{N}$ and for every $j \in [k_n]$, there exists a $l \in [d]$ such that $B_{n_j}$ is isomorphic $P_l$.

From this sequence $\mathbf{A}$, we can construct a sequence $\mathbf{G} = (G_1, G_2, ...)$ of $d$-colored graph, where, for every $n \in \mathbb{N}$, $G_n$ is the graph associated to $A_n$, similarly to what done in section 2.4: $G_n = (V_n, \mathcal{E}_n, \texttt{type}_n)$ is defined as:

- $V_n = \{1, ..., k_n\}$
- $\texttt{type}_n(i) = l$ iff $B_{n_i}$ is isomorphic to $P_l$.
- $(i, j) \in \mathcal{E}_n$ iff $\Sigma_{A_n}$ contains a synchronized action between $B_{n_i}$ and $B_{n_j}$.

Given a finite set of LTSs $\mathcal{P} = \{P_1, \ ... \ , P_d\}$ (seen as process templates), a sequence of LTSs $\mathbf{A}$ defined as above, a fragment $\mathcal{F}$ of the indexed temporal logic $ICTL^*\backslash X$ over a set of atomic propositions $AP$, we can have the following definitions.

**Definition 4.1** (Parameterized model-checking problem)**.** *The parameterized model-checking problem for $(\mathcal{P}, \boldsymbol{A}, \mathcal{F})$, abbreviated as $PMCP(\mathcal{P}, \boldsymbol{A}, \mathcal{F})$ is defined as:*

- *Input. A formula $f \in \mathcal{F}$.*
- *Output. "Yes" if $A_n \models f$ for every $n \in \mathbb{N}$, "No" otherwise.*

It is now possible to define what is a cutoff for a parameterized model-checking problem.

**Definition 4.2** (Cutoff)**.** *We say that $m \in \mathbb{N}$ is a cutoff for $(\mathcal{P}, \boldsymbol{A}, \mathcal{F})$ if:*

$$\forall f \in \mathcal{F} : \ (\forall n \in \mathbb{N} \ A_n \models f \ \ \textit{iff} \ \ \forall n < m \ A_n \models f)$$

When $\mathcal{P}$ and $\mathbf{A}$ are clear, it can be said that $m$ is a cutoff for $\mathcal{F}$.
Showing a cutoff for a parameterized model-checking problem is a very useful technique for proving the decidability of a PMCP. In fact we have the following result:

**Proposition 3.** *If* $(\mathcal{P}, \boldsymbol{A}, \mathcal{F})$ *has a cutoff, then* $PMCP(\mathcal{P}, \boldsymbol{A}, \mathcal{F})$ *is decidable.*

*Proof.* Let $m$ be a cutoff for $(\mathcal{P}, \mathbf{A}, \mathcal{F})$, i.e. $\forall f \in \mathcal{F}: (\forall n \in \mathbb{N}\ A_n \models f \text{ iff } \forall n < m\ A_n \models f)$. We can construct the following algorithm: given $f \in \mathcal{F}$, output "Yes" if for every $n < m$ it holds that $A_n \models f$, output "No" otherwise. This is a solving algorithm for $PMCP(\mathcal{P}, \mathbf{A}, \mathcal{F})$. □

## 4.2 Cutoff theorems

For particular fragments of ICTL*\X there are important cut-offs results on TPRs. Some of the most useful state that:

- The fragment $\{(\forall i : g(i)) \mid g(i)$ does not contain index quantifiers$\}$ has a cutoff of 2
- The fragment $\{(\forall i : g(i, i{+}1)) \mid g(i, i{+}1)$ does not contain index quantifiers$\}$ has a cutoff of 3
- The fragment $\{(\forall i, j : j \neq i : g(i, j)) \mid g(i, j)$ does not contain index quantifiers$\}$ has a cutoff of 4
- The fragment $\{(\forall i, j : j \neq i : g(i, i{+}1, j)) \mid g(i, i{+}1, j)$ does not contain index quantifiers$\}$ has a cutoff of 5

Let's prove them in order.

**Lemma 4.1.** *Fixed an initial state $s^0$, then $M_n, s^0 \models (\forall i : g(i))$ iff $M_n, s^0 \models g(0)$, where $g(i)$ is a $ICTL^*_\Sigma\backslash X$ formula not containing index quantifiers.*

*Proof.* We have the following chain:
$M_n, s^0 \models g(0)$
$\iff$ (definition)
$M_n, s^0, e \models g(i)$ holds for every valuation $e$ that maps $i$ to 0
$\iff$ (since g(i) has no index quantifiers, it does not distinguish between the process indexed by 0 and the processes indexed by other indices. Thanks to the symmetry of the system, we can extend the proposition to every valuation)
$M_n, s^0, e \models g(i)$ holds for every valuation $e$
$\iff$ (definition)
$M_n, s^0, e \models \forall i.g(i)$ holds for every valuation $e$
$\iff$ (definition)
$M_n, s^0 \models \forall i.g(i)$

□

**Theorem 4.2.** *For every $ICTL^*_\Sigma\backslash X$ formula $f \equiv (\forall i : g(i))$ where $g(i)$ does not contain index quantifiers, and for every $n \in \mathbb{N}$ s.t. $n \geqslant 2$, then $M_n, s^0_n \models f$ iff $M_2, s^0_2 \models f$.*

*Proof.* We have the following chain:

$$\begin{aligned} & M_n, s^0_n \models f \\ \overset{(1)}{\iff} \; & M_n, s^0_n \models g(0) \\ \overset{(2)}{\iff} \; & M_n|_0, s^0_n \models g(0) \\ \overset{(3)}{\iff} \; & M_2|_0, s^0_2 \models g(0) \end{aligned}$$

$\overset{(4)}{\iff} M_2, s_2^0 \models g(0)$
$\overset{(5)}{\iff} M_2, s_2^0 \models f$

(1) The initial state is symmetric due to the very definition of the system, so it is possible to apply the previous lemma.

(2) $g(0)$ refers only to propositions indexed by 0.

(3) It is possible to apply the Reduction Theorem with $h : 0 \mapsto 0$, noting that it satisfies the conditions required in the hypothesis. It follows that $M_n|_0$ and $M_2|_0$ are stuttering equivalent. In particular, the stuttering bisimulation $R$ constructed in the proof is such that $s_n^0$ is related to $s_2^0$ by $R$. Thus, it possible to apply Theorem 3.1. Noting that $h(g(0) = g(0)$, we obtain the double implication.

(4) $g(0)$ refers only to propositions indexed by 0.

(5) By previous lemma.

□

**Lemma 4.3.** *Fixed an initial state $s^0$, then $M_n, s^0 \models (\forall i : g(i, i+1))$ iff $M_n, s^0 \models g(0, 1)$, where $g(i, i+1)$ is a $ICTL^*_\Sigma\backslash X$ formula not containing index quantifiers.*

*Proof.* The proof is very similar to the one of the previous lemma. We have the following chain:
$M_n, s^0 \models g(0, 1)$
$\iff$ (definition)
$M_n, s^0, e \models g(i, i+1)$ holds for every valuation $e$ that maps $i$ to 0
$\iff$ (symmetry)
$M_n, s^0, e \models g(i, i+1)$ holds for every valuation $e$
$\iff$ (definition)
$M_n, s^0, e \models \forall i.g(i, i+1)$ holds for every valuation $e$
$\iff$ (definition)
$M_n, s^0 \models \forall i.g(i, i+1)$

□

**Theorem 4.4.** *For every $ICTL^*_\Sigma\backslash X$ formula $f$ of the form $(\forall i : g(i, i+1))$ where $g$ does not contain index quantifiers, and for every $n \in \mathbb{N}$ s.t. $n \geqslant 3$, then $M_n, s_n^0 \models f$ iff $M_3, s_3^0 \models f$.*

*Proof.* We have the following chain:

$M_n, s_n^0 \models g(i, i+1)$
$\overset{(1)}{\iff} M_n, s_n^0 \models g(0, 1)$
$\overset{(2)}{\iff} M_n|_{\{0,1\}}, s_n^0 \models g(0, 1)$

$\overset{(3)}{\iff} M_3|_{\{0,1\}}, s_3^0 \models g(0,1)$
$\overset{(4)}{\iff} M_3, s_3^0 \models g(0,1)$
$\overset{(5)}{\iff} M_3, s_3^0 \models g(i, i+1)$

(1) The initial state is symmetric due to the very definition of the system, so it is possible to apply the previous lemma.

(2) $g(0,1)$ refers only to propositions indexed by 0 and by 1.

(3) It is possible to apply the Reduction Theorem with $h : \{0,1\} \to \{0,1\}$ such that $h(0) = 0$ and $h(1) = 1$, noting that it satisfies the conditions required in the hypothesis. It follows that $M_n|_{\{0,1\}}$ and $M_3|_{\{0,1\}}$ are stuttering equivalent. In particular, the stuttering bisimulation $R$ constructed in the proof is such that $s_n^0$ is related to $s_3^0$ by $R$. Thus, it possible to apply Theorem 3.1. Noting that $h(g(0,1) = g(0,1)$, we obtain the double implication.

(4) $g(0,1)$ refers only to propositions indexed by 0 and by 1.

(5) By previous lemma.

□

**Lemma 4.5.** *Fixed an initial state $s^0$, then $M_n, s^0 \models (\forall i, j : i \neq j : g(i,j))$ iff $M_n, s^0 \models \forall j : g(0,j)$, where $g(i,j)$ is a $ICTL^*_\Sigma \backslash X$ formula not containing index quantifiers.*

*Proof.* We have the following chain:
$M_n, s^0 \models \forall j : g(0,j)$
$\iff$ (definition)
$M_n, s^0, e \models \forall j : g(i,j)$ holds for every valuation $e$ that maps $i$ to 0
$\iff$ (symmetry)
$M_n, s^0, e \models \forall j : g(i,j)$ holds for every valuation $e$
$\iff$ (definition)
$M_n, s^0, e \models \forall i, j : i \neq j : g(i,j)$ holds for every valuation $e$
$\iff$ (definition)
$M_n, s^0 \models \forall i, j : i \neq j : g(i,j)$

□

**Lemma 4.6.** *For $n \geqslant 4$:*

*a)* $M_n|_{(0,1)} \sim M_4|_{(0,1)}$

*b)* $M_n|_{(0,n-1)} \sim M_4|_{(0,3)}$

*c)* $M_n|_{(0,j)} \sim M_4|_{(0,2)}$ *for* $j \notin \{0, 1, n-1\}$

*Proof.* This is a straightforward application of the Reduction Theorem.

In fact, let $h : [n] \to [4]$ be defined by: $h(j) = \begin{cases} 0, & \text{if } j = 0 \\ 1, & \text{if } j = 1 \\ 3, & \text{if } j = n-1 \\ 2, & \text{otherwise} \end{cases}$.

To prove point a), it suffices to apply the Reduction Theorem with $h$ restricted to $\{0, 1\}$, whereas for b) it can be applied with $h$ restricted to $\{0, n-1\}$. In order to prove c), for every $j \notin \{0, 1, n-1\}$, the stuttering equivalence follows by the Reduction Theorem with $h$ restricted to $\{0, j\}$ each time. □

**Theorem 4.7.** *For every $ICTL^*_\Sigma\backslash X$ formula $f$ of the form $(\forall i, j : i \neq j : g(i, j))$ where $g$ does not contain index quantifiers, and for every $n \in \mathbb{N}$ s.t. $n \geqslant 4$, then $M_n, s^0_n \models f$ iff $M_4, s^0_4 \models f$.*

*Proof.* We can proceed in a way similar to the previous theorems, but we need to do an observation first. By a reasoning analogous to the one of Lemma 4.1, we can instantiate the first index quantifier $i$ to 0, for a symmetry argument. But it is not possible do to that for the second index quantifier $j$ too. In fact, fixing an evaluation $e$, the positions that $e(0)$ and $e(j)$ assume can be of three different types:

- $e(0)$ is the successor of $e(j)$
- $e(0)$ is the predecessor of $e(j)$
- $e(0)$ and $e(j)$ are not adjacent

This three cases are significantly different one from the other. In fact, in the first case there is an action sending the token from $e(0)$ to $e(j)$, in the second an action sending the token from $e(j)$ to $e(0)$, and in the third no token exchange between the two processes. This suggests that a system composed by 3 processes will not be able to cover all the formulas of the form in the hypothesis, while a system composed by 4 processes will do it: the processes indexed by $0, 1$ and $n-1$ do not change, while all the processes indexed by $2, ..., n-2$ are *simulated* by a single processes, indexed by 2.

