# Abstract

# Contents

CHAPTER 1

# Introduction

## 1.1 TARGET AUDIENCE

Graduate students who have taken basic courses in discrete mathematics, data structures, logic and computation; who want to add to their toolbox a powerful, versatile, tool

CHAPTER 2

# Formal Methods

## 2.1 VERIFICATION

## 2.2 SYNTHESIS

## 2.3 MONITORING

Testing?

## 2.4 WHAT IS NOT IN THIS BOOK

??

CHAPTER 3

# Algorithmic Foundations: Decision problems, Automata theory

CHAPTER 4

# Foundations: Logics for Verification

CHAPTER 5

# Limitations of the method

CHAPTER 6

# History of the method

CHAPTER 7

# Inverview? Biography of the inventors

# CHAPTER 8

# tex

### 8.0.1 AUTOMATA THEORY BACKGROUND

We use alternating Büchi automata over infinite trees, written ABT. All results can be found in **?**, although we note that here we use slight notational variants of the definitions.[1]

Let $B^+(X)$ denote the set of positive Boolean formulas over $X$, i.e., it consists of the constants $\mathsf{true}, \mathsf{false}$, and the formulas from the set containing the elements of $X$ closed under the operations $\vee$ and $\wedge$. Let $\Sigma$ be an alphabet, and let $D$ be a finite non-empty set of *directions*. A $D$-ary $\Sigma$-tree is a function $t : D^+ \to \Sigma$.

Tree automata recognised sets of $D$-ary $\Sigma$-trees. An *alternating Büchi tree automaton (*ABT*)* over $\Sigma$ and $D$ is a tuple $T = (Q, \iota, \delta, B)$ where $Q$ is a finite non-empty set of states, $\iota \in B^+(D \times Q)$ is the initial formula, $\delta : Q \times \Sigma \to B^+(D \times Q)$ is the transition function, and $B \subseteq Q$ is the set of *accepting* states. The set $F = \{\iota\} \cup \{\delta(q, \sigma) : q \in Q, \sigma \in \Sigma\}$ is called the set of *formulas* of $T$.

Since it is intuitive and concise, we give a game-theoretic semantics of what it means for an ABT to accept or reject a tree. An ABT $T$ accepts tree $t$ if player Automaton has a winning strategy in the following "membership game" played against player Pathfinder. Positions of the game are elements of $D^+ \times Q$. From a position $(u, q)$ player Automaton picks (if he can) a set $S \subseteq D \times Q$ satisfying $\delta(q, t(u))$; if there is no such set Automaton loses the game, if the set is empty then Automaton wins, and otherwise Pathfinder picks a position $(d', q') \in S$ and the position is updated to $(ud', q')$. To start the game off Automaton picks $S$ satisfying $\iota$ and Pathfinder picks a position $(d, q) \in S$. If the game lasts infinitely many rounds it generates an infinite sequence $(d_0, q_0)(d_0d_1, q_1)(d_0d_1d_2, q_2)\cdots$ (the sequence can be thought of as a path through $t$ labeled by states from $Q$). In this case Player Automaton wins iff there exists infinitely many $n \in \mathbb{N}$ such that $q_n \in B$.

A *nondeterministic Büchi tree automaton (*NBT*)* is an ABT whose formulas are either of the form $\mathsf{true}$, $\mathsf{false}$, or $\bigvee_i \bigwedge_{d \in D}(d, q_{i,d})$, i.e., they are in DNF in which every conjunct contains every direction exactly at most? once. A *universal Büchi tree automaton (*UBT*)* is an ABT in which none of the formulas contain disjunction. A *deterministic Büchi tree automaton (*DBT*)* is an NBT that is also a UBT, i.e., every formula that is not $\mathsf{true}$ or $\mathsf{false}$ is of the form $\bigwedge_{d \in D}(d, q_d)$.

[1] Our trees $t$ have the unconvential but convenient property that their domains do not contain the empty-string. This leads to the need for an initial formula rather than the more common an initial state.

We define *alternating co-Büchi tree automata (*ACT*)* like ABT but dualise the acceptance condition, i.e., player Automaton wins iff there are only finitely many $n \in \mathbb{N}$ such that $q_n \in B$.

A $D$-ary $\Sigma$-tree $t$ is *regular* if it is determined by a transducer with input alphabet $D$ and output alphabet $\Sigma$.

define size of automaton.