Similarly to what done before, we have the following chain:
$M_n, s^0 \models (\forall i, j : i \neq j : g(i,j))$
$\iff$ (Lemma 4.5)
$M_n, s^0 \models (\forall j : j \neq 0 : g(0,j))$
$\iff$ (By the definition, we can *split* the cases. Note that doing so is possible only due to the observation preceding the chain)
$M_n, s^0 \models (g(0,1)) \ \wedge$
$M_n, s^0 \models (g(0,n-1)) \ \wedge$
$M_n, s^0 \models (\forall j : j \notin \{0,1,n-1\} : g(0,j))$
$\iff$ (Lemma 4.6)
$M_4, s^0 \models (g(0,1)) \ \wedge$
$M_4, s^0 \models (g(0,3)) \ \wedge$
$M_4, s^0 \models (g(0,2))$
$\iff$ (By the definition, we can *merge* the cases. Note that doing so is possible only due to the observation preceding the chain)
$M_4, s^0 \models (\forall j : j \neq 0 : g(0,j))$
$\iff$ (Lemma 4.5)
$M_4, s^0 \models (\forall i, j : i \neq j : g(i,j))$

□

**Lemma 4.8.** *Fixed an initial state $s^0$, then*

$$M_n, s^0 \models (\forall i, j : neq(i, i+1, j) : g(i, i+1, j))$$

*iff*

$$M_n, s^0 \models (\forall j : neq(0, 1, j) : g(0, 1, j))$$

*where $g(i, i+1, j)$ is a $ICTL^*_\Sigma \backslash X$ formula not containing index quantifiers.*

*Proof.* We have the following chain:
$M_n, s^0 \models (\forall j : neq(0,1,j) : g(0,1,j))$
$\iff$ (definition)
$M_n, s^0, e \models (\forall j : neq(0,1,j) : g(0,1,j))$ holds for every valuation $e$ that maps $i$ to 0 and $e(j) \neq 0$ and $e(j) \neq 1$
$\iff$ (symmetry)
$M_n, s^0, e \models (\forall j : neq(i,i+1,j) : g(i,i+1,j))$ holds for every valuation $e$ such that $e(i) \neq e(j)$ and $e(i+1) \neq e(j)$
$\iff$ (definition)
$M_n, s^0, e \models (\forall i, j : neq(i,i+1,j) : g(i,i+1,j))$ holds for every valuation $e$ such that $e(i) \neq e(j)$ and $e(i+1) \neq e(j)$
$\iff$ (definition)
$M_n, s^0 \models (\forall i, j : neq(i,i+1,j) : g(i,i+1,j))$

□

**Lemma 4.9.** *For $n \geqslant 5$:*

*a)* $M_n|_{(0,1,2)} \sim M_5|_{(0,1,2)}$

*b)* $M_n|_{(0,1,n-1)} \sim M_5|_{(0,1,n-1)}$

*c)* $M_n|_{(0,1,j)} \sim M_5|_{(0,1,3)}$ *for* $j \notin \{0, 1, 2, n-1\}$

*Proof.* The proof is analogous to the one of Lemma 4.6, i.e. a straightforward application of the Reduction Theorem. Let $h : [n] \to [5]$ be defined by:

$$h(j) = \begin{cases} 0, & \text{if } j = 0 \\ 1, & \text{if } j = 1 \\ 2, & \text{if } j = 2 \\ 4, & \text{if } j = n-1 \\ 3, & \text{otherwise} \end{cases}.$$

Each point is proved by applying the Reduction Theorem with $h$ restricted to the triple involved. □

**Theorem 4.10.** *For every $ICTL^*_\Sigma\backslash X$ formula $f$ of the form $(\forall i, j : neq(i, i+1, j) : g(i, i+1, j))$ where $g$ does not contain index quantifiers, and for every $n \in \mathbb{N}$ s.t. $n \geqslant 5$, then $M_n, s^0_n \models f$ iff $M_5, s^0_5 \models f$.*

*Proof.* The idea is very similar to the one of Theorem 4.7. First, we instantiate the first index quantifier $i$ to 0 (and $i+1$ to 1, accordingly). Second, fixing an evaluation $e$, the positions that $e(j)$ can assume are of three different types:

- $e(j)$ is the successor of $e(1)$
- $e(j)$ is the predecessor of $e(0)$
- $e(j)$ is not adjacent with neither $e(1)$ nor $e(0)$

Again, we can construct from $M_n$ a system composed by 5 processes, where the processes indexed by $0, 1, 2$ and $n-1$ do not change, while all the processes indexed by $3, ..., n-2$ are *simulated* by a single processes, indexed by 3.

So we have the following chain:

$M_n, s^0 \models (\forall i, j : neq(i, i+1, j) : g(i, i+1, j))$
$\iff$ (Lemma 4.8)
$M_n, s^0 \models (\forall j : neq(0, 1, j) : g(0, 1, j))$
$\iff$ (By the definition, we can *split* the cases. Note that doing so is possible only due to the observation preceding the chain)
$M_n, s^0 \models (g(0, 1, 2)) \ \wedge$
$M_n, s^0 \models (g(0, 1, n-1)) \ \wedge$

$M_n, s^0 \models (\forall j : j \notin \{0, 1, 2, n-1\} : g(0, 1, j))$
$\iff$ (Lemma 4.9)
$M_5, s^0 \models (g(0, 1, 2)) \ \wedge$
$M_5, s^0 \models (g(0, 1, 4)) \ \wedge$
$M_5, s^0 \models (g(0, 1, 3))$
$\iff$ (By the definition, we can *merge* the cases. Note that doing so is possible only due to the observation preceding the chain)
$M_5, s^0 \models (\forall j : neq(0, 1, j) : g(0, 1, j))$
$\iff$ (Lemma 4.5)
$M_5, s^0 \models (\forall i, j : neq(i, i+1, j) : g(i, i+1, j))$

□

## 4.3 Overview over more general (un)decidability results

In 2014, Aminof et al.[1] proved a decidability result for token passing rings for CTL*\X formulas with only universal or only existential quantifiers. For a formula indexed over $k$ index variables, the cut-off is $2k$.

In 2004, Clarke et al.[4] proved a decidability result for token passing rings for formulas in LTL\X, using the decomposition technique.

It is possible to permit to the token to take multiple values, but this quickly leads to undecidability. Indeed, even the safety fragment of LTL\X is undecidable, as proven by Emerson and Namjoshi in 2003[6].

It is possible to extend the notion of token passing ring to a generic graph. Doing so, it arises the possibility for a process to send (receive) the token to (from) two or more other processes. When this possibility is denied, i.e. every process can send (receive) only to (from) a process, we talk about *direction-unaware TPS*, otherwise about *direction-aware TPS*.

In 2014, Aminof et al. proved an undecidability result for direction-aware token passing systems for formulas in LTL\X.

A complete overview over decidability and undecidability results can be found in "*Decidability of Parameterized Verification*"[2].

## 4.4 Applications

Token-passing rings have proved to be a very powerful way to handle the communication in computer networks. In fact, there is a wide variety of applications that find the token-passing ring structure to be very suitable.

For examples, token-passing rings are used by several protocols for mutual exclusion, which strongly benefit from the ring structure in terms of safety. Among these protocols, one of the most well-known is Milner's scheduler protocol [10].

**Milner's scheduler protocol** consists of a certain number of processes arranged on a ring, each of which has to accomplish a task. In order to activate the task, a process needs to have the token. After the activation, it sends the token clockwise to another process, and then starts working on the task. It can receive again the token either before or after completing the task. After having completed a task, it can activate a new task, provided it holds the token.

It is clear that Milner's scheduler protocol can be seen as a composed LTS of the form $M_n$ constructed as in section 3.1, where the base template process $P$ is defined as follows. The process can be in an initial state $in$, a "task activation" state $a$, a "working on the task" state $w$, or a "completing the task" state $c$, and in each state - except for the task activation state - it can have the token (noted with the subscript $T$) or not (noted with the subscript $N$). Formally, we define it as:

- $Q = \{in_N,\ in_T,\ a_T,\ c_N,\ c_T,\ w_N,\ w_T\}$
- $I = \{in_N\}$
- $\Sigma = \{rcv,\ snd,\ act,\ compl,\ \tau\}$
- $\delta$ is formed by the following transitions:

  - rcv: $in_N \xrightarrow{rcv} in_T, \quad c_N \xrightarrow{rcv} in_T, \quad w_N \xrightarrow{rcv} w_T$
  - snd: $a_T \xrightarrow{snd} w_N$
  - act: $in_T \xrightarrow{act} a_T$
  - compl: $w_N \xrightarrow{compl} c_N \quad w_T \xrightarrow{compl} c_T$
  - $\tau$: $c_T \xrightarrow{\tau} in_T$ (besides the transitions of the form $q \xrightarrow{\tau} q$ for every $q \in Q$)

- $L$ and $\lambda$ are constructed in the usual way for Kripke structures

Being the number of processes variable, it is clear that, if we are interested in proving general properties of Milner's scheduler protocol, we are facing a parameterized model checking problem.
Let's say, for example, that we want to check that every process, in order to

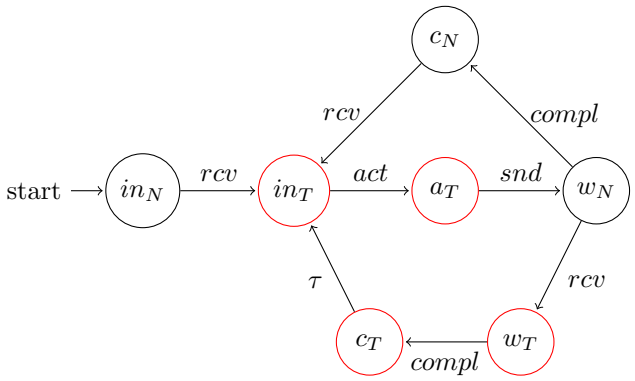


Figure 1: Milner's scheduler protocol. In red the states having the token.

activate a task, must have completed the task previously. This property can be expressed by the $ICTL^*_\Sigma\backslash X$ formula

$$\forall i \ : \ AG(a_i \rightarrow (a_i \ U_w \ \neg a_i \ U_w \ c_i))$$

In order to verify this specification in a system composed by an arbitrary number of processes, thanks to Theorem 4.2, it suffices to verify it in a system of size 2.

Another example of application is the problem of **Leader election in a synchronous ring** [9]. The problem consists, having a certain number of processes disposed on a ring, in deciding which of them has to be the "leader", i.e. which of them can initiate the communication.

In order to prevent two processes from communicating simultaneously, risking to interfere with each other and losing information, it is very important to have a clear procedure to decide which of them has the right to start the communication. Moreover, in a token-passing system, it can happen that the token gets lost, and the process to regenerate it is equivalent to electing a leader.

One idea to construct a system for the leader election is as follows. We suppose that every process has an unique identifier (UID), represented as a natural number, and it sends it around the ring. Then, when it receives an incoming UID, it checks whether it is greater than its own (in this case it keeps passing it), less than its own (in this case it passes its UID), or equal (in this case it proclames itself to be the leader). This algorithm can be viewed as an LTS, and the system as a token-passing ring

# Part II
# Token-Passing Rings as array-based systems

## 5 System description

### 5.1 Preliminaries

We work in first-order logic, and assume the usual notions of signature (requiring it includes the equality), terms, formulas, ground formulas, quantifier free formulas, interpretations, etc.

For $\Sigma$ a signature and $\underline{x}$ a finite set of variables, with $t(\underline{x})$ we indicate a term in which at most the variables in $\underline{x}$ are free, and with $\phi(\underline{x})$ we indicate a formula in which at most the variables in $\underline{x}$ are free.

By saying that a formula $\phi(\underline{x})$ is satisfiable we mean that its existential closure is satisfiable, and by saying that it is valid we mean that its universal closure is valid.