**Fact 1 (Dualising** ABT**)** *Fix $\Sigma, D$. The* dual *of an* ABT *$T = (Q, \iota, \delta, B)$ is the* ACT *$T^\partial = (Q, \iota^\delta, \delta^\partial, B)$ where $\delta^\partial(q, \sigma) := \delta(q, \sigma)^\partial$ and where $\theta^\partial$ is the transformation that swaps $\vee$ with $\wedge$ and swaps* true *with* false*, i.e.,* $\mathsf{true}^\partial \doteq \mathsf{false}$*,* $\mathsf{false}^\partial \doteq \mathsf{true}$*,* $(\theta_1 \wedge \theta_2)^\partial \doteq \theta_1^\partial \vee \theta_2^\partial$ *and* $(\theta_1 \vee \theta_2)^\partial \doteq \theta_1^\partial \wedge \theta_2^\partial$*. It follows from the definitions that, for all trees $t$, $T$ accepts $t$ if and only if $T^\partial$ does not accept $t$. The size of $T^\partial$ is the same as the size of $T$.*

**Fact 2 (Intersection of** ABT**)** *Fix $\Sigma, D$ and* ABT *$T_i = (Q_i, \iota_i, \delta_i, B_i)$ for $i = 1, 2$. Then the following* ABT *accepts those trees $t$ that are accepted by both $T_1$ and $T_2$: it is the disjoint union of $T_1$ and $T_2$ with initial formula $\iota_1 \wedge \iota_2$.*

**Fact 3 (Emptiness of** ABT**)** *Emptiness of* ABT *is decidable in* EXPTIME*, i.e., $2^{poly(n)}$ where $n$ is the number of states of the* ABT *and poly is a fixed polynomial. If the* ABT *$T$ is non-empty, the procedure can also return a regular tree in $T$.*

### 8.0.2 AUTOMATA WITH FINITARY ACCEPTANCE

An ABT $(Q, \iota, \delta, B)$ has *finitary acceptance condition* if $\delta(b, \sigma) = b$ for all $b \in B, \sigma \in \Sigma$. In particular, in the membership game, player Automaton wins an infinite play $\pi$ iff there exists $n \in \mathbb{N}$ such that $\pi_n \in B$, i.e., a reachability condition. We write AFT for these automata.

WARNING: We need better notation. AFT usually means input trees are FINITE.

**Fact 4 (Intersection of** NFT**)** *Fix $\Sigma, D$ and* NFT *$T = (Q, \iota, \delta, B)$ and $T' = (Q', \iota', \delta', B')$. Without loss of generality:*

- *Assume $B = B' = \emptyset$ (this can be done redefining $\delta$ to map $q \in B$ and $\sigma \in \Sigma$ to* true*).*[2]
- *Assume that $Q, Q'$ are disjoint.*

*Let $S \doteq (Q \times Q') \cup Q \cup Q'$. For $\theta = \bigvee_{x \in X} \bigwedge_{d \in D} (d, q_{x,d}) \in B^+(D \times Q)$ and $\theta' = \bigvee_{y \in Y} \bigwedge_{d \in D} (d, q_{y,d}) \in B^+(D \times Q')$, define $\theta \otimes \theta' \in B^+(D \times S)$ as follows:*

- $\theta \otimes \mathsf{false} \doteq \mathsf{false}$ *and* $\mathsf{false} \otimes \theta \doteq \mathsf{false}$*,*
- $\theta \otimes \mathsf{true} \doteq \theta$ *and* $\mathsf{true} \otimes \theta' \doteq \theta'$*,*
- $\theta \otimes \theta' \doteq \bigvee_{(x,y) \in X \times Y} \bigwedge_{d \in D} (d, \langle q_{x,d}, q_{y,d} \rangle)$.

[2] We remark that this simplification is not valid for $\mathrm{DFT}_f$ since the automaton would have to guess the end of the branch (although it would be valid if we assumed end-of-branch markers to all input trees).

*Then the following* NFT *accepts those trees $t$ that are accepted by both $T_1$ and $T_2$: its states are $S$, its initial state is $\iota \otimes \iota'$, and its transition function on input $\sigma$ sends state $\langle q, q' \rangle$ to formula $\delta(q,\sigma) \otimes \delta'(q',\sigma)$. This construction needs to be checked/debugged*

Just as the acceptance condition of a tree automaton can be viewed as a "membership game", so too the emptiness check can be viewed as an "emptiness game". For NFT this means that classic fixpoint algorithms (used for solving games) can be applied to solve the emptiness problem of tree automata, see, e.g., **?**.