**Definition 5.1** (Substructure)**.** *Given a $\Sigma$-structure $\mathcal{M} = (M, \mathcal{I})$, we say that $\mathcal{M}' = (M', \mathcal{I}')$ is a $\Sigma$-substructure of $\mathcal{M}$ if $M' \subseteq M$ and $\mathcal{I}'$ is obtained from $\mathcal{I}$ by restriction.*

**Definition 5.2** (Substructure generated by a subset)**.** *The $\Sigma$-structure generated by a subset $X$ of $M$ is the smallest $\Sigma$-substructure of $\mathcal{M}$ whose domain contains $X$. In case the $\Sigma$-structure generated by $X$ is the whole $\mathcal{M}$, it is said that $X$ generates $\mathcal{M}$.*

**Definition 5.3** (Theory)**.** *Given a signature $\Sigma$ and a class of $\Sigma$-structures $\mathcal{C}$, we say that the pair $(\Sigma, \mathcal{C})$ is a theory, and the structures in $\mathcal{C}$ are said models of the theory* [3]*.*

**Definition 5.4** (Closure under substructures)**.** *A class of structures $\mathcal{C}$ is closed under substructures if, for every $\mathcal{M} \in C$, every $\mathcal{N}$ isomorphic to a substructure of $\mathcal{M}$ is in $\mathcal{C}$.*

**Definition 5.5** (T-satisfiability)**.** *Given a theory $T$, a formula $\phi$ is said $T - satisfiable$ iff there exists a model $M$ of $T$ and an assignment $a$ to the free variables of $\phi$ making $\phi$ true in $M$ under $a$. It is said $T - valid$ it is valid in all the models of $T$.*

**Definition 5.6** (T-equivalence)**.** *Given a theory $T$, we say that two formulas $\phi$ and $\psi$ are $T$-equivalent if $T \models \forall \underline{x}(\phi(\underline{x}) \leftrightarrow \psi(\underline{x}))$*

[3] Note that this is the definition in the SMT-formalism, while the usual theory definition is quite different.

**Definition 5.7** (Quantifier elimination)**.** *A theory admits quantifier elimination iff for every formula $\phi(\underline{x})$ there exists a quantifier-free formula $\phi'(\underline{x})$ which is $T$-equivalent to $\phi(\underline{x})$.*

In an applications-oriented context like ours, we require $\phi'(\underline{x})$ to be computable.

**Definition 5.8** (Locally finiteness)**.** *A theory $T = (\Sigma, \mathcal{C})$ is locally finite iff $\Sigma$ is finite and for every set of variables $\underline{x}$ there are finitely many $\Sigma(\underline{x})$-terms $t_1, ..., t_k$ s. t. for every other term $u$, there exists $i \in [k]$ such that $T \models u = t_i$. The terms $t_1, ..., t_k$ are called $\Sigma(\underline{x})$-representative terms.*

**Definition 5.9** (Effectively locally finiteness)**.** *A theory $T = (\Sigma, \mathcal{C})$ is effectively locally finite iff it is locally finite and its representative terms are effectively computable from $(\underline{x})$*

**Definition 5.10** (T-partition)**.** *Given a theory $T$, a $T$-partition is a finite set of quantifier-free formulas $C_1(\underline{x}), \cdots, C_n(\underline{x})$ such that $T \models \forall \underline{x} \bigvee_{i=1}^{n} C_i(\underline{x})$ and $T \models \bigwedge_{i \neq j} \forall \underline{x} \neq (C_i(\underline{x}) \wedge C_j(\underline{x}))$.*

**Definition 5.11** (Case-definable extension)**.** *Given a theory $T = (\Sigma, \mathcal{C})$, a case-definable extension $T' = (\Sigma', \mathcal{C}')$ is an extention obtained form $T$ by applying finitely many times the following procedure:*

*(i) Take a $T$-partition $C_1(\underline{x}), \cdots, C_n(\underline{x})$ and the $\Sigma$-terms $t_1(\underline{x}), \cdots, t_n(\underline{x})$.*

*(ii) Define $\Sigma' = \Sigma \cup \{F\}$ where $F \notin \Sigma$ and the arity of $F$ is equal to the length of $\underline{x}$*

*(iii) Let $\mathcal{C}'$ be the class of $\Sigma'$-structures $\mathcal{M}$ such that its $\Sigma$-reduct is a model of $T$ and such that $\mathcal{M} \models \bigwedge_{i=1}^{n} \forall \underline{x}(C_i(\underline{x}) \rightarrow F(\underline{x}) = t_i(\underline{x}))$*

*These new function symbols are called case-defined functions.*

**Definition 5.12** (Enumerate data-type theory)**.** *An enumerated data-type theory $T$ is a theory whose class of models contains exactly one finite $\Sigma$-structure $\mathcal{M} = (M, \mathcal{I})$ such that for every $m \in M$ there exists a constant $c \in \Sigma$ such that $c^{\mathcal{I}} = m$.*

## 5.2 Array-based systems

**Definition 5.13.** *An index theory $T_I = (\Sigma_I, \mathcal{C}_I)$ is a mono-sorted theory (let's call its sort* `INDEX`*) which is closed under substructures and whose fragment formed by only quantifier free formulas is decidable for $T_I$-satisfiability.*

**Definition 5.14.** *An element theory $T_E$ is a multi-sorted theory which admits quantifier elimination and whose fragment formed by only quantifier free formulas is decidable for $T_E$-satisfiability.*

Given an index theory $T_I(\Sigma_I, \mathcal{C}_I)$ and an element theory $T_E$, we can construct a third theory $A_I^E = (\Sigma, \mathcal{C})$ having three sort symbols: `INDEX`, `ELEM`, and `ARRAY`. $\Sigma$ contains all the symbols in $\Sigma_I$ and all the symbols in $\Sigma_E$, plus a binary function symbol *apply* having sort $\texttt{ARRAY}, \texttt{INDEX} \rightarrow \texttt{ELEM}$ (instead of $apply(a, i)$ we will use the more array-style like notation $a[i]$). The class $\mathcal{C}$ contains exactly the three-sorted structures $\mathcal{M} = (\texttt{INDEX}^{\mathcal{M}}, \texttt{ELEM}^{\mathcal{M}}, \texttt{ARRAY}^{\mathcal{M}}, \mathcal{I})$ such that $\texttt{ARRAY}^{\mathcal{M}}$ is the set of functions from $\texttt{INDEX}^{\mathcal{M}}$ to $\texttt{ELEM}^{\mathcal{M}}$, the function symbol *apply* is interpreted as a function application, and $(\texttt{INDEX}^{\mathcal{M}}, \mathcal{I}_{|\Sigma_I})$ and $(\texttt{ELEM}^{\mathcal{M}}, \mathcal{I}_{|\Sigma_E})$ are models of $T_I$ and $T_E$, respectively. By $\mathcal{M}_I$ (resp. $\mathcal{M}_E$) we will indicate $(\texttt{INDEX}^{\mathcal{M}}, \mathcal{I}_{|\Sigma_I})$ (resp. $(\texttt{ELEM}^{\mathcal{M}}, \mathcal{I}_{|\Sigma_E})$ )

**Definition 5.15** (Array-based system)**.** *An array-based system for an index theory $T_I$ and an element theory $T_E$ is a triple $\mathcal{S} = (\underline{a}, I, \tau)$, where:*

- *$\underline{a}$ is a tuple of variables of sort* `ARRAY` *(these variables are called state variables)*
- *$I(\underline{a})$ is a $\sigma(\underline{a})$-formula (called the initial state formula)*
- *$\tau(\underline{a}, \underline{a}')$ is a $\Sigma(\underline{a}, \underline{a}')$-formula, where $\underline{a}'$ is a renamed copy of the tuple $\underline{a}$ (this formula is called the transition formula)*

## 5.3 Backward reachability

In this section we assume that $I(a)$ is a formula of the form $\forall \underline{i}.\phi(\underline{i}, a[\underline{i}])$ (shorthand: $I(a)$ is a $\forall^I$-formula) and that $\tau(a, a')$ is in *functional form*, i.e. it is a disjunction of formulas of the form $\exists \underline{i}(\phi_L(\underline{i}, a[\underline{i}]) \wedge \forall j : a'[j] = F_G(\underline{i}, a[\underline{i}], j, a[j])$ where $\phi_L$ (called *guard*) is quantifier-free and $F_G$ is a case-defined function.

**Definition 5.16** (Safety problem)**.** *Given an array-based system $\mathcal{S} = (\underline{a}, I, \tau)$ and an $\exists^I$-formula $U(a)$, an instance of the safety problem is to establish if the formula*

$$I(a_0) \wedge \tau(a_0, a_1) \wedge \cdots \wedge \tau(a_{n-1}, a_n) \wedge U(a_n)$$

*is $A_I^E$-satisfiable for some natural number $n$.*

We say that $\mathcal{S}$ is *safe* w.r.t. $U(a)$ if there is no such $n$, otherwise it is *unsafe*.

**Definition 5.17** (n-pre-image)**.** *Given $n \geqslant 0$ and a formula $K(a)$, the n-pre-image of $K(a)$ is inductively defined as follows:*

$$Pre^0(\tau, K) := K$$

$$Pre^{n+1}(\tau, K) := Pre(\tau, Pre^n(\tau, K))$$

*where*

$$Pre(\tau, K) := \exists a'.(\tau(a, a') \wedge K(a'))$$

Given an array-based system $\mathcal{S} = (\underline{a}, I, \tau)$ and a formula $U(a)$, the formula $Pre^n(\tau, K)$ describes the set of states that are backward reachable in $n$ steps. To check if $U$ is safe, we use the backward reachability algorithm (`BReach`).

At the end of $n$-th iteration, `BReach` stores the formula $BR^n(\tau, U) := \bigvee_{i=0}^{n} Pre^i(\tau, U)$

**Algorithm 1:** BReach

**input:** ($U : \exists^I$-formula)

$P \leftarrow U$ ;
$B \leftarrow \bot$ ;
**while** *($P \wedge \neg B$ is $A_I^E$-sat)* **do**
  **if** *($I \wedge P$ is $A_I^E$-sat)* **then**
    **return** *unsafe*;
  **end**
  $B \leftarrow P \vee B$;
  $P \leftarrow Pre(\tau, P)$
**end**
**return** *(safe,B)*

in the variable $B$, that hence contains exactly the states that are backward reachable from the states in $U$ in at most $n$ steps, while the formula $Pre^{n+1}(\tau, U)$ is stored in $P$. If at some point $I \wedge P$ is $A_I^E$-sat (i.e. $I \wedge Pre^n(\tau, U)$ is satisfiable in $A_I^E$) the algorithm gives an unsat output. Moreover, at every steps it checks if a fix-point has been reached, since ($P \wedge \neg B$ is $A_I^E$-sat) is false if and only if $BR^{n+1}(\tau, U) \rightarrow BR^n(\tau, U)$ is valid in $A_I^E$. If the algorithm terminates, the variable $B$ expresses the formula that describes the set of states which are backward reachable form $U$, that is a fix-point.

In order for `BReach` to be a true procedure, we need to check that:

(i) $\exists^I$-formulas are closed under pre-image

(ii) The $A_I^E$-satisfiability test for safety and the $A_I^E$-satisfiability test for fix-point are effective

For (i), it suffices to use the following result:

**Proposition 4.** *Given a formula $K(a)$ of the form $\exists \underline{k} \phi(\underline{k}, a[\underline{k}])$ and $\tau(a, a') := \bigvee_{h=1}^{m} \exists \underline{i}(\phi_L^h(\underline{i}, a[\underline{i}]) \wedge a' = \lambda j. F_G^h(\underline{i}, a[\underline{i}], j, a[j]))$, then $Pre(\tau, K)$ is $A_I^E$-equivalent to an effective computable $\exists^I$-formula.*

*Proof.* In [7] section 3.2. □

For (ii), we need a decidability procedure for the fragment $\exists^* \forall^*$. The existence of this procedure strongly depends on the index theory used.

## 5.4 Formalization of token-passing rings

Now we want to formalize token-passing rings as array-based systems. The index theory $T_I$ is the theory of finite rings.

The theory of finite rings is defined as follows:

- Signature $\Sigma$:
  - *function symbols*: $s$, $p$ (both of arity 1)
  - *predicate symbols*: $=$
- Class of $\Sigma$-structures: It contains every structure of the form $(\{1, ..., n\}, \mathcal{I})$ such that:
  - $n \in \mathbb{N} - \{0\}$
  - $s^{\mathcal{I}}(i) \equiv i + 1 \mod n$
  - $p^{\mathcal{I}} \equiv i - 1 \mod n$

The elements theory $T_E$ is a multi-sorted theory, having the two enumerated sorts $Bool = \{0, 1\}$ and $States = \{q_0, \cdots, q_n\}$. Hence, the two array variables are:

- $tok : \texttt{INDEX} \rightarrow \{0, 1\}$
- $a : \texttt{INDEX} \rightarrow \{q_0 \cdots, q_n\}$

where $tok[i] = 1$ means that $i$ has the token and $q_0, ..., q_n$ are the internal states.

We can now express the initial state assignment with the formula

$$I : \ \forall i \, (a[i] = q_0 \wedge tok[i] = 0)$$

and the transitions with the formula

$$\tau : \ \exists i_1, \cdots, \exists i_k \begin{pmatrix} Guard(i_1, \cdots, i_k) & \wedge \\ \lambda j Update(j, i_1, \cdots, i_k)) & \end{pmatrix}$$

where $Update$ is a function dependent only by $i_1, \cdots, i_k$ that assigns univocally the values of $a'[j]$ and $tok'[j]$ and that is expressed as an exclusive disjunction of cases where each case is a quantifier-free formula. Moreover, $Update$ is required to assign the token to exactly one process.

We have now defined an array-based system for token-passing rings. We would like to implement a procedure for backward reachability to test safety properties. However, in order to do so, we need first to establish a decidability

procedure for the fragment $\exists^*\forall^*$ of the theory of arrays over finite rings.

This will be done in two ways: first by translating the problem into a S1S problem, secondly using a direct approach. The rest of the thesis is dedicated to prove the decidability of this fragment.

Given these results, the Algorithm `BReach` is proven to be an effective procedure for solving safety problems. However, we have not shown that the algorithm terminates. In [7], termination is obtained by imposing model-theoretic conditions on $T_E^I$, but these conditions do not hold for arrays over finite rings.

A future development of this project would be to find suitable conditions, possibly by taking inspiration from the cut-offs technique treated in the first part of the thesis.

# 6 Theory of arrays over finite rings and MSOL

Second-order logic is an extension of first-order logic which gives the possibility to quantify over relations.

Monadic second-order logic (MSOL) is a restriction of second-order logic, which allows second-order quantifiers only over unary relation (i.e. sets). Two important theories are S2S and S1S. While S2S is defined over infinite binary trees having a successor function of arity two, S1S is defined over natural numbers with the usual 1-arity successor function.

Since we are dealing with directed rings, i.e. graphs where every vertex has only one predecessor and one successor (plus a modular structure that we will see later how to treat), S1S seems to be a good choice for encoding the theory of arrays over finite rings in MSOL, being expressive enough[4].

## 6.1 Introduction of S1S

S1S is the second-order logic interpreted on the naturals, having the constant 0, the successor function symbol $s$[5], and only unary predicate variables.

Formally:

- If $x$ is a first-order variable and $P$ is a second-order variable, then $P(x)$ is a formula
- If $x$ and $y$ are first-order variables, then $x = y$ and $succ(x, y)$ are formulas
- If $\phi$ and $\psi$ are formulas, then $\phi \wedge \psi$, $\phi \vee \psi$, and $\neg\phi$ are formulas
- If $x$ is a first-order variable and $\phi$ is a formula, then $\exists x\phi$ is a formula
- If $P$ is a second-order variable and $\phi$ is a formula, then $\exists P\phi$ is a formula

Note: in the definition above, we are treating the successor $s$ as a 2-arity predicate. However, in order to use a notation more similar to the one of the previous chapters, we would like to write $s(x) = y$ instead of $succ(x, y)$, i.e. expressing the successor as a 1-arity function. So we will do, since the formulas expressible in the two languages are the same[11].

Given a finite number of second-order variables $P_1, \cdots, P_N$, an *assignment* $\sigma$ is a function that maps every $P_i$ into $\sigma(P_i) \subseteq \mathbb{N}$.
Now we introduce the Büchi automata, that play an important role in showing a decidability procedure for S1S.

[4]For an extensive description of the properties of S1S and S2S, we refer to [8] (which also provides a theoretical framework for SMOL)

[5]The successor function could be avoided by introducing a 2-arity predicate symbol $S$, interpreted as $S(x, y) : \iff s(x) = y$. However, these two languages are interchangable.

**Definition 6.1** (Büchi automaton)**.** *A Büchi automaton is a 5-tuple* $A = (\Sigma, Q, q_I, \delta, F)$ *where*

- $\Sigma$ *is a finite set, called alphabet of* $A$
- $Q$ *is a finite set, called set of the states of* $A$
- $q_I \in Q$ *is the initial state of* $A$
- $\delta : \Sigma \times Q \to \mathcal{P}(Q)$ *is the transition of* $A$
- $F \subseteq Q$ *is the set of final states of* $A$

**Definition 6.2** (Run)**.** *Given a* $\omega$*-word* $\sigma \in \Sigma^\omega$*, a run over* $\sigma$ *is an infinite sequence of states*

$$\rho = \rho(0), \rho(1), ...$$

*such that* $\rho(0) = q_I$ *and* $\rho(i+1) \in \delta(\sigma(i), \rho(i))$ *for every* $i \in \mathbb{N}$*.*

**Definition 6.3** (Acceptance)**.** *A is said to accept a run* $\rho$ *iff there exists* $q \in F$ *such that* $q$ *occurs infinitely many times in* $\rho(0), \rho(1), ....$

Given a Büchi automaton $A$, with $L(A)$ we indicate the $\omega$-language including exactly the infinite words of $\Sigma^\omega$ that are accepted by $A$. It can be proved that checking if $L(A) = \varnothing$ is in NLOGSPACE.

Büchi automata and S1S are connected by the following theorem:

**Theorem 6.1.** *Given a formula* $\phi$ *in S1S, there exists a Büchi automaton* $A_\phi$ *such that* $\phi$ *is satisfiable in* $\mathbb{N}$ *if and only if* $L(A_\phi) \neq \varnothing$*.*

A detailed proof can be found in [8]. This results implies that S1S is decidable.

## 6.2 Encoding the theory of arrays over finite rings into S1S

First, we apply the following rules:

- For all variables $x$ and $y$ and $n > 0$, $s^n(s(x)) = y$ is replaced by $\forall w(s(x) = w \rightarrow s^n(w) = y)$
- For all $x$ and $n > 0$, $a[s^n(x)] = e_k$ is replaced by $\forall w(s(x) = w \rightarrow a[s^{n-1}(w)] = e_k)$

By applying these rules, we obtain a formula containing at most first-grade powers of the successor function symbol $s$.

Now, a model for the theory of arrays over finite rings is a finite ring of dimension $d$ in which every vertex has a value $e_k \in \{e_1, \cdots, e_{n_e}\}$. The central idea of the encoding is to express this model as a subset of $\mathbb{N}$ of the form $\{0, 1, \cdots, N\}$ (with $N = k - 1$) having a partition $\{C_1, \cdots, C_{n_e}\}$, where $n \in C_i$ represent that $a[n] = e_i$, and where the successor of $N$ is 0.

First, note that, by using the heterodox definition of theory given in 5.3, we are implicitly stating that a formula is satisfiable iff there exists an $N \in \mathbb{N}$

We define a translation $\tau$ that maps a formula $\phi$ of the theory of arrays over finite rings into a formula $\tau(\phi)$ of S1S. First, we inductively define a partial translation $\tau'$ as follows:

- $\tau' : s(x_1) = x_2 \ \mapsto \ (x_1 < N \wedge x_2 = s(x_1)) \vee (x_1 = N \wedge x_2 = 0)$
- $\tau' : a[x] = e_j \ \mapsto \ C_j(x)$
- $\tau' : \forall x \phi \ \mapsto \ \forall x \leqslant N : \tau'(\phi)$, where $\phi$ does not contain existential quantifiers
- $\tau' : \exists y \phi \ \mapsto \ \exists y \leqslant N : \tau'(\phi)$

Then:

$$\tau \ : \ \phi \ \mapsto \ \exists N > 0 : \big(\exists C_1 \cdots \exists C_{n_e} : Partition_N(C_1, \cdots, C_{n_e}) \ \wedge \ \tau'(\phi)\big)$$

where

$$Partition_N(C_1, \cdots, C_{n_e}) \equiv \forall n < N : \big( \bigvee_{1 \leqslant i \leqslant n_e} C_i(n) \ \wedge \bigwedge_{\substack{1 \leqslant i \leqslant n_e \\ 1 \leqslant j \leqslant n_e \\ j \neq i}} \neg C_i(n) \vee \neq C_j(n)\big)$$

**Theorem 6.2.** *An $\exists^*\forall^*$-formula $\phi$ is satisfiable in the theory of arrays over finite rings if and only if $\tau(\phi)$ is satisfiable in S1S.*

*Proof.* ( $\Longrightarrow$ ) Suppose $\phi$ is satisfiable, i.e. there exists a finite ring having dimension $d$ and a *coloration* $\alpha : [d] \to \{e_1, \cdots, e_n\}$ in which $\phi$ is valid, and call this model $M$. Define the assignment $\sigma$ as follows:

- $\sigma(C_j) = \{n \in \mathbb{N} : \alpha(n) = e_j\}$ for every $j$.
- $\sigma(N) = d$

Let's prove inductively that $\tau'$ does not alter the validity of a formula:

- If $M \models s(x_1) = x_2$, then $x_1^M + 1 = x_2^M \pmod d$. Since $x_i^M \in [d]$, this is true if and only if either $\sigma(x_1) < d$ and $\sigma(x_2) = \sigma(x_1) + 1$, or $\sigma(x_1) = d$ and $\sigma(x_2) = 0$, that is $\sigma \models \tau'(s(x_1) = x_2)$.
- If $M \models a[x] = e_j$, then $\alpha(x^M) = e_j$, that is true if and only if $\sigma \models C_j(x)$.
- If $M \models \forall x\phi$ for $\phi$ without existential quantifiers, then for every $n \in [d]$ it holds that $M \models \phi(n/x)$, that by inductive hypothesis is equivalent to $\sigma \models \tau'(\phi(n/x))$ for every $n \in \{0, d\}$, i.e. $\sigma \models \exists x \leqslant N : \tau'(\phi)$.
- If $M \models \exists y\phi$, then exists $n \in [d]$ such that $M \models \phi(n/y)$, that by inductive hypothesis is equivalent to $\sigma \models \tau'(\phi(n/y))$, i.e. $\sigma \models \exists y \leqslant N : \tau'(\phi)$.

( $\Longleftarrow$ ) Suppose $\tau(\phi)$ is satisfiable in S1S, i.e. there exists an assignment $\sigma$. Define the colored graph $M$ as follows:

- $d := \sigma(N)$
- for $n \in [d]$, $\alpha(n) := e_j$ such that $\sigma \models C_j(n)$ (it is unique since $C_1, \cdots, C_{n_e}$ is a partition)

The proof that $M$ is a model is the same as before read backwards.

□

# 7 Decidability of the $\exists^*\forall^*$-fragment of the theory of finite rings

The theory of finite rings is defined as follows:

- Signature $\Sigma$:
  - *function symbols*: $s$, $p$ (both of arity 1)
  - *predicate symbols*: $=$
- Class of $\Sigma$-structures: It contains every structure of the form $(\{1, ..., n\}, \mathcal{I})$ such that:
  - $n \in \mathbb{N} - \{0\}$
  - $s^{\mathcal{I}}(i) \equiv i + 1 \mod n$
  - $p^{\mathcal{I}} \equiv i - 1 \mod n$

We consider the problem of determining whether a finite set of literals is satisfiable or not. Explicitly we have the following:
Problem **P**:

- Input: A finite set of literals $F$ in the language of $\Sigma$
- Output: **Yes** if $F$ has a model, **No** otherwise.