**Fact 5 (Emptiness of NFT)** *Emptiness of* NFT *is decidable in linear time. Moreover, if an* NFT $T$ *is non-empty, one can also return a regular tree in $T$. Can be cut... Emptiness of* NFT $N$ *is equivalent to deciding if Player Automaton has a winning strategy in the following two-player zero-sum game of perfect information played on $N$. The adversary is called Pathfinder. Play goes as follows: from a state $q$ Player Automaton picks a symbol $\sigma$ and a conjunct $\bigwedge_{d \in D}(d, q_d)$ in $\delta(q,\sigma)$, and player Pathfinder picks $d \in D$, and play proceeds to state $q_d$; play starts with Automaton picking a disjunct in $\iota$ and Pathfinder picking the next direction and state; if $\delta(q,\sigma) = \mathsf{false}$ then Automaton loses, and if $\delta(q,\sigma) = \mathsf{true}$ then Automaton wins.*

**Automata for finite sequences** $\text{NFW}_f$. We only need the definition of nondeterministic word automaton. For conciseness, we reuse our existing definitions.

Formally, a *finite* $\Sigma$*-word* is a partial function $t : D^+ \to \Sigma$ with $|D| = 1$, whose domain is finite and closed under taking non-empty prefixes. Define $\text{NFW}_f$ to be like an NFT except that it operates on finite words, and the membership game has the following additional rule: from a position $(u, q)$, if the word has ended, then Automaton wins iff $q \in B$. Thus, $\text{NFW}_f$ is simply a notational variants of the usual nondetermistic word automata (sometimes denoted NFA in the literature).

## 8.1 PRELIMINARIES

An *alphabet* is a finite non-empty set $X$ of symbols, e.g., $\{0, 1, 2\}$. Let $X^+$ (resp. $X^*$) denote the set of non-empty (resp. possibly empty) finite sequences over alphabet $X$. The main technical objects of this paper are functions of the form $X^+ \to Y$ for alphabets $X, Y$. Such functions represent strategies in game-theory and trees in automata-theory.

Sequences are indexed starting at 0, thus we write $x = x_0 x_1 \cdots$.

### 8.1.1 LINEAR-TIME TEMPORAL LOGIC (LTL)

Fix a finite non-empty set $AP$ of atomic propositions. The *formulas of* LTL *(over $AP$)* are generated by the following grammar: $\varphi ::= p \mid \varphi \vee \varphi \mid \neg\varphi \mid \mathsf{X}\varphi \mid \varphi \,\mathsf{U}\, \varphi$ where $p \in AP$.

Formulas are interpreted over infinite traces $\alpha \in (2^{AP})^\omega$. Define the satisfaction relation $\models$ as follows:

1. $(\alpha, n) \models p$ iff $p \in \alpha_n$;
2. $(\alpha, n) \models \varphi_1 \vee \varphi_2$ iff $(\alpha, n) \models \varphi_i$ for some $i \in \{1, 2\}$;
3. $(\alpha, n) \models \neg\varphi$ iff it is not the case that $(\alpha, n) \models \varphi$;
4. $(\alpha, n) \models \mathsf{X}\,\varphi$ iff $(\alpha, n+1) \models \varphi$;
5. $(\alpha, n) \models \varphi_1 \,\mathsf{U}\, \varphi_2$ iff there exists $i \geq n$ such that $(\alpha, i) \models \varphi_2$ and for all $i \leq j < n$, $(\alpha, j) \models \varphi_1$.

Write $\alpha \models \varphi$ if $(\alpha, 0) \models \varphi$ and say that $\alpha$ *satisfies* $\varphi$ and that $\alpha$ is a *model* of $\varphi$. We use the following abbreviations, $\varphi \rightarrow \varphi' \doteq \neg\varphi \vee \varphi'$, $\mathsf{true} := p \vee \neg p$, $\mathsf{false} \doteq \neg\mathsf{true}$, $\mathsf{F}\,\varphi \doteq \mathsf{true}\,\mathsf{U}\,\varphi$, and $\mathsf{G}\,\varphi \doteq \varphi\,\mathsf{U}\,\mathsf{false}$.