## 7.1 Decidability for finite sets of literals

**Theorem 7.1.** *The problem **P** is decidable.*

*Proof.* Let $F$ be any finite set of literals. It can be seen as a union of two sets $P$ and $Q$ of the form $P = \{s_i = t_i\}_I$ and $N = \{s_j \neq t_j\}_j$, with $i \in I$ and $j \in J$. By replacing the variables with new constants, we can suppose that $s_i$ and $t_i$ are ground for every $i \in I \cup J$.

Let's manipulate these two sets by replacing and removing literals by applying some operations that preserve equisatisfiability. With a slight abuse of notation, we will continue to call the sets $P$ and $N$ even after the modifications.

Rewriting rules are the following:

1. Every occurrence of $s(p(t))$ and every occurrence of $p(s(t))$ are replaced by $t$.
2. Every literal of the form $p(t_1) = p(t_2)$ is replaced with $t_1 = t_2$, and every literal of the form $p(t_1) \neq p(t_2)$ is replaced with $t_1 \neq t_2$.
3. Every literal of the form $t_1 = p(t_2)$ or $p(t_2) = t_1$ is replaced by $s(t_1) = t_2$, and every literal of the form $t_1 \neq p(t_2)$ or $p(t_2) \neq t_1$ is replaced by $s(t_1) \neq t_2$.

4. Every literal of the form $s(t_1) = s(t_2)$ is replaced by $t_1 = t_2$, and every literal of the form $s(t_1) \neq s(t_2)$ is replaced by $t_1 \neq t_2$.

5. For every literal of the form $a = b$ (where $a$ and $b$ are constants), replace every other occurrence of $b$ with $a$ and then remove the literal $a = b$.

6. Per every literal of the form $s^n(a) = b$ such that $a \neq b$, replace every other occurrence of $b$ with $s^n(a)$ and then remove the literal $s^n(a) = b$.

Let's apply the rules 1, 2, and 3 until no more rule can be applied. After that, the symbol $p$ does not appear anywhere.

Now let's apply rules 4, 5 and 6 until no more rule can be applied. At this point, the set $P$ contains only literals of the form $s^n(a) = a$, where $n > 0$.

Indeed, we got $P = \{s^{n_i}(a_i) = a_i\}_{i \in I}$, where $I$ is finite (possibly empty). The strategy is to determine effectively $\bar{d}$ such that if $P \cup N$ is satisfiable, then there is a model whose dimension is less then or equal to $\bar{d}$. This implies that the problem is decidable, since there is only a finite number of candidate models (having finite cardinality) to check. There are two cases:

- $P \neq \varnothing$. If there exists a model that satisfies $P \cup N$, it satisfies $P$ too, hence every literal $s^{n_i}(a_i) = a_i$ must be valid in that model, and this tells us that the dimension $d$ of the model is such that $d | n_i$ for every $i$. Therefore $d \big| \gcd(n_1, ..., n_{|I|})$. Thus the set of the candidate models is restricted to the rings whose dimension $d$ is such that $d \big| \gcd(n_1, ..., n_{|I|})$, i.e. to a finite number of candidate models of finite cardinality.

- $P = \varnothing$. If $N$ is empty, also $F$ is empty, thus it has a model. Suppose, without loss of generality, that $N$ is of the form $N = \{s^{m_j}(a_j) \neq b_j\}_{j \in J}$. Say $\mu := \sum_{j \in J}(m_j + 1)$ . We show that if there exists a model that satisfies $N$, then there is a model that satisfies $N$ whose dimension is $d \leqslant \mu$.
  In fact, suppose there is a model $M = ([d_M], \mathcal{I}_M)$ such that $d_M > \mu$. If we prove that there is an element that can be *removed* from the ring without losing the satisfiability, we are done, since this operation can be done $(d_M - \mu)$-times, obtaining at the end a model $\bar{M} = ([d_{\bar{M}}], \mathcal{I}_{\bar{M}})$ with $d_{M'} \leqslant \mu$.
  Let's say that an element $i \in [d_{M'}]$ of the ring is *bounded* if there exists a $j \in J$ such that $i \in [\mathcal{I}(a_j)\ ,\ \mathcal{I}(s^{m_j}(a_j))]$ [6]. In $M$ there is at least one element which is not bounded. In fact, for every $j \in J$, exactly the $m_j$ elements $(\mathcal{I}(a_j)\ ,\ \mathcal{I}s(a_j)\ ...\ \mathcal{I}(s^{m_j}(a_j)))$ are bounded. Therefore at most $\sum_{j \in J} m_j = \mu$ elements are bounded.
  Given a non bounded element $i$, we can construct a new model $M' = ([d_{M'}], \mathcal{I}_{M'})$, where $d_{M'} = d_M - 1$ and for every variable $c$
  $$\mathcal{I}_{M'}(c) = \begin{cases} \mathcal{I}_M(c), & \text{if } \mathcal{I}_M(c) < i \\ \mathcal{I}_M(c) - 1, & \text{if } \mathcal{I}_M(c) \geqslant i \end{cases}.$$
  Every disequality still holds, so the new structure is a model too.

[6] With $[x, y]$ we mean the clockwise segment of the ring starting at $x$ and ending at $y$

Having restricted the satisfiability problem to check a finite number of possible models, the decidability follows. □

## 7.2 Satisfiability of a quantified fragment

Let's consider a formula of the form

$$\exists \underline{y} \forall x A$$

where $x$ is a single variable.

After having skolemized all the existential quantifiers, we write $A$ in CNF and we distribute the universal quantifier over the conjunctions, obtaining a finite number of formulas of the form (*)

$$\forall x(\neg A_1 \vee \cdots \vee \neg A_n \vee B_1 \vee \cdots \vee B_m)$$

In order to find a model of the starting formula, it is necessary to find a model that satisfies every formula obtained through this process. Let's analize them individually.

First, note that we can move outside the scope of the quantifier every literal $\neg A_i$ e i $B_i$ in which do not appear $x$. In fact $\forall x(\phi \vee \psi)$ is equivalent to $\phi \vee \forall x(\psi)$ if $x$ do not appear in $\phi$. With a slight abuse of notation, let still write (*) to indicate the formula inside the scope of the quantifier.
Every $A_i$ and $B_j$ can have one of the following forms:

a) $s^n(c) = x$ or $p^n(c) = x$

b) $s^n(x) = x$

Let's apply the following rules:

1. If $A_i$ has the form $s^n(c) = x$ (or $p^n(c) = x$), rewrite (*) as

   $$\neg A_2(t/x) \vee \cdots \neg A_n(t/x) \vee B_1(t/x) \vee \cdots \vee B_m(t/x)$$

   where $t$ is $s^n(c)$ (or $p^n(c)$). This is an equivalent formula, since (*) can be first written as

   $$\forall x(A_1 \rightarrow \neg A_2 \vee \cdots \neg A_n \vee B_1 \vee \cdots \vee B_m)$$

   and then it is possible to apply the logic rule "$\forall x(x = t \rightarrow \phi)$ is equivalent to $\phi(t/x)$ if $t$ do not contains $x$".

2. Given the $B_i$ of the form $s^{n_i}(x) = x$, take $k := max_i(n_i)$ (where $i$ is iterated only on the $B_i$ of this form). Test the satisfiability in the rings having dimension $\leqslant k$. It this test gives a *sat* output, a model is found; if it gives an *unsat* output, we replace every $B_i$ of the form $s^{n_i}(x) = x$ with $\perp$ and test the models having dimension $\geqslant k$.

3. If $A_i$ has the form $s^n(x) = x$ (hence $\neg A_i \equiv s^n(x) \neq x$), every ring having dimension not dividing $n$ is a model. The rings having dimension dividing $n$ are finitely many, so the satisfiability can be tested individually by substituting $\neg A_i$ with $\bot$.

4. If the algorithm has not terminated yet, (*) is of the form

$$\forall x(B_1 \vee \cdots \vee B_m)$$

where every $B_i$ has the form $s^n(c) = x$ or $p^n(c) = x$. So (*) can be rewritten as:

$$\forall x(t_1 = x \vee \cdots \vee t_m = x)$$

where $t_i$ is $s^{n_i}(c_i)$ or $p^{n_i}(c_i)$. Then it suffices to test the satisfiability in the models having dimension $\leqslant m$. In fact, given a model that satisfies the formula, $x$ can take at most $m$ different values.
If the satisfiability test fails, the formula is replaced by $\bot$.

After having applied these rules, either a model is found or only quantifier-free formulas remained, hence reducing the satisfiability problem to the one of the previous section.

In case the original formula has more than a single universal quantifiers, i.e. it has the form $\exists \underline{y} \forall x_n \cdots \forall x_1 A$, we apply the same procedure by induction, treating the more external variables $(x_n \cdots x_2)$ as constants.

# 8 Decidability of the $\exists^*\forall^*$-fragment of the theory of arrays over finite rings

## 8.1 Decidability for finite sets of literals

We consider the case where there is only an array $a$, and the new atoms are all the ones of the form $a[c] = e$.

Let $F$ be a set of literals of the form $t(c) = t'(c')$ or $a[c] = e$ (or their negation).
Consider the problem **P2**: "*Does $F$ have a model?*".

**Theorem 8.1.** *The problem **P2** is decidable.*

*Proof.* $F$ can be seen as a union of four sets:

- $P = \{t_i(c_i) = t'_i(c'_i))\}_I$
- $N = \{t_j(c_j) \neq t'_j(c'_j)\}_J$
- $P' = \{a[t_k(c_k)] = e_k\}_K$
- $N' = \{a[t_l(c_l)] \neq e_l\}_L$

By the same argument of the theorem 8.1, if $P$ is not empty, decidability follows.

Suppose $P = \varnothing$. The idea is the same of the previous theorem: to effectively determine an integer $\mu$ such that if there exists a model that satisfies $F$, then there is a model with dimension $\leqslant \mu$ that satisfies $F$.
Being every term $t_i$ of the form $s^{m_i}(c_i)$ or $p^{m_i}(c_i)$ and every term $t'_i$ of the form $s^{m'_i}(c'_i)$ or $p^{m'_i}(c'_i)$, it is possible to define

$$\mu := \sum_{j\in J}(m_j + 1 + m'_j + 1) + \sum_{k\in K}(m_k + 1) + \sum_{l\in L}(m_l + 1)$$

Suppose there is a model $M$ with dimension $d > \mu$. There is at least an element that can be removed from the model without affecting the validity of the formulas in $F$. Hence it is possible to define a new model with dimension $d' = d - 1$. This operation can be done until the dimension of the new model is $\leqslant \mu$. □

## 8.2 With one quantifier

Let's consider a formula of the form

$$\exists \underline{y} \forall x A$$

where $A$ is a formula without quantifiers and $x$ is a single index variable.

As done in the section 8.2, the existential quantifiers can be skolemized, $A$ can be written in CNF, and the universal quantifiers can be distributed over the conjunctions.

We obtain formulas of form (*):

$$\forall x(\neg A_1 \vee \cdots \vee \neg A_n \vee B_1 \vee \cdots \vee B_m)$$

when $A_i$ and $B_j$ are atoms.

Now apply rules 1, 2, and 3 from section 8.2, plus a new rule

5. Every literal of the form $a[t(x)] \neq e_i$ is replaced by

$$\bigvee_{\substack{1 \leqslant j \leqslant n_e \\ j \neq i}} a[t(x)] = e_j$$

After having applied these rules, if the quantifier is not removed, we got formulas of the form:

$$\forall x(t'_1 = x \ \vee \cdots \vee \ t'_m = x \ \vee \ a[t_1(x)] = e_{l_1} \vee \cdots \vee a[t_{m'}(x)] = e_{l_{m'}})$$

Let's use the convention that $s^{-m}(x)$ is interchangeable with $p^m(x)$ and $s^0(x)$ is interchangeable with $x$.

---

### 8.2.1 Particular case

For now, consider the restricted case where $A$ is in CNF

$$A \ \equiv \ \forall x D_1 \wedge \cdots \wedge \forall x D_{n'}$$

and every $D_k$ is of the form

$$a[s^{n_k(1)}(x)] = e_{k(1)} \vee \cdots \vee a[s^{n_k(m_k)}(x)] = e_{k(m_k)}$$

with $n_k(i) \leqslant n_k(i')$ for $i < i'$.

Since the quantifier is distributed over every $D_k$, we can apply the following rule:

6a. Given $D_k$, let $d_k := -n_k(1)$. Replace every $s^{n_k(i)}$ with $s^{n_k(i)+d_k}$.

This way, for every $D_k$, the first literal in the disjunction is $a[x] = e_{k(1)}$, and every $n_k(i)$ is positive. Basically, we got rid of all the predecessor function symbols and of the redundant successor function symbols.

Let's transform the formula into DNF:

$$A \equiv \forall x(C_1 \vee \cdots \vee C_n)$$

Observe that there could be some $C_k$ (statistically, a lots) containing subformulas like $a[s^n(x)] = e_j \wedge a[s^n(x)] = e_{j'}$ with $e_j \neq e_{j'}$. Apply the following rule:

7. If a $C_k$ contains both $a[s^n(x)] = e_j$ and $a[s^n(x)] = e_{j'}$, with $e_j \neq e_{j'}$, remove $C_k$.

After having applied the rule exhaustively, every remaining $C_k$ has the form

$$a[s^{n_k(1)}(x)] = e_{k(1)} \wedge \cdots \wedge a[s^{n_k(m_k)}(x)] = e_{k(m_k)}$$

where $n_k(i) \neq n_k(i')$ for every $i \neq i'$.

Let $M := \max_{k,i} n_k(i)$.

Given a $C_k$, it is possible to construct a one-to-one correspondence between $C_k$ and a partial function having domain $\{0, 1, \cdots, M\}$ and codomain the set of the elements $\{1, 2, ..., n_e\}$.

*Example*: suppose $M = 4$, $n_e = 2$, and consider the formula

$$a[x] = 2 \wedge a[s^2(x)] = 1 \wedge a[s^3(x)] = 2$$

It can be seen as the tuple:

$$(2,\ -,\ 1,\ 2,\ -)_x$$

In order to improve the readability, the latter notation will be used most of the time, using implicitly the one-to-one correspondence.

At a logic level, $(2,\ -,\ 1,\ 2,\ -)_x$ is equivalent to

$$\begin{aligned}
&(2,\ \textit{1},\ 1,\ 2,\ \textit{1})_x \vee \\
&(2,\ \textit{1},\ 1,\ 2,\ \textit{2})_x \vee \\
&(2,\ \textit{2},\ 1,\ 2,\ \textit{1})_x \vee \\
&(2,\ \textit{2},\ 1,\ 2,\ \textit{2})_x
\end{aligned}$$

This operation can be done for every $K_i$, obtaining a disjunction of total functions having same domain and same codomain.

Let $M$ be any model of $A$. Given an index $i$, by the semantic of the disjunction there exists a $C_k \cong (e_0, e_1, ..., e_M)$ such that

$$M \models (e_0, e_1, ..., e_M)_i$$

Given $i+1$, there exists a $C_{k'} \cong (e'_0, e'_1, ..., e'_M)$ such that

$$M \models (e'_0, e'_1, ..., e'_M)_{i+1}$$

and such that

$$(e_1, e_2..., e_M) = (e'_0, e'_1, ..., e'_{M-1}) \qquad (\dagger)$$

Since this reasoning applies for every model and for every index, we can deduce the following rule:

8a. Given a tuple $(e_0, e_1, ..., e_M)$, if there is no tuple $(e'_0, e'_1, ..., e'_M)$ such that $(e_1, e_2..., e_M) = (e'_0, e'_1, ..., e'_{M-1})$, then the tuple $(e_0, e_1, ..., e_M)$ can be removed.

After having applied this rule exhaustively, there are two scenarios:

- Every tuple has been removed. The algorithm gives an *unsat* output.
- Some tuples remain. This means that there exists a *cycle*: given any tuple $C_{k_1}$, there exists $C_{k_2}$ such that they satisfy the property (†). For $C_{k_2}$, there exists $C_{k_3}$ such that they satisfy the property (†), and so on, until a $C_{k_i}$ is repeated. Following the *cycle*, it is easy to construct a model. The algorithm gives a *sat* output.

### 8.2.2 General case

Now, consider the case where $A$ is in CNF

$$A \equiv \forall x D_1 \wedge \cdots \wedge \forall x D_{n'}$$

and every $D_k$ is of the form

$$a[s^{n_k(1)}(x)] = e_{k(1)} \vee \cdots \vee a[s^{n_k(m_k)}(x)] = e_{k(m_k)} \vee s^{n_{k'}(1)}(x) = c_{k'(1)} \vee \cdots \vee s^{n_{k'}(m_{k'})}(x) = c_{k(m_{k'})}$$

with $n_k(i) \leqslant n_k(j)$ and $n_{k'}(i) \leqslant n_{k'}(j)$ for $i < j$.

Since the quantifier is distributed over every $D_k$, we can apply the following rule:

6b. Given $D_k$, let $d_k := -min(n_k(1), n_{k'}(1))$. Replace every $s^{n_k(i)}$ with $s^{n_k(i)+d_k}$ and every $s^{n_{k'}(i)}$ with $s^{n_{k'}(i)+d_k}$ .

Let's transform the formula into DNF:

$$A \equiv \forall x(C_1 \vee \cdots \vee C_n)$$

and remove subformulas equivalent to $\perp$ (rule 7).

Every remaining $C_k$ has the form

$$a[s^{n_k(1)}(x)] = e_{k(1)} \wedge \cdots \wedge a[s^{n_k(m_k)}(x)] = e_{k(m_k)} \wedge s^{n_{k'}(1)}(x) = c_{k'(1)} \wedge \cdots \wedge s^{n_{k'}(m_{k'})}(x) = c_{k(m_{k'})}$$

where $n_k(i) \neq n_k(j)$ for every $i \neq j$. Note that the same does not necessarily hold for every $n_{k'}(i)$ and $n_{k'}(j)$.

Let $M := \max_{k,i} \max(n_k(i), n_{k'}(i))$.

Let $Const$ be the set containing every constant that appears in the formula $A$. Given a $C_k$, it is possible to construct a one-to-one correspondence between $C_k$ and a partial function having domain $\{0, 1, \cdots, M\}$ and codomain the set $\{1, 2, ..., n_e\} \times \mathcal{P}(Const)$.

*Example*: suppose $M = 3$, $n_e = 2$, $Const = \{c, c', c''\}$ and consider the formula

$$a[x] = 2 \wedge a[s^2(x)] = 1 \wedge a[s^3(x)] = 2 \wedge x = c \wedge s(x) = c' \wedge s^3(x) = c \wedge s^3(x) = c'$$

It can be seen as the (2,M+1)-matrix:

$$\begin{pmatrix} \{c\} & \{c'\} & \varnothing & \{c, c'\} \\ 2 & - & 1 & 2 \end{pmatrix}_x$$

which is logically equivalent to

$$\begin{pmatrix} \{c\} & \{c'\} & \varnothing & \{c, c'\} \\ 2 & \mathit{1} & 1 & 2 \end{pmatrix}_x \vee \begin{pmatrix} \{c\} & \{c'\} & \varnothing & \{c, c'\} \\ 2 & \mathit{2} & 1 & 2 \end{pmatrix}_x$$

This corrispondence is not expressive enough. In fact, note that the $i$-th entry of the first row of a matrix is expressing that $s^i(x)$ has to be equal to *at least* the constants in the entry. Indeed, $s^i(x) = c \wedge s^i(x) = c'$ does not implies that $s^i(x) \neq c''$.

In order to avoid this ambiguity, let's replace a matrix with the logically equivalent disjunction of matrices such that every $i$-th entry of the first row expresses exactly the set of the constants that $s^i(x)$ is equal to.

*Example*: suppose $M = 1$, $n_e = 2$, $Const = \{c, c'\}$ and consider the formula

$$a[x] = 2 \wedge \ a[s(x)] = 1 \ \wedge \ s(x) = c'$$

We have that $s(x) = c$ is logically equivalent to $(s(x) = c' \wedge s(x) = c) \vee (s(x) = c' \wedge s(x) \neq c)$ and there are no conditions on $x$, i.e. $(x = c \wedge x = c') \vee (x = c \wedge x \neq c') \vee (x \neq c \wedge x = c') \vee (x \neq c \wedge x \neq c')$

Hence the formula is expressible by:

$$\begin{pmatrix} \{c,c'\} & \{c'\} \\ 2 & 1 \end{pmatrix}_x \vee \begin{pmatrix} \{c,c'\} & \{c,c'\} \\ 2 & 1 \end{pmatrix}_x \vee$$
$$\begin{pmatrix} \{c\} & \{c'\} \\ 2 & 1 \end{pmatrix}_x \vee \begin{pmatrix} \{c\} & \{c,c'\} \\ 2 & 1 \end{pmatrix}_x \vee$$
$$\begin{pmatrix} \{c'\} & \{c'\} \\ 2 & 1 \end{pmatrix}_x \vee \begin{pmatrix} \{c'\} & \{c,c'\} \\ 2 & 1 \end{pmatrix}_x \vee$$
$$\begin{pmatrix} \varnothing & \{c'\} \\ 2 & 1 \end{pmatrix}_x \vee \begin{pmatrix} \varnothing & \{c,c'\} \\ 2 & 1 \end{pmatrix}_x$$

where the $i-th$ entry of the first column of each matrix express *exactly* which constants $s^i(x)$ is equal to.

Let $M$ be any model of $A$. Given an index $i$, by the semantic of the disjunction there exists a $C_k \cong \begin{pmatrix} S_0 & \cdots & S_M \\ e_0 & \cdots & e_M \end{pmatrix}$ such that

$$M \models \begin{pmatrix} S_0 & \cdots & S_M \\ e_0 & \cdots & e_M \end{pmatrix}_i$$

Given $i+1$, there exists a $C_{k'} \cong \begin{pmatrix} S'_0 & \cdots & S'_M \\ e'_0 & \cdots & e'_M \end{pmatrix}_{i+1}$ such that

$$M \models \begin{pmatrix} S'_0 & \cdots & S'_M \\ e'_0 & \cdots & e'_M \end{pmatrix}_{i+1}$$

and such that

$$\begin{pmatrix} S_1 & S_2 & \cdots & S_M \\ e_1 & e_2 & \cdots & e_M \end{pmatrix} = \begin{pmatrix} S'_0 & S'_1 & \cdots & S'_{M-1} \\ e'_0 & e'_1 & \cdots & e'_{M-1} \end{pmatrix} \qquad (\dagger)$$

Since this reasoning applies for every model and for every index, we can deduce the following rule:

8b. Given a matrix $\begin{pmatrix} S_0 & \cdots & S_M \\ e_0 & \cdots & e_M \end{pmatrix}$, if there is no matrix $\begin{pmatrix} S'_0 & \cdots & S'_M \\ e'_0 & \cdots & e'_M \end{pmatrix}$ such that † holds, then the matrix $\begin{pmatrix} S_0 & \cdots & S_M \\ e_0 & \cdots & e_M \end{pmatrix}$ can be removed.

If, after having applied this rule exhaustively, every matrix has been removed, the algorithm gives an *unsat* output.

Otherwise, consider all the *cycles*, i.e. sequences of the form $C_0, C_1, ..., C_n$ such that $C_i$ and $C_{i+1}$ satisfy the property (†) for every $i \in \{0, \cdots, n-1\}$ and $C_n$ and $C_0$ satisfy it too. If there exists a model, it should be formed by one of these cycles (with an abuse of notation, we could identify the model with the cycle it is obtained from).

If we show that it is possible to find a *bound*, i.e. it is possible to effectively compute a natural number $d$ such that if there exists a model then there exists a model having dimension $\leqslant d$, then we are done.

We say that a matrix $\begin{pmatrix} S_0 & \cdots & S_M \\ e_0 & \cdots & e_M \end{pmatrix}$ is a *matrix without constants* if $S_0 = S_1 = \cdots = S_M = \varnothing$. Otherwise we say it is a *matrix with constants*.