### 8.1.2 REACTIVE REALIZABILITY AND SYNTHESIS

Reactive Synthesis is the problem of producing a finite-state reactive module that satisfies a given property no matter how the environment behaves. For the most part we follow the notation in **?**.[3] The set of propositions $AP$ is partitioned into two sets: those controllable by the agent $AP_{\mathsf{ag}}$, and those not controllable by the agent $AP_{\mathsf{env}}$. Let $\Sigma_{\mathsf{ag}} = 2^{AP_{\mathsf{ag}}}$ and $\Sigma_{\mathsf{env}} = 2^{AP_{\mathsf{env}}}$ be the corresponding sets of *assignments*, and let $\Sigma = \Sigma_{\mathsf{ag}} \cup \Sigma_{\mathsf{env}}$. A *reactive module* or *agent-strategy* is a function $\sigma : (\Sigma_{\mathsf{env}})^+ \rightarrow \Sigma_{\mathsf{ag}}$. Say that $\sigma$ is *finite-state* if it is determined by a transducer with input alphabet $\Sigma_{\mathsf{env}}$ and output alphabet $\Sigma_{\mathsf{ag}}$.[4] A sequence $\pi = \pi_1\pi_2\ldots$ is *consistent with agent-strategy* $\sigma$ if for every $k \geq 1$, $\pi_k \cap \Sigma_{\mathsf{ag}} = \sigma((\pi_1 \cap \Sigma_{\mathsf{env}}) \cdots (\pi_k \cap \Sigma_{\mathsf{env}}))$.

In LTL Reactive Synthesis the full specification is given as a single LTL formula $\varphi$.

**Definition 8.1** LTL **Realisability.** Given an LTL-formula $\varphi$ over alphabet $\Sigma = \Sigma_{\mathsf{ag}} \cup \Sigma_{\mathsf{env}}$, decide if there exists an agent-strategy $\sigma$ such that every play $\pi$ consistent with $\sigma$ satisfies $\varphi$. In this case, say that $\sigma$ *realises* $\varphi$, and write $\sigma \rhd \varphi$.

**Definition 8.2** LTL **Synthesis.** Assume that $\varphi$ is realisable. The synthesis problem asks to return a finite-state agent-strategy realising $\varphi$.

**Theorem 8.3** LTL *realizability is* 2EXPTIME*-complete, and synthesis can be solved in* 2EXPTIME. *Does it make sense to talk about synthesis being complete?*

**Notation.** From now on we drop the adjective "Reactive" and just say, e.g., "LTL Synthesis" instead of "LTL Reactive Synthesis".

[3] As in the modern literature, we have conveniently simplified the formulation in **?** by considering the predicates for each player to be Boolean variables, rather than predicates over terms over static and dynamic variables.

[4] A *transducer* is a deterministic finite-state machine that is fed symbols from an input alphabet $I$, and symbol by symbol, it produces a symbol from an output alphabet $O$, and changes its internal state. A transducer determines a function $I^+ \rightarrow O$. Formalise this more?

### 8.1.3 ALGORITHM AND COMPLEXITY

We now give the building-blocks that will be used in our Algorithm.

**Proposition 8.4** **From LTL to NBW** *For every LTL formula $\psi$ we can build an NBW $N_\psi$ of size $2^{O(|\psi|)}$ that accepts the models of $\psi$* **?**.

**Lemma 8.5** *Let $\phi$ be an LTL formula. There is an NBT $T_\exists$ of size $2^{O(|\phi|)}$ that accepts all strategies $\sigma$ such that $\sigma \not\triangleright \neg\phi$.*

***Proof.*** We use the following fact: For every NBW $N$ there is an NBT $T$ of size linear in $N$ that accepts all strategies $\sigma$ such that some play consistent with $\sigma$ is labeled by an infinite word accepted by $N$. To see this, the NBT $T$ guesses the path and runs the NBW $N$ on that path, guessing the accepting run of $N$. That is, given $N = (Q, \iota, \delta, B)$ define $T = (Q, \iota', \delta', B)$ by replacing every formula $\bigvee_i q_i$ of $N$ by $\bigvee_i \bigvee_{d\in D}(d, q_i)$.

Apply Proposition 8.4 to $\phi$ to build an NBW $N_\phi$ that accepts the models of $\phi$. By the fact, build an NBT that accepts $\sigma$ iff some play consistent with $\sigma$ satisfies $\phi$, i.e., $\sigma \not\triangleright \neg\phi$. □

We now give the dual lemma.