We say that a cycle $\mathcal{C}$ is a *power* of a cycle $\mathcal{C}'$ if $\mathcal{C}$ can be obtained through finitely many concatenations of $\mathcal{C}'$[7].

Consider a cycle

$$\mathcal{C} = C_0, C_1, \cdots, C_n$$

Observe that if it contains a repetition of a matrix with constants, say $C_i = C_j$, then, in order to led to a consistent model, $\mathcal{C}$ must be a power of

$$\mathcal{C}' = C_i, C_{i+1}, \cdots, C_{j-1}, C_j$$

So let's consider only cycles without repetitions of matrices with constants.

What about repetitions of matrices without constants? Suppose that $\mathcal{C}$ has a repetition of a matrix without constants, say $C_i = C_j$. If the subcycle $C_{i+1}, \cdots, C_j$ does not includes a matrix with constants, we can consider the cycle

$$\mathcal{C}'' = C_0, \cdots, C_{i-1}, C_i, C_{j+1}, C_{j+2}, \cdots C_n$$

obtained by removing the subcycle $C_{i+1}, \cdots, C_j$. If $\mathcal{C}$ forms a model, then $\mathcal{C}''$ forms a model too, since in the removed subcycle no constants appear.

Thus, between the repetition of a matrix without constants there should be at least a matrix with constants. Since every matrix with constants cannot appear more than one time, then every matrix without constants can appear at most $m$ times, with $m$ be the number of matrices with constants. Hence, we found a *bound*.

[7]Note that we are implicitly considering $C_0, C_1, \cdots, C_n$ and $C_i, C_{i+1}, \cdots, C_n, C_0, C_1, \cdots, C_{i-1}$ to be the same object.

## 8.3 Handling constants

Let's consider a formula of the form

$$\exists \underline{y} \forall \underline{x} A$$

As done in the section 7.2, the existential quantifiers can be skolemized. In order to deal with these new constants, in particular with literals of the form $x = s^n(c)$, we can introduce new arrays.

Let $c$ be a constant and let $n$ be the maximal number such that $s^n(c)$ appears in $A$. We check all the possible models having dimension $\leqslant n$. If no model is found, we know that it is not possible that $s^m(c) = s^{m'}(c)$ with $m \neq m'$ (with $m, m' \leqslant n$).

Introduce the array symbol $b$ having values in $\{-1, 0, 1, \cdots, n\}$. We should think of $b[x] = j$ as $x = s^j(c)$ for $j \in \{0, \cdots, n\}$, and $b[x] = -1$ as $x \neq c \wedge x \neq s(c) \wedge \cdots \wedge x \neq s^n(c)$.

Note that this new *elements* $-1, \cdots, n$ must satisfy the following relations:

- $\forall x_1 \forall x_2 (b[x1] = b[x2] > -1 \rightarrow x_1 = x_2)$
- $\forall x (n > b[x] > -1 \rightarrow b[s(x)] = b[x] + 1)$
- $\exists x b[x] = 0$

where the symbol $>$ is just a synthetic way to express all the cases. These realtions are constraints that a formula must satisfy in order to be consistent.

We can do so for every constant, since having a finite number of arrays is the same as having an array with values in the cartesian product.

If we have the array $a$ having values in $E_a = \{e_1, \cdots, e_{n_e}\}$ and the array $b$ having values in $E_b = \{-1, 0, 1, \cdots, n\}$, we introduce a new array $\bar{a}$ having values in the cartesian product $E_a \times E_b$.

More precisely, every literal of the form $a[x] = e_j$ is replaced by $\bigvee_{i \in E_b} \bar{a}[x] = (e_j, i)$ and every literal of the form $b[x] = i$ is replaced by $\bigvee_{j \in E_a} \bar{a}[x] = (e_j, i)$. If there are many constants, this procedure is applied as many times.

Having showed this transformation, when dealing with constants we can use only an array $a$ and a set of elements $E_a = \{e_1, \cdots, e_{n_e}\}$, keeping in mind that the meaning of $a$ has changed, and that the constraints written above must be added to the formula of which we are testing the satisfiability.

## 8.4 Some observations on the general case with many universal quantifiers

*Note: this section is a work in progress and still need some improvements.*

Consider the case with more quantifiers. Suppose we have a formula of the form

$$\exists \underline{y} \forall \underline{x} A$$

where $\underline{x} = x_{n_q}, \cdots, x_1$ and $A$ is a formula without quantifiers.

The existential quantifiers can be skolemized. We have two slightly different ways to deal with the new constants. The first is applying the tactic used in section 8.2.2, while the second is using the transformation introduced in the previous section 8.3. In the following we will use the latter, which has the advantage of allowing a cleaner notation. However, with a little ingenuity it will be clear that the first method works as well.

Apply rule 5 plus the following rules:

- $a[t_i(x_i)] = a[t_j(x_j)]$ is replaced by

$$\bigvee_{1 \leqslant k \leqslant n_e} a[t_i(x_i)] = e_k \wedge a[t_j(x_j)] = e_k$$

- $a[t_i(x_i)] \neq a[t_j(x_j)]$ is replaced by

$$\bigvee_{\substack{1 \leqslant k,k' \leqslant n_e \\ k \neq k'}} a[t_i(x_i)] = e_k \wedge a[t_j(x_j)] = e_{k'}$$

Write $A$ in CNF distributing the quantifiers over the conjunctions:

$$A \equiv \forall \underline{x} D_1 \wedge \cdots \wedge \forall \underline{x} D_{n'}$$

and apply rules 1, 2 and 3, plus a new rule:

4c. If there is a $D_k$ of the form

$$D_k \equiv (s^n(x_i) \neq x_j \vee \phi)$$

replace $\forall x_j D_k$ with $\phi(s^n(x_i)/x_j)$

Let's apply:

6c. Given $D_k$, let $d_k := \min\{n \mid s^n \text{ appears in } D_k\}$. Replace every $s^{n_k(i)}$ with $s^{n_k(i)-d_k}$.

Now transform $A$ into DNF:

$$A \equiv \forall \underline{x}(C_1 \vee \cdots \vee C_n)$$

and remove subformulas equivalent to $\perp$ (rule 7).

The remaining atoms are of the form $a[s^n(x_i)] = e_j$ or of the form $s^{n_1}(x_i) = s^{n_2}(x_j)$. The latters can be replaced by $x_i = s^{n_2-n_1}(x_j)$ if $n_2 \geqslant n_1$, or by $s^{n_1-n_2}(x_i) = x_j$ if $n_2 < n_1$.

Every $C_k$ can be written as

$$C_k \equiv C_{k,x_1} \wedge \cdots \wedge C_{k,x_{n_q}} \wedge C_{k,E}$$

where:

- Each $C_{k,x_i}$ (shortening: $C_{k,i}$) is the conjunction of all the formulas containing only the variable $x_i$, i.e. the ones of the form $a[s^n(x_i)] = e_j$ (remember we got rid of atoms of the form $s^n(x_i) = x_i$ by applying rule 2).
  Define $C_{k,I} := C_{k,1} \wedge \cdots \wedge C_{k,n_q}$.
- $C_{k,E}$ is the conjunction of all the formulas containing two index variables, i.e. the equalities of the form $s^n(x_i) = x_j$.

Let's for a moment analyze $C_{k,E}$. We would like to write it in a nice and treatable form. For example, if it contains both the formulas $s^2(x_1) = x_2$ and $s^4(x_2) = x_3$, we would like to substitute the latter with $s^6(x_1) = x_3$. Let's consider the weighted directed graph $\mathcal{G}$ given by:

- $V = \{1, ..., n_q\}$
- $A = \{(i,j) \mid s^{n_{ij}}(x_i) = x_j \text{ appears in } C_{k,E}\}$
- $w(i,j) = n_{ij}$ for all $(i,j) \in A$. Notation: $i \xrightarrow{n_{ij}} j$ or $j \xrightarrow{-n_{ij}} i$. Moreover, for all $i \in V$ $w(i,i) = 0$.

Note that by using this definition we are assuming that the graph is not a multigraph. This assumption is not yet properly justified, but it will be treated later.

Apply the following algorithm.
*Algorithm "G"*: For every $i, j, k \in V$ such that $i \xrightarrow{n_{ij}} j$ and $j \xrightarrow{n_{jk}} k$

- If there is no edge $(i,k) \in A$, add $i \xrightarrow{n_{ij}+n_{jk}} k$ to $\mathcal{G}$.
- If there is an edge $i \xrightarrow{n_{ik}} k$, check if $n_{ik} = n_{ij} + n_{jk}$. If yes, proceed to another triple. If no, terminate and return as output $|n_{ik} - (n_{ij} + n_{jk})|$.

When the the algorithm has visited every triple without terminating, it returns as output $\mathcal{G}$.

Apply the following rule:

9. Run Algorithm "G".

   - If it returns a positive integer, it means that $\mathcal{G}$ contains a non-zero sum cycle
   $$i_1 \xrightarrow{n_{12}} i_2 \xrightarrow{n_{23}} \cdots \xrightarrow{n_{(m-1)m}} i_m \xrightarrow{n_{m1}} i_1$$
   Returning to the logic notation, this means that $x_{i_1} = s^d(x_{i_1})$, where $d = n_{12} + \cdots + n_{m1}$. Hence, a model of $C_k$ has to have dimension $\leqslant d$. The algorithm tests these finitely many candidate models, and if the output is `unsat`, $C_k$ is replaced by $\bot$ and the algorithm proceeds.
   - If it returns $\mathcal{G}$, every cycle is zero sum. For every (weakly) connected components $K$ of $\mathcal{G}$, there exists at least one vertex (say $r_K$) having no positive ingoing arcs.
   Replace $C_{k,E}$ by $\bigwedge_K (\bigwedge_{i \in K} s^{w(r_K,i)}(x_{r_K}) = x_i)$

*Note*: above we assumed that the graph is not a multigraph. Actually, it could be, but there are only two cases when that happens: either there is repeated atom $s^n(x_i) = x_j$ (and then one of the copies can be removed), or there are two atoms of the form $s^{n_1}(x_i) = x_j$ and $s^{n_2}(x_i) = x_j$ with $n_1 \neq n_2$. This case is analogous to the one where Algorithm "G" returns a positive integer ($|n_1 - n_2|$).

Now, let $M := \max \{n \mid s^n \text{ appears in } A\}$.

Every $C_k$ can be seen as the pair ($C_{k,I}$, $C_{k,E}$), and $C_{k,I}$ can be seen as a $n_q$-tuple of partial functions having domain $\{0, 1, \cdots, M\}$ and codomain the set of the elements $\{1, 2, ..., n_e\}$.

*Example*: suppose $M = 3$, $n_e = 2$, $n_q = 3$ and consider the formula

$$\begin{aligned}
&a[x_1] = 2 \wedge a[s(x_1)] = 1 \wedge a[s^2(x_1)] = 1 \wedge \\
&a[x_2] = 1 \wedge a[s(x_2)] = 1 \wedge a[s^2(x_2)] = 1 \wedge a[s^3(x_2)] = 1 \wedge \\
&a[s(x_3)] = 2 \wedge a[s^2(x_3)] = 2 \wedge a[s^3(x_3)] = 2 \wedge \\
&x_2 = s^2(x_1)
\end{aligned}$$

It can be seen as:

$$\left(\begin{array}{cccc|c} 2 & 1 & 1 & - & - \\ 1 & 1 & 1 & 1 & s^2(x_1) \\ - & 2 & 2 & 2 & - \end{array}\right)$$

which is logically equivalent to the disjunction of "*full*" formulas

$$\left(\begin{array}{cccc|c} 2 & 1 & 1 & \mathit{1} & - \\ 1 & 1 & 1 & 1 & s^2(x_1) \\ \mathit{1} & 2 & 2 & 2 & - \end{array}\right) \vee \left(\begin{array}{cccc|c} 2 & 1 & 1 & \mathit{2} & - \\ 1 & 1 & 1 & 1 & s^2(x_1) \\ \mathit{1} & 2 & 2 & 2 & - \end{array}\right) \vee$$
$$\left(\begin{array}{cccc|c} 2 & 1 & 1 & \mathit{1} & - \\ 1 & 1 & 1 & 1 & s^2(x_1) \\ \mathit{2} & 2 & 2 & 2 & - \end{array}\right) \vee \left(\begin{array}{cccc|c} 2 & 1 & 1 & \mathit{2} & - \\ 1 & 1 & 1 & 1 & s^2(x_1) \\ \mathit{2} & 2 & 2 & 2 & - \end{array}\right)$$

Note that there are *inconsistent* formulas, like the two on the right. In fact, $a[s^4(x_1)] = 2 \wedge a[s^2(x_2)] = 1 \wedge s^2(x_1) = x_2$ implies $a[s^4(x_1)] = 2 \wedge a[s^4(x_1)] = 1$.