**Lemma 8.6** *Let $\phi$ be an LTL formula. There is an UCT $T_\forall$ of size $2^{O(|\phi|)}$ that accepts all strategies $\sigma$ such that $\sigma \triangleright \phi$.*

***Proof.*** Apply Lemma 8.5 to $\neg\phi$ and build an NBT that accepts all strategies $\sigma$ such that $\sigma \not\triangleright \phi$. Dualise the NBT (by Fact 1) to get a UCT that accepts the complement, i.e., all strategies $\sigma$ such that $\sigma \triangleright \phi$. □

We are ready to give an algorithm for the upper bound of Theorem **??**.

1. Build NBT $T_1$ by Lemma 8.5 applied to $\Phi_{\mathsf{asmp}}$.
2. Build UCT $T_2$ by Lemma 8.6 applied to $\Phi_{\mathsf{asmp}} \rightarrow \Phi_{\mathsf{goal}}$.
3. Build ABT $T$ by Fact 2 as the intersection of $T_1$ and $T_2$.
4. Test the emptiness, by Fact 3, of the ABT $T$, and if non-empty return a finite-state strategy.

This algorithm constructs an ABT $T$ that accepts exactly the strategies $\sigma$ such that i) $\sigma \not\triangleright \neg\Phi_{\mathsf{asmp}}$ and ii) $\sigma \triangleright \Phi_{\mathsf{asmp}} \rightarrow \Phi_{\mathsf{goal}}$. Indeed, the NBT $T_1$ accepts the $\sigma$ that satisfy i) and the UCT $T_2$ accepts the $\sigma$ satisfying ii). The size of $T$ is $|T_1| + |T_2| = 2^{O(|\Phi_{\mathsf{asmp}}|+|\Phi_{\mathsf{goal}}|)}$. By Fact 3, testing for $T$'s emptiness results in a total cost of $2^{2^{O(|\Phi_{\mathsf{asmp}}|+|\Phi_{\mathsf{goal}}|)}}$.

### 8.1.4 ALGORITHM AND COMPLEXITY

The following is proved in **?**, and is the finitary version of Proposition 8.4.

**Proposition 8.7** *For every* LTLf *formula* $\phi$ *there is a* NFW$_f$ $N_\phi$ *of size* $2^{O(|\phi|)}$ *that accepts the models of* $\phi$.

We now refine Lemmas 8.5 and 8.6 for finite sequences. We remark that strategies are still infinite trees, and thus we will use NFT and DFT to define sets of strategies. The first lemma is proved just as Lemma 8.5 but replacing Proposition 8.4 by Proposition 8.7.

**Lemma 8.8** *Let* $\phi$ *be an* LTLf *formula. There is an* NFT $T_\exists$ *of size* $2^{O(|\phi|)}$ *that accepts all strategies* $\sigma$ *such that* $\sigma \not\rhd \neg\phi$.

The second lemma is similar to Lemma 8.6 except that we determinise the formula $\varphi$ (paying the extra exponent now, rather than later).

**Lemma 8.9** *Let* $\phi$ *be an* LTLf *formula. There is an* DFT $T_\forall$ *of size* $2^{2^{O(|\phi|)}}$ *that accepts all strategies* $\sigma$ *such that* $\sigma \rhd \phi$. *Note that* $T_\forall$ *is deterministic.*

### 8.1.5 ALGORITHM AND COMPLEXITY

Here is the algorithm for the upper bound in Theorem **??**.

1. Apply Lemma 8.8 to formula $\Phi_{\mathsf{asmp}}$ to get NFT $T_1$.
2. Apply Lemma 8.9 to formula $\Phi_{\mathsf{asmp}} \rightarrow \Phi_{\mathsf{goal}}$ to get DFT $T_2$.
3. Apply Fact 4 to get an NFT $T$ whose language is the intersection of the languages of $T_1$ and $T_2$.
4. Use Fact 5 to test the emptiness of the NFT $T$, and if non-empty, return a regular tree which encodes the desired finite-state ag-strategy.

The size of $T$ is $|T_1| \times |T_2|$, and emptiness of $T$ is linear in $|T|$; thus the total cost of this algorithm is $2^{2^{O(|\Phi_{\mathsf{asmp}}|+|\Phi_{\mathsf{goal}}|)}}$.

# Bibliography