Now consider the set $\mathcal{F}$ containing all the formulas $C$ such that $C_I$ is *full*, i.e. for every $i \in \{1, \cdots, n_q\}$ and for every $n \in \{0, \cdots, M\}$, then $C$ contains the term $a[s^n(x_i)]$. Remove from $\mathcal{F}$ the formulas that are inconsistent with $A$. The remaining formluas are called *mosaics*.

Consider all the cycles $C_0, C_1, \cdots, C_n, C_0$ of formulas in $\mathcal{F}$ such that for every $i = 0, \cdots, n$ and $C_i$ and $C_{i+1}$ of the form[8]

$$C_i \equiv \left(\begin{array}{cccc|c} e_{10} & e_{11} & \cdots & e_{1M} & \sim \\ e_{20} & e_{21} & \cdots & e_{2M} & \sim \\ \cdots & \cdots & \cdots & \cdots & \sim \\ e_{n_q 0} & e_{n_q 1} & \cdots & e_{n_q M} & \sim \end{array}\right)$$
$$C_{i+1} \equiv \left(\begin{array}{cccc|c} e'_{10} & e'_{11} & \cdots & e'_{1M} & \sim \\ e'_{20} & e'_{21} & \cdots & e'_{2M} & \sim \\ \cdots & \cdots & \cdots & \cdots & \sim \\ e'_{n_q 0} & e'_{n_q 1} & \cdots & e'_{n_q M} & \sim \end{array}\right)$$

it holds that

$$(e_{11}, e_{12} ..., e_{1M}) = (e'_{10}, e'_{11}, ..., e'_{1(M-1)}) \qquad (\dagger)$$

In order to prove the decidability, it suffices to apply the reasoning of section 8.2.2 that shows a *bound* for the length of these cycles.

Wanting to sketch a more detailed algorithm, we could use the following argument.

Given a cycle, and given the permutation $\sigma_i$ that exchanges $i$ and 1, i.e.

$$\sigma_i(j) := \begin{cases} 1 & \text{if } j = i \\ i & \text{if } j = 1 \\ j & \text{otherwise} \end{cases}, \text{ every formula in the cycle}^9$$

[8] Here, $\sim$ means that the entry can be empty or not

[9] By $\sigma(\phi)$ we indicate the formula obtained from $\phi$ by replacing simultaneously every occurrence of $x_i$ with $x_{\sigma(i)}$

$\sigma_i(C_0), \sigma_i(C_1), \cdots, \sigma_i(C_n), \sigma_i(C_0)$ should be contained in $\mathcal{F}$.

Indeed, if the cycle $C_0, C_1, \cdots, C_n$ is consistent with a model $M$, it means that there exists an instance $\underline{j} = (j_1, \cdots, j_{n_q})$ of $\underline{x}$ such that

$$\begin{aligned} &M \models C_0(\underline{j}) \\ &M \models C_1((j_1 + 1, j_2, \cdots, j_{n_q})) \\ &\cdots \\ &M \models C_n((j_1 + n, j_2, \cdots, j_{n_q})) \end{aligned}$$

Given the instance

$$\sigma_i(\underline{j}) = (j_i, j_2, \cdots, j_{i-1}, j_1, j_{i+1}, \cdots, j_{n_q})$$

it must hold that

$$\begin{aligned} &M \models \sigma_i(C_0)(\sigma_i(\underline{j})) \\ &M \models \sigma_i(C_1)(j_i, j_2, \cdots, j_{i-1}, j_1 + 1, j_{i+1}, \cdots, j_{n_q})) \\ &\cdots \\ &M \models \sigma_i(C_1)(j_i, j_2, \cdots, j_{i-1}, j_1 + n, j_{i+1}, \cdots, j_{n_q})) \end{aligned}$$

This implies that if a formula $C$ is in $\mathcal{F}$ but there exists $i$ such that $\sigma_i(C)$ is not in $\mathcal{F}$, then $C$ cannot be contained in any consistent cycle, hence it can be removed from $\mathcal{F}$.

After that, every remaining cycle $C_0, C_1, \cdots, C_n$ is such that for every $i \in \{2, \cdots, n_q\}$, also the cycle $\sigma_i(C_0), \sigma_i(C_1), \cdots, \sigma_i(C_n), \sigma_i(C_0)$ remained.

Hence, for every remaning cycle there is a *permuted* cycle for every *direction* $i = 1, 2, \cdots, n_q$.

*Example.* Let $n_q = 2$ and $M = 2$. Suppose we found the cycle

$$\left(\begin{array}{cc|c} 1 & 2 & \sim \\ 1 & 2 & \sim \end{array}\right), \left(\begin{array}{cc|c} 2 & 3 & \sim \\ 1 & 2 & \sim \end{array}\right), \left(\begin{array}{cc|c} 3 & 1 & \sim \\ 1 & 2 & \sim \end{array}\right), \left(\begin{array}{cc|c} 1 & 2 & \sim \\ 1 & 2 & \sim \end{array}\right),$$

For $\sigma_2$ we have an other cycle

$$\left(\begin{array}{cc|c} 1 & 2 & \sim \\ 1 & 2 & \sim \end{array}\right), \left(\begin{array}{cc|c} 1 & 2 & \sim \\ 2 & 3 & \sim \end{array}\right), \left(\begin{array}{cc|c} 1 & 2 & \sim \\ 3 & 1 & \sim \end{array}\right), \left(\begin{array}{cc|c} 1 & 2 & \sim \\ 1 & 2 & \sim \end{array}\right),$$

We can see these cycles as sides of a square (in the general case, of a $n_q$-cube):

$$\left(\begin{array}{cc|c} 1 & 2 & \sim \\ 1 & 2 & \sim \end{array}\right), \left(\begin{array}{cc|c} 2 & 3 & \sim \\ 1 & 2 & \sim \end{array}\right), \left(\begin{array}{cc|c} 3 & 1 & \sim \\ 1 & 2 & \sim \end{array}\right), \left(\begin{array}{cc|c} 1 & 2 & \sim \\ 1 & 2 & \sim \end{array}\right)$$
$$\left(\begin{array}{cc|c} 1 & 2 & \sim \\ 2 & 3 & \sim \end{array}\right),$$
$$\left(\begin{array}{cc|c} 1 & 2 & \sim \\ 3 & 1 & \sim \end{array}\right),$$
$$\left(\begin{array}{cc|c} 1 & 2 & \sim \\ 1 & 2 & \sim \end{array}\right),$$

and then check if the new formulas included in its *completition* (below in italics) belong to $\mathcal{F}$.

$$\left(\begin{array}{cc|c} 1 & 2 & \sim \\ 1 & 2 & \sim \end{array}\right), \left(\begin{array}{cc|c} 2 & 3 & \sim \\ 1 & 2 & \sim \end{array}\right), \left(\begin{array}{cc|c} 3 & 1 & \sim \\ 1 & 2 & \sim \end{array}\right), \left(\begin{array}{cc|c} 1 & 2 & \sim \\ 1 & 2 & \sim \end{array}\right)$$
$$\left(\begin{array}{cc|c} 1 & 2 & \sim \\ 2 & 3 & \sim \end{array}\right), \left(\begin{array}{cc|c} \mathit{2} & \mathit{3} & \sim \\ \mathit{2} & \mathit{3} & \sim \end{array}\right), \left(\begin{array}{cc|c} \mathit{3} & \mathit{1} & \sim \\ \mathit{2} & \mathit{3} & \sim \end{array}\right), \left(\begin{array}{cc|c} \mathit{1} & \mathit{2} & \sim \\ \mathit{2} & \mathit{3} & \sim \end{array}\right)$$
$$\left(\begin{array}{cc|c} 1 & 2 & \sim \\ 3 & 1 & \sim \end{array}\right), \left(\begin{array}{cc|c} \mathit{2} & \mathit{3} & \sim \\ \mathit{3} & \mathit{1} & \sim \end{array}\right), \left(\begin{array}{cc|c} \mathit{3} & \mathit{1} & \sim \\ \mathit{3} & \mathit{1} & \sim \end{array}\right), \left(\begin{array}{cc|c} \mathit{1} & \mathit{2} & \sim \\ \mathit{3} & \mathit{1} & \sim \end{array}\right)$$
$$\left(\begin{array}{cc|c} 1 & 2 & \sim \\ 1 & 2 & \sim \end{array}\right), \left(\begin{array}{cc|c} \mathit{2} & \mathit{3} & \sim \\ \mathit{1} & \mathit{2} & \sim \end{array}\right), \left(\begin{array}{cc|c} \mathit{3} & \mathit{1} & \sim \\ \mathit{1} & \mathit{2} & \sim \end{array}\right), \left(\begin{array}{cc|c} \mathit{1} & \mathit{2} & \sim \\ \mathit{1} & \mathit{2} & \sim \end{array}\right)$$

If there is a new formula not in $\mathcal{F}$, the cycle is discarded. If there is any formula $C$ (new or old) such that $C_E$ does not match with its position in the hypercube (e.g. if in the position $(0,1)$ there is the formula $\left(\begin{array}{cc|c} 2 & 3 & - \\ 1 & 2 & s^2(x_1) \end{array}\right)$) then the cycle is discarded. If every cycle is discarded, the algorithm terminates with an `unsat` output; otherwise if a cycle is not discarded, it defines a model and the algorithm terminates with a `sat` output.

# Bibliography


[1] Benjamin Aminof et al. *Parameterized Model Checking of Token-Passing Systems*. Ed. by Kenneth L. McMillan and Xavier Rival. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 262–281. ISBN: 978-3-642-54013-4. DOI: 10.1007/978-3-642-54013-4_15.

[2] Roderick Bloem et al. *Decidability of Parameterized Verification*. Synthesis Lectures on Distributed Computing Theory. Morgan Claypool Publishers, 2015. ISBN: 1627057439.

[3] Michael C. Browne, Edmund M. Clarke, and Orna Grumberg. " Reasoning about networks with many identical finite state processes". In: *Information and Computation* 81 (01 1989), pp. 13–31. DOI: 10.1016/0890-5401(89)90026-6.

[4] Edmund Clarke et al. "Verification by Network Decomposition". In: (2004). Ed. by Philippa Gardner and Nobuko Yoshida, pp. 276–291. DOI: 10.1007/978-3-540-28644-8_18.

[5] E. Allen Emerson and Kedar S. Namjoshi. "Reasoning about Rings." In: (1995). Ed. by Ron K. Cytron and Peter Lee, pp. 85–94.

[6] E. Allen Emerson and Kedar S. Namjoshi. "On reasoning about rings". In: *International Journal of Foundations of Computer Science* 14.04 (2003), pp. 527–549. DOI: 10.1142/S0129054103001881.

[7] Silvio Ghilardi and Silvio Ranise. "Backward Reachability of Array-based Systems by SMT solving: Termination and Invariant Synthesis". In: *Logical Methods in Computer Science* Volume 6, Issue 4 (Dec. 2010). DOI: 10.2168/LMCS-6(4:10)2010. URL: https://lmcs.episciences.org/966.

[8] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, eds. *Automata Logics, and Infinite Games: A Guide to Current Research*. Berlin, Heidelberg: Springer-Verlag, 2002. ISBN: 3540003886. DOI: 10.5555/938135.

[9] Nancy Lynch. *Distributed Algorithms*. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, 1996. ISBN: 9781558603486.

[10] R. Milner. *Communication and Concurrency*. USA: Prentice-Hall, Inc., 1989. ISBN: 0131150073.

[11] Klaus Schneider. *Verification of Reactive Systems*. Jan. 2004, p. 428. DOI: 10.1007/978-3-662-10778-2.